WAFL
Funkcionalni programski jezik za razvoj Veb aplikacija

Implementiran podskup

-

verzija jezika 0.3.9.1

verzija implementacije 0.3.0.0

NAPOMENE

Programski jezik WAFL nastao je kao rezultat istraživanja u oblastima funkcionalnih programskih jezika i razvoja Veb aplikacija i dinamičkih Veb lokacija. Odlikuju ga neka od svojstava uobičajenih za savremene funkcionalne programske jezike i prilagođenost domenu Veb aplikacija.

U ovom tekstu je predstavljen deo programskog jezika WAFL koji je već implementiran, kao i neki elementi koji će biti implementirani u bližoj budućnosti. Puna definicija programskog jezika i predstavljanje okolnosti koje su uticale na njegovu formu i funkcionalnost biće objavljeni u vidu magistarskog rada "WAFL – Funkcionalni programski jezik za razvoj Veb aplikacija".

Prvo poglavlje uvodi neke osnovne koncepte i ciljeve oblikovanja programskog jezika WAFL. U drugom poglavlju opisani su elementi programskog jezika WAFL. Radi lakšeg sagledavanja nekih elemenata programskog jezika, u dodacima su priloženi: formalna definicija sintakse, sumarni pregled standardne biblioteke, način imenovanja programskih datoteka i definisanja parametara servisa kao i opis tabela baze podataka koja je upotrebljavana u primerima. Rečnik pojmova i Indeks pružaju čitaocu dopunske informacije o najvažnijim pojmovima koji se pojavljuju u ovom radu, efikasnije snalaženje u tekstu. Na kraju je priložen i spisak literature korišćene pri definisanju programskog jezika WAFL.

U slučaju pronalaženja grešaka ili neusaglašenosti implementacije i ovog teksta, kao i u slučaju da imate neke primedbe ili sugestije, molimo da svoja zapažanja dostavite autoru.

Beograd, decembar 2001. godine

Saša Malkov
smalkov@matf.bg.ac.yu

Sadržaj

Spisak tabela

1. Uvod

Sagledavanjam osnovnih karakteristika problema razvoja Veb aplikacija i dinamičkih Veb lokacija i pregledom osnovnih karakteristika predstavljenih rešenja može se utvrditi da većina rešenja ne zadovoljava u potpunosti iskazane potrebe. Nasuprot tome, pregled osnovnih osobina funkcionalnih programskih jezika stvara utisak da bi njihova primena mogla dovesti do pozitivnih efekata. Na osnovu izvedenih analiza, mogu se definisati karakteristike koje bi programski jezik trebalo da ima da bi zadovoljio iskazane potrebe u oblasti razvoja Veb aplikacija. Zadatak koji je autor ovog rada sebi postavio je:

Napraviti programski jezik za razvoj Veb aplikacija koji zadovoljava navedene ciljeve.

1.1. Osnovni principi

Na osnovu sagledanih karakteristika problema razvoja dinamičkih Veb lokacija i karakteristika postojećih razvojnih rešenja, definisane su poželjne karakteristike programskog jezika za razvoj Veb lokacija.

Polazni kriterijum pri određivanju ciljnih karakteristika programskog jezika čine dva osnovna pricnipa:

1. **Izbor funkcionalne paradigme**.
   Prepoznate karakteristike problema i osnovna svojstva funkcionalnih programskih jezika su u velikoj meri kompatibilni. Odsustvo interaktivnosti iz postupka generisanja pojedinačnih stranica čini da se on može prirodno predstaviti funkcijom, koja preslikava klijentski zahtev u stranicu lokacije koja predstavlja odgovor.
2. **Prilagođavanje problemu**.
   Koliko god da je poželjno da ciljni jezik bude generalno primenjiv, neophodno je da bude posebno pogodan za razvoj dinamičkih Veb lokacija. Poželjno je da prilagođenost programskog jezika problemu bude u većoj meri realizovana odgovarajućim zasnivanjem osnovnih elemenata jezika, a u manjoj meri ugradnjom dodatnih elemenata jezika.

1.2. Opšte ciljne karakteristike

Pregled ciljnih karakteristika programskog jezika podeljen je na dve celine. Najpre sledi pregled opštih poželjnih karakteristika ciljnog funkcionalnog programskog jezika, a zatim slede i specifične karakteristike koje u izvesnoj meri odražavaju specifičnost problema kome je jezik prilagođen.

Opšte ciljne karakteristike programskog jezika su:

1. **Opšta primenjivost**.
   Idealno bi bilo da programski jezik ne ograničava programera u bilo kom smislu i da omogućava da se na njemu napiše proizvoljan program. Neke karakteristike funkcionalnih programskih jezika čine ih manje podobnim, u odnosu na imperativne programske jezike, za određene primene kao što su pisanje delova korisničkog interfejsa ili nekih sistemskih programa. Nezavisno od toga, ne bi trebalo dodavati nova ograničenja.
2. **Slobodan izbor arhitekture aplikacije**.
   Razvojni alat, pa ni programski jezik, nipošto ne bi smeo ograničavati korisnika pri izboru najpogodnije arhitekture za neku konkretnu aplikaciju.
3. **Izražajnost**.
   Upotrebljivost programskog jezika zavisi, pored ostalih činilaca, i neposredno od njegove izražajnosti, tj. od mogućnosti da se što kraćim zapisom opiše neki složen algoritam[1]. Kako je izražajnost prilično subjektivna kategorija [Ghez1997], načini za njeno formalno ustanovljavanje su prilično složeni [Booth1996].
4. **Čitljivost**.
   Programski jezik mora omogućavati što lakše praćenje logike programa i uočavanje grešaka u tekstu programa. I čitljivost je subjektivna kategorija, ali se može smatrati da se postiže omogućavanjem definisanja struktura podataka i programa.
5. **Bezbednost**.
   Programski jezik ne bi trebalo da obuhvati mogućnosti koje dopuštaju pisanje "ranjivih" programa [Ghez1997]. Na primer, programski jezik koji nema naredbu `goto` i pokazivačke promenljive eliminiše dva najčešća izvora teško uočljivih grešaka. Na nesreću, mogućnosti koje smanjuju opasnosti, često umanjuju i snagu i fleksibilnost programskog jezika.
6. **Kompozitni tipovi podataka**.
   Neke osnovne kompozitne tipove je poželjno ugraditi u jezik, dok se podrška za ostale može obezbediti u vidu biblioteke.
7. **Modularnost**.
   Omogućavanjem podele razvoja programa na više modula pojednostavljuju se organizacija i raspodela posla u fazi razvoja i održavanje lokacije. Modularnost je jedan od ključnih preduslova za uspešno programiranje [Hugh1990].

[1] Izražajnost u smislu mogućnosti pisanja programa za svaki izračunljiv problem postignuta je obuhvaćenim osobinama funkcionalnih programskih jezika.

8. **Proširivost**.
   Proširenja mogućnosti programskog jezika mogu da se realizuju kroz biblioteke potprograma. Izdvajanjem često upotrebljavanih segmenata programa u biblioteke omogućava se brži razvoj i jednostavnije održavanje. Istovremeno, dodavanje bibliotečkih modula pruža osnove za prilagođavanje okruženja konkretnim potrebama.
9. **Otvorenost prema drugim programskim jezicima**.
   Mogućnost upotrebe biblioteka napisanih na drugim programskim jezicima, kao što su C i C++, predstavlja korak dalje u proširivanju okruženja, jer se time omogućava da se najvažniji elementi biblioteke razvijaju upotrebom najefikasnijih tehnologija.
10. **Apstraktni tipovi podataka**.
    Obezbeđivanjem jednog skupa ugrađenih apstraktnih tipova podataka i obezbeđenjem mogućnosti za definisanje novih apstraktnih tipova podataka, omogućava se pravljenje intuitivnijih modela problema i programa, koje je jednostavnije projektovati i održavati.
11. **Standardni interfejs prema bazama podataka**.
    Obezbeđivanjem jednog standardnog interfejsa prema bazama podataka isključuje se potreba za učenjem više različitih interfejsa i obezbeđuju se jednostavnost upotrebe i prenosivost programa. Poželjno je dati prednost upotrebi baza podataka u odnosu na datoteke.
12. **Rad sa datotekama**.
    Interfejs prema datotekama je neophodan za većinu ozbiljnih primena. Iako u okviru konkretnih definisanih ciljeva ima manji značaj od rada sa bazama podataka, rad sa datotekama se nipošto ne sme zanemariti.
13. **Implicitna statička provera tipova**.
    Provera tipova u fazi prevođenja programa predstavlja veoma važan element u blagovremenom prepoznavanju grešaka. Uvođenje implicitne provere tipova, uz automatsko razrešavanje umesto navođenja u tekstu programa, pojednostavljuje pisanje i omogućava jednostavno pisanje polimorfnih funkcija i tipova podataka.
14. **Sintaksna sličnost sa programskim jezicima C, C++, Java i JavaScript.**
    Veliki deo programerske populacije povremeno ili redovno upotrebljava neki od programskih jezika C, C++, Java i JavaScript. Da bi im se olakšalo učenje novog programskog jezika, poželjno je da on bude sintaksno sličan tim programskim jezicima. Iako je semantička razlika ogromna, jer je jedan od usvojenih osnovnih principa da novi programski jezik bude funkcionalan, poželjno je da se bar na nivou zapisa izraza postigne što veća sličnost.
15. **Konceptualna sličnost sa savremenim funkcionalnim programskim jezicima**.
    Kao uzor za neke osnovne osobine jezika bi trebalo uzeti ML i Haskell, kao predstavnike savremenih funkcionalnih programskih jezika.
16. **Podsistem za obradu izuzetaka**.
    Ugradnjom podsistema za reagovanje na uočene nepravilnosti omogućava se pisanje robusnih programa. Podsistem za obradu izuzetaka bi trebalo da bude sintaksno sličan odgovarajućem mehanizmu programskog jezika C++, a semantički prilagođen funkcionalnoj prirodi programskog jezika.

17. **Ograničavanje bočnih efekata**.
    Elemente programskog jezika koji narušavaju principe čisto funkcionalnog programiranja trebalo bi svesti na minimum. Delove programa koji ih upotrebljavaju i izvode neku vrstu bočnog efekta neohodno je istaći.
18. **Implicitno zaključavanje podataka**.
    Podrška konkurentnom izvršavanju bi trebalo da se realizuje kroz zaključavanje deljenih podataka, a vezano za segmente programa koji izvode bočne efekte.
19. **Efikasnost i produktivnost**.
    Potreba za efikasnošću napisanih programa je aktuelna još od prvih dana računarstva. Iako je hardver sve moćniji i jevtiniji, problemi koji se rešavaju pomoću računara su sve složeniji, što za posledicu ima da se efikasnost nikako ne sme zanemariti. Sa druge strane, rastući problem je produktivnost, jer je sve veća potražnja za programerima. Pored već pomenutih karakteristika, poput izražajnosti, čitljivosti i bezbednosti, za produktivnost je veoma značajna mogućnost ponovne upotrebe delova koda, bilo u istom ili drugom projektu.

1.3. Specifične ciljne karakteristike

Na osnovu usvojenog principa prilagođavanja problemu razvoja Veb aplikacija, a u odnosu na ranije opisane karakteristike problema razvoja Veb aplikacija i dinamičkih Veb lokacija, određuju se specifične karakteristike koje bi trebalo da ima programski jezik:

1. **Jednostavnost sintakse**.
   Ciljnu populaciju korisnika u velikoj meri čine mlađi programeri i HTML programeri. Da bi takvi korisnici bilu u stanju da samostalno napišu elementarne programske segmente, programski jezik koji upotrebljavaju mora biti što jednostavniji za upotrebu. Da jednostavnost osnovnih elemenata programskog jezika ne bi dovela do usložnjavanja naprednijih elemenata, neophodno je da osnovni koncepti budu dobro i jasno zasnovani.[2]
2. **Lakoća obučavanja**.
   Potrebno je da osnovni elementi programskog jezika budu prilagođeni brzom i lakom učenju. Pored uobičajene motivacije, dodatni faktor je prethodno pomenut sastav ciljne populacije korisnika programskog jezika.
3. **Prilagođenost generisanju HTML, SGML i XML dokumenata**.
   Pošto HTML predstavlja specijalizaciju jezika SGML i XML, navedeno proširivanje ciljnih tipova dokumenata značajno doprinosi upotrebljivosti programskog jezika, a ne donosi značajno usložnjavanje same definicije jezika.
4. **Jednostavnost postavljanja upita i upotrebe rezultata upita**.
   Činjenica da se generisanje dinamičkih stranica Veb lokacije često oslanja na usluge SUBP posebno ističe značaj interfejsa prema bazama podataka. U kontekstu razvoja Veb aplikacija jedna od najvažnijih karakteristika interfejsa prema bazama podataka

[2] Jednostavnost se može postići smanjivanjem broja mogućnosti programskog jezika, što nije preporučljivo. Na primer, Pascal je jednostavniji ali i manje moćan nego C++ [Ghez1997].

mora biti jednostavnost postavljanja upita i upotrebe rezultata upita. Neophodno je da se upiti mogu postavljati na SQL-u, uz mimalno opterećivanje konstrukcijama samog programskog jezika.

5. **Podsistem za transakcije**.
   Poseban aspekt upotrebe baza podataka su transakcije. Podsistem za transakcije mora omogućiti izvođenje transakcija nad bazom podataka, ali i obezbediti mehanizam transakcija nad internim podacima sesije i servisa.
6. **Upotreba ugneždenog SQL-a**.
   Najznačajnije prednosti ugneždenog SQL-a nad dodavanjem specifičnih konstrukcija ili biblioteka programskom jeziku su:
   - kako je SQL standardni jezik za rad sa relacionim bazama podataka, to su odgovarajući segementi aplikacije prenosivi i mogu se razvijati i testirati i upotrebom drugih alata;
   - programeri koji su obučeni za upotrebu relacionih baza podataka po pravilu poznaju i SQL;
   - upotreba ugneždenog SQL-a prenosi operacije nad bazom podataka na SUBP, čime se dobija na efikasnosti;
   - eventualna upotreba nekog drugog upitnog jezika se može izvesti bez značajnih izmena na nivou programskog jezika;
   - na nivou SQL-a moguće je upotrebljavati specifična napredna svojstva konkretnih sistema za upravljanje bazom podataka.
7. **Interfejs prema drugim programima i komponentama srednjeg sloja**.
   Da bi se programski jezik mogao upotrebljavati za razvoj aplikacija zasnovanih na bilo kojoj arhitekturi ili na već formiranim okruženjima, neophodno je da poseduje mogućnosti komunikacije sa nezavisnim aplikacijama i komponentama srednjeg sloja. Time bi se omogućila i upotreba programskog jezika u distribuiranim arhitekturama, kao što su CORBA i COM.
8. **Nije neophodna interaktivnost**.
   Zato što ne postoji interaktivnost u postupku generisanja pojedinačnih stranica, nije neophodno da interaktivna komunikacija bude podržana osnovnim elementima programskog jezika.
9. **Upravljanje sesijama i dijalozima**.
   Mogućnost upravljanja sesijama i dijalozima doprinela bi upotrebljivosti programskog jezika i značajnom pojednostavljivanju projektovanja i implementiranja složenih interakcija sa posetiocem lokacije.
10. **Modularnost grafičkih aspekata razvoja**.
    Zbog toga što je definisanje grafičkog dizajna Veb lokacije veoma značajan segment njenog razvoja, neophodno je da programski jezik namenjen razvoju Veb lokacija omogući modularan grafički dizajn i pravljenje biblioteka grafičkih elemenata koji se pojavljuju na stranicama.
11. **Paralelizam na nivou podataka**.
    Staranje o problemima koji se pojavljuju u konkurentnim okruženjima nije dobro izvoditi eksplicitno u okviru programa već automatski na nivou podataka. U skladu sa time, podsistem za transakcije mora obuhvatiti i bezbednu upotrebu parametara sesije i servisa.

12. **Mogućnost razdvajanja servisne logike i korisničkog interfejsa**.
    Programski jezik mora podržati mogućnost razdvajanja servisne logike i korisničkog interfejsa, ali ne sme na tome insistirati. Pod time se podrazumeva omogućavanje razdvajanja programskih i grafičkih segmenata definisanja stranica, ali i izdvajanje delova programa koji se odnose na upotrebu baze podataka. Dobro razvojno okruženje bi moralo da obezbedi da se odlučivanje o navedenim razdvajanjima i njihova upotreba mogu prilagoditi potrebama razvojnog tima.
13. **O toku sesije se stara okruženje**.
    O praćenju toka sesije se mora starati okruženje a ne programer. Neophodno je omogućiti da se po potrebi pristupa podacima o sesiji i posetiocu na koga se ona odnosi, ali se identifikovanje sesije prepušta okruženju. Podsistem za transakcije mora da se stara i o podacima koji se odnose na sesije.

2. Programski jezik WAFL

Programski jezik WAFL projektovan je prema postavljenim ciljevima, uz pridržavanje usvojenih osnovnih principa. U narednim odeljcima sledi definicija programskog jezika WAFL. Nakon uvoda u osnovne osobine programskog jezika, uvode se osnovni elementi jezika, koji omogućavaju njegovu relativno opštu primenu, a zatim i specifični elementi koji ga čine posebno prilagođenim za zahtevane primene.

2.1. Osnovne osobine

Programski jezik WAFL je strogo tipiziran vredan funkcionalan objektno orijentisan programski jezik sa implicitnim staranjem o memoriji i (uglavnom) nestriktnom semantikom. Karakteristike programskog jezika WAFL omogućavaju njegovu široku primenu, mada je posebno prilagođen pisanju programa za automatsko generisanje dokumenata.

Većina konstrukcija programskog jezika omogućava pisanje čisto funkcionalnih programa. Ipak, da bi se mogao upotrebljavati i za programiranje transakcija, programski jezik WAFL obuhvata i nekoliko konstrukcija koje ostvaruju bočni efekat bilo menjanjem sadržaja baze podataka, bilo menjanjem određenih objekata u domenu programa. Jasna razdvojenost delova programa koji predstavljaju definiciju transakcija od drugih, čisto funkcionalnih delova programa, omogućava jednostavnije dokazivanje korektnosti čisto funkcionalnih delova programa. Time je pojednostavljeno i dokazivanje korektnosti programa u celosti.

Stroga tipiziranost programskog jezika WAFL se ostvaruje automatskim izvođenjem i razrešavanjem tipova u fazi prevođenja programa, bez eksplicitnog navođenja tipova u izrazima i definicijama.

Objektna orijentacija programskog jezika WAFL ogleda se u podsistemu tipova i načinu formiranja novih apstraktnih tipova podataka. Takođe, i modularnost programa je realizovana primenom principa objektno orijentisanog programiranja. Za razliku od imperativnih objektno orijentisanih programskih jezika, objekti u programskom jeziku WAFL nisu promenljivi, kako bi se ispoštovao princip referencijalne transparentnosti. Jednom formiran objekat je konstantan tokom svog postojanja.

Operacije nad radnom memorijom se izvode implicitno. Svaki put pri formiranju novog objekta automatski se alocira potrebna memorija. Takođe, kada objekat postane suvišan automatski se izvodi oslobađanje memorije koju je zauzimao. Izbor vrste memorijskog podsistema spada u domen implementacije jezika.

Osnovni elementi programskog jezika WAFL omogućavaju njegovu relativno opštu primenu. Obuhvatanjem nekih specifičnih osobina, programski jezik WAFL je posebno prilagođen pisanju programa za automatsko generisanje dokumenata. U fazi projektovanja jezika bilo je u vidu pre svega generisanje HTML dokumenata, ali je istovremeno prilagođen i pisanju programa za automatsko generisanje dokumenata drugih tipova. Tu se pre svega misli na druge tipove dokumenata koji počivaju na tekstualnom zapisu, kao što su dokumenti zapisani na jezicima poput SGML, XML, TeX ili RTF (pomenuti jezici za opis dokumenata definisani su u [ISO:8879], [W3C:2000], [Knuth1984] i [MS:1999]), ali se mogu generisati i tipovi dokumenata koji počivaju na binarnom zapisu, kao što su slike u formatima poput GIF ili JPEG..

WAFL je u osnovi vredan funkcionalan programski jezik. Kako su svi izrazi koji su vredno izračunljivi istovremeno i lenjo izračunljivi, oni izrazi za koje se može dokazati da su vredno izračunljivi mogu se implementirati i primenom lenje semantike a da se time ne menja njihov smisao. Lenja implementacija takvih elemenata programskog jezika može doprineti uštedi neophodnih resursa. To se, pre svega, odnosi na upite.

Programski jezik WAFL se odlikuje uglavnom nestriktnom semantikom, što znači da se redosled izračunavanja uglavnom ne garantuje. Time se pojednostavljuje paralelizovana implementacija programskog jezika. Postojanje određenih elemenata striktne semantike je posledica vredne prirode programskog jezika WAFL i potrebe da se izračunavanje određenih izraza učini efikasnijim.

2.1.1. Osnovni elementi sintakse

Za opisivanje sintakse programskog jezika WAFL upotrebljava se proširena BNF:

- terminali se zapisuju istaknuto;
- neterminali se zapisuju u obliku `<neterminal>`;
- znak `|` razdvaja alternativne elemente;
- male zagrade se upotrebljavaju za grupisanje elemenata;
- zapis `[<element>]` označava opciono (jedno ili nijedno) pojavljivanje elementa;
- zapis `{<element>}` označava nula ili više ponavljanja obuhvaćenog elementa;
- zapis `<neterminal> ::= izraz` označava da se sintaksa neterminala definiše navedenim izrazom.

Formalna sintaksa programskog jezika WAFL priložena je u dodatku

Formalna sintaksa, na strani 75.

U programskom jeziku WAFL komentari se zapisuju na dva načina: u stilu programskog jezika C:

```
/* { <znak> } */
```

i u stilu programskog jezika C++:

```
// { <znak> } <novi_red>
```

Ilustrativni primeri

Pre detaljnog opisa programskog jezika sledi par ilustrativnih primera na programskom jeziku WAFL.

Najjednostavniji program je izraz koji se sastoji od jedne literalne konstante:

```
"Zdravo!"
```

Naredni primer je program koji predstavlja akciju formulara putem koga korisnik upisuje svoje ime. Na osnovu vrednosti elementa formulara sa nazivom `ime` program formira pozdravnu HTML stranicu:

```
pozdrav(FormValue("ime"))
where {
    pozdrav(x) = "<html>"
                 "<head><title>Pozdrav</title></head>"
                 "<body> Zdravo korisniče " + x + "! </body>"
                 "</html>";
    }
```

2.2. Izrazi

Neformalno, program na programskom jeziku WAFL je izraz, a rezultat izračunavanja programa je vrednost izraza kojim je program definisan. Stroga definicija izraza i programa u programskom jeziku WAFL predstavljena je u daljem tekstu.

2.2.1. Prosti tipovi podataka

Prosti tipovi podataka su označeni celobrojni tip `int`, realni tip `float`, logički tip `bool` i tip niski `string`.

Celi brojevi

Konstantna vrednost celobrojnog tipa `int` zapisuje se kao niz cifara kome može neposredno prethoditi znak broja:

```
<ceo_broj> ::= [<znak_broja>]<niz_cifara>
<niz_cifara> ::= <cifra>{<cifra>}
<znak_broja> ::= + | -
```

Implementacija celobrojnog tipa mora biti bar 32-bitna.

Celobrojni operatori su:

Operator	Značenje
+	sabiranje
-	oduzimanje
*	množenje
/	celobrojno deljenje
%	ostatak celobrojnog deljenja
- (unarno)	promena znaka
|	disjunkcija po bitovima
&	konjunkcija po bitovima
~	negacija po bitovima

Tabela 1: Celobrojni operatori

Standardna biblioteka sadrži sledeće celobrojne funkcije:

Funkcija	Vrednost
`abs(x)`	apsolutna vrednost
`random(x)`	slučajan ceo nenegativan broj manji od x

Tabela 2: Celobrojne funkcije

Deljenje nulom i pokušaj izračunavanja ostatka pri deljenju nulom dovode do izuzetka `EIntZeroDivide`. Pokušaj izračunavanja apsolutne vrednosti najmanjeg celog broja može (zavisno od implementacije, tj. ako je odgovarajući pozitivan broj van domena tipa) dovesti do izuzetka `EOutOfRange`.

Realni brojevi

Konstante realnog tipa `float` se zapisuju u obliku:

```
<realan_broj> ::= <ceo_broj>.[<niz_cifara>][<eksponent>]
<eksponent> ::= (e|E)<ceo_broj>
```

Preciznost i opseg brojeva u pokretnom zarezu su u skladu sa standardom IEEE 754 [IEEE:754].

Osnovni operatori nad brojevima realnog tipa su:

Operator	Značenje
+	sabiranje
-	oduzimanje
*	množenje
/	deljenje
- (unarno)	promena znaka

Tabela 3: Operatori nad brojevima realnog tipa

Standardna biblioteka sadrži sledeće funkcije nad brojevima realnog tipa:

Funkcija	Vrednost
`abs(x)`	apsolutna vrednost
`sqrt(x)`	kvadratni koren
`pow(x,y)`	x^y
`exp(x)`	e^x
`ln(x)`	prirodni logaritam
`log(x)`	logaritam za osnovu 10
`sin(x)`	sinus
`cos(x)`	kosinus
`tan(x)`	tangens
`asin(x)`	arkus sinus
`acos(x)`	arkus kosinus
`atan(x)`	arkus tangens
`atan2(x,y)`	arkus tangens od x/y
`sinh(x)`	hiperbolički sinus
`cosh(x)`	hiperbolički kosinus
`tanh(x)`	hiperbolički tangens
`round(x)`	zaokrugljivanje na najbliži ceo broj (rezultat je tipa `int`)
`ceil(x)`	zaokrugljivanje na veći ceo broj (rezultat je tipa `int`)
`floor(x)`	zaokrugljivanje na manji ceo broj (rezultat je tipa `int`)

Tabela 4: Funkcije nad brojevima realnog tipa

Deljenje nulom proizvodi izuzetak `EZeroDivide`. Pokušaj primene neke realne funkcije na vrednost van njenog domena dovodi do izuzetka `EInvalidOperand`. U slučaju da rezultat realne operacije izlazi iz opsega tipa, proizvodi se izuzetak `EOverflow`.

Niske

Konstantna niska tipa `string` se zapisuje kao niz znakova omeđen navodnicima ili apostrofima. Niska se može nastavljati u više redova tako što se na kraju jednog reda privremeno zatvore graničnici, da bi se ponovo otvorili u narednom redu. Ako je niska ograničena navodnicima, u njoj se smeju pojaviti apostrofi i obratno:

```
<niska> ::= <deo_niske>{{<prazan_prostor>}<deo_niske>}
<deo_niske> ::= ("{<znak_osim_">}") | ('{<znak_osim_'>}')
```

Znaci navodnika i apostrofa i ostali specijalni znaci se mogu zapisivati u okviru niske na sličan način kao u programskom jeziku C:

Simbol	Oznaka
navodnik: "	\"

Simbol	Oznaka
apostrof: '	`\'`
obrnuta kosa crta: \	`\\`
novi red	`\n`
povratak na početak reda	`\r`
horizontalni tabulator	`\t`
vertikalni tabulator	`\v`
nova strana	`\f`
jedan znak unazad	`\b`

Tabela 5: Specijalni znaci

Implementacija tipa `string` mora dopuštati da sadržaj niske čine bilo koji znaci koji se mogu kodirati sa 8 bita, uključujući i one koji nemaju vizualnu reprezentaciju[3]. Na taj način se obezbeđuje da se niske mogu upotrebljavati za rad sa bilo kojim nizovima bajtova, a ne samo sa njihovim podskupom koji nazivamo tekstualnim niskama.

Nad niskama je definisan operator sabiranja:

Operator	Značenje
+	dopisivanje niski

Tabela 6: Operatori nad niskama

Standardna biblioteka sadrži konstantu `NullString` i sledeće funkcije nad niskama:

Funkcija	Vrednost
`isNull(x)`	provera da li je niska `x` jednaka konstanti `NullString`
`ifNull(x,y)`	ako važi `isNull(x)` rezultat je `y`, inače je rezultat `x`
`strLen(x)`	izračunava dužinu niske `x`
`strCat(x,y)`	izračunava nisku koja se dobija dopisivanjem niske `y` iza niske `x` (isto kao "sabiranje" niski: `x+y`)
`subStr(x,p,n)`	izdvaja podnisku niske `x`, dužine `n`, počev od znaka sa indeksom `p` (prvi znak je sa indeksom `0`)
`strLeft(x,n)`	izdvaja podnisku niske `x`, koja se sastoji od prvih `n` znakova (kao `subStr(x,0,n)`)
`strRight(x,n)`	izdvaja podnisku niske `x`, koja se sastoji od poslednjih `n` znakova (kao `subStr(x,length(x)-n,n)`)
`strLTrim(x)`	izračunava nisku koja se dobija kada se sa početka niske `x` uklone sve praznine (blanko, novi red, tabulacija i sl.)
`strRTrim(x)`	izračunava nisku koja se dobija kada se sa kraja niske `x` uklone sve praznine (blanko, novi red, tabulacija i sl.)

[3] U slučaju ASCII to su znaci sa kodovima od 0 do 31. U slučaju nekog drugog rasporeda mogu biti neki drugi znaci.

Funkcija	Vrednost
`strTrim(x)`	izračunava nisku koja se dobija kada se sa početka i kraja niske `x` uklone sve praznine (blanko, novi red, tabulacija i sl.)
`strPos(x,y)`	izračunava indeks prvog znaka prve pojave podniske `y` u okviru niske `x`, ili `-1` ako niska `x` ne sadrži traženu podnisku
`strNextPos(x,y,n)`	izračunava indeks prvog znaka prve pojave podniske `y` u okviru niske `x`, ili `-1` ako niska `x` ne sadrži traženu podnisku, pri čemu se ne razmatraju prvih `n` znakova niske `x`
`strLastPos(x,y)`	izračunava indeks prvog znaka poslednje pojave podniske `y` u okviru niske `x`, ili `-1` ako niska `x` ne sadrži traženu podnisku
`strNextLastPos(x,y,n)`	izračunava indeks prvog znaka poslednje pojave podniske `y` u okviru niske `x`, ili `-1` ako niska `x` ne sadrži traženu podnisku, pri čemu se razmatra samo prvih `n` znakova niske `x`
`strPosI(x,y)`	kao `strPos`, ali bez pravljenja razlike između malih i velikih slova
`strNextPosI(x,y,n)`	kao `strNextPos`, ali bez pravljenja razlike između malih i velikih slova
`strLastPosI(x,y)`	kao `strLastPos`, ali bez pravljenja razlike između malih i velikih slova
`strNextLastPosI(x,y,n)`	kao `strNextLastPos`, ali bez pravljenja razlike između malih i velikih slova
`strLowerCase(x)`	izračunava nisku koja predstavlja kopiju niske `x` u kojoj su sva velika slova zamenjena malim slovima
`strUpperCase(x)`	izračunava nisku koja predstavlja kopiju niske `x` u kojoj su sva mala slova zamenjena velikim slovima
`strReplace(x,y,z,n)`	izračunava nisku koja predstavlja kopiju niske `x` u kojoj je `n`-ta pojava podniske `y` zamenjena podniskom `z`
`strReplaceI(x,y,z,n)`	izračunava nisku koja predstavlja kopiju niske `x` u kojoj je `n`-ta pojava podniske `y` zamenjena podniskom `z`, zanemarujući pri poređenju veličinu slova
`strReplaceAll(x,y,z)`	izračunava nisku koja predstavlja kopiju niske `x` u kojoj su sve pojave podniske `y` zamenjene podniskom `z`
`strReplaceAllI(x,y,z)`	izračunava nisku koja predstavlja kopiju niske `x` u kojoj su sve pojave podniske `y` zamenjene podniskom `z`, zanemarujući pri poređenju veličinu slova
`strSplit(x,d)`	iz niske `x` izdvaja listu podniski međusobno razdvojenih graničnikom `d`
`strJoin(l,d)`	izračunava nisku koja se dobija spajanjem niski koje predstavljaju elemente liste `l` i umetanjem graničnika `d` između njih
`strReverse(x)`	izračunava nisku sa obrnutim redosledom znakova u

Funkcija	Vrednost
	odnosu na `x`

Tabela 7: Funkcije nad niskama

Pokušaj pristupanja nepostojećem delu niske, tj. upotrebe neispravnih indeksa kao argumenata navedenih funkcija, proizvodi izuzetak `EInvalidIndex`.

Logički tip

Konstante logičkog tipa `bool` su logičke vrednosti `true` (istinitosna vrednost *tačno*) i `false` (istinitosna vrednost *netačno*):

```
<logička_vrednost> ::= true | false
```

Nad vrednostima logičkog tipa definisani su sledeći operatori:

Operator	Značenje
`and`	konjunkcija
`&&`	konjunkcija
`or`	disjunkcija
`||`	disjunkcija
`not`	negacija
`!`	negacija

Tabela 8: Operatori logičkog tipa

Zbog saglasnosti zapisa kako sa programskim jezicima C i C++, tako i sa upitnim jezikom SQL, za svaku operaciju postoje po dva ekvivalentna logička operatora.

Nad prostim tipovima podataka, osim logičkog, definisane su uobičajene operacije poređenja, koje za rezultat imaju vrednost logičkog tipa. Zbog saglasnosti zapisa kako sa programskim jezicima C i C++, tako i sa SQL-om, postoje po dva ekvivalentna operatora za proveru jednakosti i različitosti. U slučaju niski, poređenje se izvodi leksikografski.

Operator	Značenje
`==`	jednako
`=`	jednako
`!=`	različito
`<>`	različito
`>`	veće od
`<`	manje od
`>=`	veće od ili jednako
`<=`	manje od ili jednako

Tabela 9: Operatori poređenja

Konverzija tipova

Nije dopuštena implicitna (automatska) konverzija vrednosti jednog prostog tipa u vrednost drugog prostog tipa. Umesto toga, definisane su funkcije za eksplicitnu konverziju.

Funkcija	Vrednost
`asInt(x)`	Konvertuje vrednost `x`, necelobrojnog tipa, u celobrojni tip: • Vrednost realnog tipa se konvertuje zaokrugljivanjem na najbliži ceo broj (identično funkciji `round(x)`); • Vrednost `true` logičkog tipa konvertuje se u ceo broj `1`, dok se vrednost `false` konvertuje u `0`; • Niska se kovertuje u ceo broj ako predstavlja ispravan dekadni zapis broja, inače se konvertuje u ceo broj `0`.
`asFloat(x)`	Konvertuje vrednost `x`, koja nije realnog tipa, u realni tip: • Vrednost celobrojnog tipa se konvertuje u odgovarajuću vrednost realnog tipa; • Vrednost `true` logičkog tipa konvertuje se u realan broj `1.0`, dok se vrednost `false` konvertuje u `0.0`; • Niska se kovertuje u realan broj ako predstavlja ispravan dekadni zapis realnog broja, inače se konvertuje u realan broj `0.0`.
`asString(x)`	Konvertuje vrednost `x`, koja nije tipa `string`, u nisku: • Vrednost celobrojnog tipa se konvertuje u nisku koja predstavlja njen zapis u dekadnom brojnom sistemu; • Vrednost realnog tipa se kovertuje u nisku koja predstavlja njen decimalni zapis u dekadnom brojnom sistemu; • Vrednost `true` logičkog tipa konvertuje se u nisku `"true"`, dok se vrednost `false` konvertuje u nisku `"false"`.
`asChar(x)`	Konvertuje vrednost x, koja nije tipa niske, u znak (nisku dužine jedan): • Vrednost celobrojnog tipa se konvertuje u znak sa kodom `x`; • Vrednost realnog tipa se kovertuje u celobrojnu vrednost i dalje u nisku kao da je u pitanju ceo broj; • Vrednost `true` logičkog tipa konvertuje se u nisku `"T"`, dok se vrednost `false` konvertuje u nisku `"F"`.
`asBool(x)`	Konvertuje vrednost `x`, koja nije logičkog tipa, u logičku vrednost: • Vrednost celobrojnog tipa se konvertuje u `true`, ako je različita od `0` ili u `false` ako je jednaka `0`; • Vrednost realnog tipa se konvertuje u `true`, ako je različita od `0.0` ili u `false` ako je jednaka `0.0`; • Niske `"true"` i `"T"` se konvertuju u `true`, a sve ostale niske u `false`.

Tabela 10: Funkcije za konverziju prostih tipova

U nekim od predstavljenih slučajeva konverzije može se dogoditi da se konverzija ne može izvesti. U tim slučajevima se proglašava izuzetak `EOutOfRange`:

- vrednost realnog tipa se ne može prevesti u ceo broj ako je van opsega celih brojeva;

- vrednost celobrojnog tipa se ne može prevesti u znak ako je negativna ili veća od 255.

2.2.2. Izrazi

U programskom jeziku WAFL izrazi se dele na *proste izraze*, *operatorske izraze* i *složene izraze*. Za proste izraze i operatorske izraze upotrebljava se naziv *izraz*, dok se svi izrazi, uključujući i složene, nazivaju *opšti izrazi*.

Prosti izrazi

Prosti izrazi u programskom jeziku WAFL su:

- literalna konstanta bilo kog osnovnog ili složenog tipa;
- zapis konstantne liste, zapis konstantnog niza i zapis konstantnog kataloga[4];
- imenovana vrednost, tj. vrednost na koju se referiše putem nekog imena, a najčešće se odnosi na argumente funkcije ili na definisane (tj. imenovane) vrednosti ili funkcije[5];
- izraz u zagradi, oblika: `(<izraz>)`;
- uslovni izrazi `if` i `switch`;
- aplikativni izraz.

Izraz if

Uslovni izraz `if` obezbeđuje uslovno izračunavanje tačno jedne od ponuđenih dveju grana, u zavisnosti od istinitosne vrednosti datog uslova. Izraz `if` ima oblik:

```
if <izraz1> then <izraz2> else <izraz3>
```

Prvi izraz mora biti logičkog tipa. Alternativni izrazi moraju biti istog tipa.

Izračunavanje izraza `if` započinje izračunavanjem prvog izraza. Ako je njegova vrednost `true`, tada se izračunava alternativni izraz iza ključne reči `then` i vrednost čitavog izraza `if` je njegova vrednost; inače se izračunava alternativni izraz iza ključne reči `else` i vrednost čitavog izraza `if` je njegova vrednost. U oba slučaja se preostali alternativni izraz ne računa.

Sledi primer izraza `if` koji izračunava veći od brojeva `x` i `y`:

```
if x>y then x else y
```

Izraz switch[6]

Uslovni izraz `switch` obezbeđuje uslovno izračunavanje tačno jedne od više ponuđenih grana, u zavisnosti od vrednosti datog izraza koji predstavlja kriterijum. Izraz `switch` ima oblik:

---

[4] Načini zapisivanja konstantne liste, konstantnogg niza i konstantnog kataloga predstavljeni su u odeljcima *2.4.3 Lista*, *2.4.4 Niz* i *2.4.5 Katalog*, sa početkom na strani 36.

[5] U nefunkcionalnim programskim jezicima se obično upotrebljava izraz *promenljiva*, ali kako imenovana vrednost u funkcionalnom programskom jeziku ne može biti izmenjena, izraz *promenljiva* ne odgovara. U odeljku *2.3.7 Biblioteke* se kao opštiji oblik imena uvodi *kvalifikovano ime*.

[6] Izraz switch još nije implementiran.

```
switch <izraz> {
    <klauzula_case>{<klauzula_case>}
    <klauzula_default>
    }
```

gde je `<klauzula_case>` definisana kao:

```
case <literalna_konstanta>:{case <literalna_konstanta>:}[<izraz>;]
```

a `<klauzula_default>` kao:

```
default <izraz>
```

Izraz koji predstavlja kriterijum grananja mora biti prostog tipa. Sve literalne konstante koje se navode kao oznake klauzula `case` moraju biti istog tog tipa. Svi alternativni izrazi koji se pojavljuju u klauzulama `case` i u klauzuli `default` moraju biti istog tipa.

Izračunavanje izraza `switch` započinje izračunavanjem kriterijuma grananja. Ako postoji klauzula `case` uz koju je navedena literalna konstanta čija je vrednost jednaka vrednosti dobijenoj izračunavanjem kriterijuma grananja, tada se izračunava alternativni izraz naveden u toj klauzuli case i vrednost čitavog izraza `switch` je njegova vrednost; inače se izračunava alternativni izraz iza ključne reči `default` i vrednost čitavog izraza `switch` je njegova vrednost. U oba slučaja preostali izrazi se ne računaju.

Sledi primer izraza `switch` koji izračunava broja dana u mesecu `mesec`:

```
switch mesec {
    case 4 : case 6 : case 9 : case 11 :
        30;
    case 2 :
        if godina%4==0 and (godina%100!=0 or godina%400==0)
           then 29
           else 28;
    default :
        31;
    }
```

Aplikativni izraz

Aplikativni izraz predstavlja izraz u kome postoji aplikacija funkcije ili metoda. Zbog srodne sintakse u aplikativne izraze se ubraja i izraz sa indeksiranjem. Sintaksa aplikativnog izraza je:

```
<izraz_sa_aplikacijom> ::= <nosilac_aplikacije>
    ( <aplikacija> | <indeksiranje> | <poziv_metoda> )
<nosilac_aplikacije> ::=
    <izraz_u_zagradi>
    | <ime> [::<ime>]
    | <izraz_sa_aplikacijom>
<aplikacija> ::= ( [ <izraz> {, <izraz> } ] )
<indeksiranje> ::= [<izraz>]
<poziv_metoda> ::= .<ime><aplikacija>
```

Aplikaciju funkcije ili metoda nazivamo i *funkcijski poziv*. Funkcijskim pozivima se bavi naredni odeljak, dok je izraz sa indeksiranjem opisan u odeljku *2.4.4 Niz* na strani 38.

Funkcijski poziv

U slučaju svih korisničkih funkcija funkcijski poziv se izračunava vredno. Najpre se izračunavaju izrazi koji predstavljaju argumente funkcijskog poziva, a zatim se izračunava i vrednost funkcije za izračunate vrednosti argumenata. Uvek se izračunavaju svi argumenti funkcije, pri čemu se ne garantuje redosled njihovog izračunavanja.

Samo u slučaju nekih primitivnih bibliotečkih funkcija (koje će biti posebno opisane) argumenti se ne izračunavaju pre nego što postanu neophodni, a u nekim slučajevima mogu i ostati neizračunati. Takođe, samo neke od primitivnih bibliotečkih funkcija garantuju redosled izračunavanja argumenata.

Svaka funkcija koja ima bar jedan argument može se zapisati upotrebom objektno orijentisane sintakse tako što se prvi argument izdvoji iz liste argumenata i navede kao nosilac poziva. Obrnuto ne važi, jer se metodi klasa ne mogu upotrebljavati kao funkcije[7]. Naredna dva funkcijska poziva su ekvivalentna:

```
subStr(s,5,15)
s.subStr(5,15)
```

Osnovni razlog za uvođenje alternativnog načina zapisivanja izraza je radi povećavanja čitljivosti zapisa u slučajevima kada neka vrednost prolazi kroz veći broj transformacija da bi se dobio rezultat. U programima na funkcionalnim programskim jezicima takve situacije se pojavljuju relativno često. Programer može birati način zapisivanja izraza prema kontekstu. Radi ilustracije ćemo jedan izraz zapisati najpre na uobičajen način:

```
dsgLib::makeFrame(
    opisArtikla(
        asInt( ifNull( FormValue("art_id"), "0")),
        12,
        "plava"
        ),
    "Knjižara - Opis knjige"
    )
```

a zatim primenom objektno orijentisane sintakse:

```
FormValue("art_id")
    .ifNull("0")
    .asInt()
    .opisArtikla(12,"plava")
.dsgLib::makeFrame("Knjižara - Opis knjige")
```

Operatorski izrazi

Izrazi u kojima se bar jedan operator pojavljuje van svih zagrada nazivaju se *operatorski izrazi*. Operatorski izrazi mogu sadržati unarne i binarne operatore. Njihova sintaksa je:

```
<operatorski_izraz> ::=
    { <unarni_operator> } <prost_izraz>
    [ <binarni_operator> <operatorski_izraz> ]
```

[7] O klasama i metodima klasa biće reči u nastavku teksta, u odeljku *2.5 Klase*, na strani 44.

Složeni izrazi[8]

Složeni izrazi predstavljaju izraze sa pomoćnim definicijama i izraze sa hvatačima izuzetaka. Naredna tabela sadrži njihovu sintaksu, dok detaljna objašnjenja slede u nastavku ovog poglavlja. Formalno, opšti oblik složenog izraza se naziva *izraz except*:

Oblik izraza	Značenje

<izraz_except> ::=
  <izraz_where> [<except_podizraz>]
<except_podizraz> ::=
  except { <hvatač>{<hvatač>} **}**
<hvatač> ::=
  catch([[<ime>] <ime>]**)=**<izraz_except>**;**
	izraz sa obradom izuzetaka

<izraz_where> ::= <izraz> [<where_podizraz>] <where_podizraz> ::= **where {** <definicija> {<definicija>} **}**	izraz sa lokalnim definicijama

Tabela 11: Složeni izrazi

2.2.3. Način izračunavanja izraza

Način izračunavanja izraza u programskom jeziku WAFL definisan je:

- načinom prenosa argumenata funkcija i operatora;
- redosledom izračunavanja izraza;
- prioritetom operatora;
- lenjošću semantike izraza.

Prenos argumenata

U programskom jeziku WAFL svi argumenti funkcija i operatora se prenose po vrednosti[9]. To je neophodno da bi se održao princip referencijalne transparentnosti.

Redosled izračunavanja izraza

Kako se programski jezik WAFL odlikuje uglavnom nestriktnom semantikom, u većini slučajeva se ne garantuje redosled izračunavanja. Striktna semantika (garantovan redosled izračunavanja) se primenjuje samo u narednim situacijama:

- u operatorskim izrazima se poštuju prioriteti operatora, što utiče na redosled izračunavanja;
- svi argumenti korisničkih funkcija se izračunavaju pre same korisničke funkcije, pri čemu se redosled izračunavanja argumenata ne garantuje;

[8] Izraz except još nije implementiran. svuda gde se pominje za sada je omogućena primena izraza where.

[9] Primetimo da je to semantička pretpostavka. Način implementacije prenosa argumenata nije značajan, odnosno može biti i po imenu (ili adresi), sve dok se garantuje da argumenti ne mogu biti promenjeni pri izračunavanju funkcije.

- operandi operatora logičke konjunkcije, logičke disjunkcije i dopisivanja niski se izračunavaju sleva udesno;
- garantuje se redosled izračunavanja izraza koji čine transakcije.

Za operatore logičke konjunkcije i disjunkcije striktna semantika se uvodi da bi se omogućilo njihovo lenjo izračunavanje. Što se tiče dopisivanja niski, striktna semantika je uvedena da bi se u slučaju izvođenja transakcija pri izračunavanju desnog operanda onemogućio uticaj tih transakcija na vrednost levog operanda. U slučaju transakcija radi se o specijalnom slučaju pravila za operator konjunkcije, jer se telo transakcije tretira kao logička konjunkcija više izraza čiji je rezultat logička vrednost.

Prioritet operatora

Naredna tabela predstavlja prioritet i asocijativnost operatora. Opis navedenih operatora koji do sada nisu opisani (operatori za referenciranje, pozivanje i indeksiranje) sledi u nastavku poglavlja:

Operator	Značenje	Prioritet	Asocijativnost
`::`	referenciranje elementa biblioteke	1 (najviši)	Leva
`.`	referenciranje metoda klase ili pozivanje funkcije poput metoda	1	Leva
`[]`	indeksiranje	1	Leva
`not, !, -, ~`	unarne operacije	2	Desna
`*, /, %, &`	multiplikativne operacije	3	Leva
`:`	formiranje liste	4	Desna
`+, -, |`	aditivne operacije	5	Leva
`<, >, <=, >=, <>, !=, ==, =`	poređenja	6	Leva
`and, &&, &`	konjunkcija	7	Leva
`or, ||, |`	disjunkcija	8	Leva

Tabela 12: Prioritet i asocijativnost operatora

Može se primetiti namerna sličnost sa prioritetima operatora u programskom jeziku C++ [Stro1997].

Lenjost

Programski jezik WAFL je u osnovi vredan programski jezik, što znači da se argumenti funkcija i operatora izračunavaju pre nego što se započne izračunavanje samih funkcija. Ipak, postoji određena doza lenjosti koja je uvedena zbog povećanja efikasnosti. Opšte pravilo je da se izrazi ne izračunavaju ako njihova vrednost pouzdano ne utiče na rezultat.

Sledi pregled konkretnih slučajeva u kojima se primenjuje lenja semantika:

- pri izračunavanju uslovnih izraza `if` i `switch` izračunavaju se samo uslov i odgovarajući alternativni izraz, dok se ostali alternativni izrazi ne izračunavaju;

- pri izračunavanju argumenata operatora logičke konjunkcije i disjunkcije: `and` i `or`, desni operand se ne izračunava ako se rezultat može ustanoviti samo na osnovu levog operanda;
- implementacije bibliotečkih funkcija `forall` i `exists` su u skladu sa prethodnim slučajem, tj. kao da su implementirane uz upotrebu operatora `and` i `or` na sledeći način:

```
forall(l,f) =
    if empty(l) then true
    else f(hd(l)) and forall(tl(l),f);

exists(l,f) =
    if empty(l) then false
    else f(hd(l)) or exists(tl(l),f);
```

- u nekim situacijama se mogu formirati *lenje liste* ili *lenji nizovi*, čiji se elementi ne izračunavaju dok njihove vrednosti ne budu potrebne. U nastavku rada su opisane neke konkretne situacije. Primer su liste koje predstavljaju rezultate upita nad bazom podataka;
- u nekim situacijama se mogu formirati *lenje niske*, čiji se sadržaj izračunava parcijalno i po potrebi. Konkretne situacije u kojima je takav pristup su opisane u nastavku rada. Primer su niske koje predstavljaju rezultat čitanja sadržaja datoteke.

U konkretnim implementacijama se može uvoditi lenjo izračunavanje i na druga mesta, ali uz obavezu da se time ni na koji način ne menja semantika programa. Zbog toga se lenjost sme uvoditi samo na mestima gde ne postoji (ili je na neki način neutralisan) uticaj spoljnih faktora

2.3. Struktura programa

Program na programskom jeziku WAFL se definiše jednim izrazom, koji može biti prost izraz, izraz where ili izraz except. Rezultat izračunavanja programa je rezultat izračunavanja izraza kojim je on definisan.

Da bi tekst programa bio jasniji, jednostavniji za zapisivanje, manje podložan greškama i lakši za održavanje, potrebno je da bude na određeni način struktuiran. U imperativnim programskim jezicima to se postiže uvođenjem određenih kontrolnih naredbi, kao što su naredbe ponavljanja ili uslovnog izvršavanja, i definisanjem potprograma. U slučaju funkcionalnih programskih jezika kontrolne naredbe se zamenjuju uslovnim izrazima i rekurzijom, a kao vid potprograma se definišu funkcije.

WAFL je funkcionalan programski jezik, pa se struktuiranost programa u njemu postiže na sličan način kao u drugim funkcionalnim programskim jezicima: upotrebom rekurzije, uslovnih izraza `if` i `switch` i definisanjem funkcija. Zbog prilagođenosti WAFL-a određenim specifičnim primenama u oblasti Veb aplikacija i baza podataka, moguće je definisati i neke specifične oblike funkcija.

U narednim odeljcima opisan je način definisanja funkcija u programskom jeziku WAFL, dok su neke specifične primene predstavljene kasnije.

2.3.1. Izraz where

Da bi se zapis programa učinio preglednijim i jasnijim uvode se korisničke definicije. Izraz *where* predstavlja osnovnu formu za uvođenje korisničkih definicija. Prisetimo se njegove sintakse:

```
<izraz> where {<definicija> {<definicija>}}
```

gde definicije imaju sledeći opšti oblik:

```
<definicija> ::= <ime> [<formalni_parametri>] = <telo_definicije>;
<formalni_parametri> ::= (<ime> {,<ime>} )
```

2.3.2. Prostori imena

Prostor imena je deo izraza u kome se definišu i mogu upotrebljavati definisana imena. Postoje tri osnovna tipa prostora imena:

- svakoj definiciji odgovara jedan prostor imena, tzv. *prostor imena definicije*, kome pripadaju formalni parametri definicije;
- svakom izrazu *where* odgovara jedan prostor imena, tzv. *prostor imena izraza where*, kome pripadaju sva imena definisana neposredno u okviru tog izraza *where*;
- *globalni prostor imena* čine globalno definisana imena .

Početak i kraj prostora imena definišu se u odnosu na tekst programa. Prostori imena se ne mogu preklapati, ali jedan prostor imena može obuhvatiti druge manje prostore imena. Prostor koji obuhvata nazivamo *obuhvatajući prostor imena*. Prostor koji je obuhvaćen nazivamo *obuhvaćen prostor imena*. Definicijom se ime *uvodi* u tačno jedan prostor imena, onaj koji neposredno obuhvata definiciju, pa se tako ime uvedeno u neki prostor imena ne uvodi istovremeno i u prostor imena koji ga obuhvata.

Ime se može upotrebljavati u nekom prostoru imena ako i samo ako je u njemu *vidljivo*. Vidljivost imena definiše se na sledeći način:

1. imena iz globalnog prostora imena vide se u svim prostorima imena koji su njime obuhvaćeni;
2. nijedno ime uvedeno u nekom obuhvaćenom prostoru imena ne vidi se u prostoru imena koji ga obuhvata, pa ni van njega;
3. ako je neko ime uvedeno u obuhvaćeni prostor imena, a isto ime (uvedeno nekom drugom definicijom) je vidljivo u obuhvatajućem prostoru imena, tada se u obuhvaćenom prostoru imena vidi samo ime koje je u njemu definisano;
4. imena iz prostora imena definicije, ne vide se u obuhvaćenim prostorima imena (tj. formalni parametri definicije vide se samo u telu definicije, a ako je telo izraz *where*, onda samo u osnovnom izrazu tog izraza *where*, a ne i unutar definicija uvedenih u tom izrazu *where*);
5. imena iz prostora imena izraza *where* vide se u svim obuhvaćenim prostorima imena (pa i u čitavom izrazu *where*, uključujući i njegov osnovni izraz i sve definicije uvedene u tom izrazu *where*).

Vidljivost imena u odnosu na druge složene izraze opisana je u odeljcima koji su posvećeni tim složenim izrazima.

Raspoznavanje imena u u okviru nekog izraza u fazi prevođenja programa odvija se na sledeći način:

1. najpre se pokušava sa pronalaženjem imena u prostoru imena u kome se nalazi izraz;
2. ako ime nije pronađeno u nekom prostoru imena, pokušava se sa pronalaženjem imena u prostoru imena kojim je ovaj neposredno obuhvaćen;
3. korak 2. se primenjuju rekurzivno;
4. ukoliko primena prethodnih koraka dovede do globalnog prostora imena, a da ni u njemu ne postoji traženo ime, tada ime nije pronađeno i u pitanju je greška u programu.

2.3.3. Definisanje funkcija

U funkcionalnim programskim jezicima u programima se najčešće definišu funkcije. Definicija predstavlja definiciju funkcije ako je telo definicije neki prost izraz, izraz where ili izraz except.

```
<definicija> ::= <ime> [<formalni_parametri>] = <telo_definicije>;
<formalni_parametri> ::= (<ime> {,<ime>} )
<telo_definicije> ::= <except_izraz>
```

Primer definicije funkcije

```
hipotenuza(x,y) = sqrt( x*x + y*y );
```

Slično, ali uz pomoćnu definiciju funkcije `kvadrat`:

```
hipotenuza(x,y) = sqrt( kvadrat(x) + kvadrat(y) )
where {
    kvadrat(x) = x*x;
    };
```

2.3.4. Rekurzija

Kao što je već navedeno, u okviru tela funkcije vidljiva su, između ostalog, sva imena definisana u prostoru imena u kome se nalazi definicija funkcije. Direktna posledica je da se rekurzivne funkcije mogu definisati bez posebnog označavanja, jer je ime funkcije vidljivo unutar tela njene definicije.

Kao primer definicije rekurzivne funkcije može poslužiti jednostavna (mada neefikasna) funkcija koja izračunava elemente Fibonačijevog niza:

```
fib(n) = if n<3 then 1 else fib(n-1)+fib(n-2);
```

Naredni primer definiše funkciju koja proverava da li broj `n` ima neparnih delilaca u opsegu [`poč`, `kraj`], uz pretpostavku da je `poč` neparan broj:

```
imaDelilaca(n,poč,kraj) =
    if poč>kraj then false
    else if (n%poč)==0 then true
    else imaDelilaca(n,poč+2,kraj);
```

U slučaju uzajamne rekurzije postoji ograničenje da sve funkcije, čije međusobno pozivanje predstavlja uzajamnu rekurziju, moraju biti definisane u okviru istog prostora imena. Ovo ograničenje se može prevazići tako što se neka od funkcija uključenih u rekurziju (ili sve) prenese kao parametar.

2.3.5. Polimorfizam

Jedno od osnovnih svojstava programskog jezika WAFL je mogućnost definisanja polimorfnih funkcija. Polimorfne funkcije mogu da se izračunavaju nad različitim tipovima argumenata, čime se postiže značajno povećanje efikasnosti programera, jer ne moraju ponavljati pisanje ekvivalentne funkcije za svaki od tipova nad kojim ju je moguće definisati.

Raspoznavanje tipova u fazi prevođenja programa se izvodi automatski, bez eksplicitnog navođenja imena tipova u programu. Zbog toga svaka implementacije programskog jezika WAFL mora obuhvatiti mehanizam za *automatsko razrešavanje tipova*. Automatsko razrešavanje tipova počiva na analizi definicija funkcija pri čemu se pokušava raspoznavanje tipa funkcije uz očuvanje njegove što veće opštosti.

Da bi funkcija bila polimorfna dovoljno je da je definisana i da izraz koji predstavlja njeno telo može da se ispravno protumači za više različitih tipova argumenata. Nije potrebno nikakvo posebno označavanje. Zapravo, dok se ne utvrde tačno tipovi funkcija, sve funkcije se smatraju za polimorfne.

U narednom primeru se definiše polimorfna fukcija `saberi`, koja može da se izračunava nad celim brojevima, realnim brojevima i niskama, a zatim se upotrebljava u tri različita konteksta kao celobrojna, realna i funkcija nad niskama:

```
asString(saberi(1,2))           // celobrojna funkcija saberi
+ asString(saberi(3.1,4.2))     // realna funkcija saberi
+ saberi("pet","šest"))         // funkcija saberi nad niskama
where {
    saberi(x,y) = x+y;
    }
```

Rezultat izračunavanja ovog izraza je:

```
37.3petšest
```

Polimorfizam u programskom jeziku WAFL ne ide na uštrb provere tipova u fazi prevođenja. Za razliku od programskih jezika koji u fazi prevođenja nemaju (ili skoro da nemaju) proveru tipova, kao na primer LISP, programski jezik WAFL je strogo tipiziran jezik pa se u fazi prevođenja mora izvoditi stroga provera tipova. Svaka neusaglašenost tipova mora biti prepoznata već u fazi prevođenja.

2.3.6. Funkcije višeg reda

U programskom jeziku WAFL funkcije su *građani prvog reda*. To znači da funkcije mogu da se pojave kao argumenti i rezultati izračunavanja, kako pri radu sa nekim funkcijama standardne biblioteke, tako i u radu sa korisnički definisanim funkcijama. Funkcije čiji su argumenti ili rezultat nekog funkcijskog tipa nazivaju se *funkcije višeg reda*.

Kao i u slučaju polimorfizma, nema potrebe da se posebno naglašava ni da li je funkcija koja se definiše funkcija višeg reda ili ne. To se ustanovljava automatski tokom raspoznavanja tipa funkcije.

U narednom primeru se, zavisno od vrednosti niske, izvodi sabiranje ili oduzimanje dva broja (primetimo da su funkcije `sabiranje`, `oduzimanje` i `računanje` polimorfne i da se jednako dobro mogu koristiti i nad realnim brojevima):

```
f(5,6,"+")
where {
    f(x,y,operator) =
         if operator == "+" then računanje(x,y,sabiranje)
         else if operator == "-" then računanje(x,y,oduzimanje)
         else x;
    sabiranje(x,y) = x+y;
    oduzimanje(x,y) = x-y;
    računanje(x,y,f) = f(x,y);
    }
```

Upotreba funkcija višeg reda podrazumeva da se funkcije mogu pojaviti ne samo kao argumenti, već i kao rezultati izraza. Tako se prethodni primer može napisati i na sledeći način:

```
f(5,6,"+")
where {
    f(x,y,operator) = računanje(x,y,
                          if operator == "+" then sabiranje
                          else if operator == "-" then oduzimanje
                          else prviArgument
                          );
    sabiranje(x,y) = x+y;
    oduzimanje(x,y) = x-y;
    prviArgument(x,y) = x;
    računanje(x,y,f) = f(x,y);
    }
```

Rezultat izračunavanja izraza u prethodnim primerima je `11`.

Funkcija `map` iz standardne biblioteke, koja primenjuje datu funkciju redom na sve elemente date liste i kao rezultat daje listu rezultata, može se definisati kao u narednom primeru:

```
map([1,2,3,4,5],kvadrat)
where {
    map(l,f) =  if empty(l)
                   then nil
                   else f(hd(l)):map(tl(l),f);
    kvadrat(x) = x*x;
    };
```

Rezultat izračunavanja izraza iz prethodnog primera je lista `[1, 4, 9, 16, 25]`.

Funkcije višeg reda u standardnoj biblioteci

Standardna biblioteka obuhvata nekoliko osnovnih funkcija višeg reda koje se najčešće upotrebljavaju. U narednoj tabeli su predstavljene najvažnije funkcije višeg reda iz standardne biblioteke, koje vrše izračunavanje nad listama:

Funkcija	Opis i tip
`l.map(f)`	Rezultat je lista rezultata funkcije `f` za vrednosti iz liste `l`. Redosled rezultata odgovara redosledu elemenata u listi `l`. Tip je: `List['a] * ('a -> 'b) -> List['b]`
`l.filter(f)`	Rezultat je lista vrednosti iz `l` koje zadovoljavaju uslov `f`. Redosled elemenata je očuvan. Tip je: `List['a] * ('a -> bool) -> List['a]`
`l.aggregate(f,n)`	Desno asocijativno agregiranje ("sumiranje") elemenata liste `l` funkcijom `f`, pri čemu se počinje od vrednosti `n`. Tip je: `List['a] * ('a*'b -> 'b) * 'b -> 'b`
`l.leftAggregate(f,n)`	Levo asocijativno agregiranje elemenata liste `l` funkcijom `f`. Početna vrednost agregacije je `n`. Tip je: `List['a] * ('b*'a -> 'b) * 'b -> 'b`
`l.forall(f)`	Konjunkcija elemenata liste `l.map(f)`. Rezultat je `true` ako i samo ako sve vrednosti iz `l` zadovoljavaju uslov `f`. Nakon što se pronađe prvi element liste za koji `f` ima vrednost `false`, prestaje se sa izračunavanjem. Tip je: `List['a] * ('a -> bool) -> bool`
`l.exists(f)`	Disjunkcija elemenata liste `l.map(f)`. Rezultat je `true` ako i samo postoji vrednost u `l` koja zadovoljava uslov `f`. Nakon što se pronađe prvi element liste za koji `f` ima vrednost `true`, prestaje se sa izračunavanjem. Tip je: `List['a] * ('a -> bool) -> bool`

Tabela 13: Najvažnije funkcije višeg reda u standardnoj biblioteci

2.3.7. Biblioteke

Od više srodnih definicija može se napraviti *biblioteka*. Biblioteka je programska celina koja omogućava jednostavno pozivanje na imena definisana u njoj iz drugih programskih celina. Biblioteke su u programskom jeziku WAFL namenjene za sistematizovanje definicija koje se u okviru aplikacije upotrebljavaju u više programa.

Definisanje biblioteka

Biblioteka se definiše u zasebnoj datoteci sa podrazumevanim nastavkom `wlib`. Definicija biblioteke ima oblik:

```
<biblioteč̌ki_modul> ::=
    library [<ime>] { <definicija> {<definicija>} }
    [<where_podizraz>]
```

Biblioteci odgovaraju dva prostora imena:

- *javni prostor imena biblioteke*, kome pripadaju imena uvedena definicijama u prvom bloku definicija, neposredno iza imena biblioteke;
- *privatni prostor imena biblioteke*, kome pripadaju imena uvedena definicijama u opcionom bloku definicija koji je označen ključnom reči *where*.

U okviru javnog prostora imena uvode se definicije koje će se upotrebljavati u drugim programskim celinama. Privatni prostor imena služi za pomoćne definicije, koje se upotrebljavaju u telima definicija u biblioteci.

Vidljivost imena uvedenih u prostorima imena biblioteke definiše se narednim pravilima:

- imena iz privatnog prostora imena biblioteke vide se u javnom prostoru imena biblioteke (prema tome, a u skladu sa ranijim definicijama, vide se i u svim prostorima imena obuhvaćenim bilo javnim bilo privatnim prostorom imena biblioteke);
- imena iz javnog prostora imena biblioteke vide se u prostorima imena u kojima se biblioteka referiše, shodno pravilima o načinu referisanja elemenata biblioteke;
- naziv biblioteke ne pripada nijednom prostoru imena i ima samo deklarativnu svrhu.

Naredni primer predstavlja biblioteku sa dve javne funkcije za rad sa prostim brojevima i četiri pomoćne funkcije:

```
library prostiBrojevi {
    // funkcija proverava da li je broj n prost
    prost(n) =
        if (n==2) or (n==3) then true
        else if n<2 or (n%2)==0 then false
        else !imaDelilaca(n,3,koren(n)+1);

    // funkcija izračunava n-ti prost broj
    ntiProst(n) =
        if n>3 then prviVećiProst(ntiProst(n-1))
        else if (n==2) or (n==1) then n+1
        else 0;
    }
where {
    // f.ja proverava da li broj n ima neparnih delilaca
    // u opsegu [poč,kraj], uz pretpostavku da je poč neparan
    imaDelilaca(n,poč,kraj) =
        if poč>kraj then false
        else if (n%poč)==0 then true
        else imaDelilaca(n,poč+2,kraj);

    // f.ja računa gornju celobrojnu aproksimaciju kv. korena
    koren(n) = k1(n,1,n/2);
    k1(n,i,j) =
        if i==j then i
        else if i==(j-1) then j
        else k1(n,j,(j+(n/j))/2);

    // f.ja izračunava prvi prost broj veći od n (n je neparan)
    prviVećiProst(n) =
        if prost(n+2) then n+2 else prviVećiProst(n+2);
    };
```

Upotreba biblioteka

Biblioteke se u nekom modulu mogu upotrebljavati na dva osnovna načina: deklarisanjem i uvođenjem u globalni prostor imena modula. Pod pojmom *modul* (ili *programska celina*) podrazumevamo jedan programski tekst, tj. jednu datoteku, bilo da se radi o programu, biblioteci, klasi ili šablonu.

Deklarisanjem se ime biblioteke uvodi u prostor imena, a imenima definisanim u biblioteci se pristupa posredstvom operatora za referenciranje elemenata biblioteke. Deklaracija biblioteke ima uobičajenu formu definicije sa odgovarajućim telom definicije:

```
<deklaracija_biblioteke> ::=
    <ime> = library [[file | ( extern [<ime>] ) ] <naziv>] ;
```

gde je `<naziv>` niska koja predstavlja naziv datoteke sa definicijom biblioteke. Po potrebi, a u skladu sa specifičnostima implementacije, osim naziva se može navoditi i putanja ili čak URL teksta definicije biblioteke.

Imenima elemenata biblioteke pristupa se pomoću *operatora za referenciranje elemenata biblioteke*. u obliku:

```
<naziv_biblioteke>::<naziv_elementa_biblioteke>
```

Formalnom sintaksom programskog jezika WAFL na sledeći način je definisano *kvalifikovano ime*:

```
<kvalifikovano_ime> ::= [<kvalifikovano_ime>::]<ime>
```

U narednom primeru se pomoću operatora za referenciranje elemenata biblioteke pristupa funkciji iz deklarisane biblioteke:

```
prLib::ntiProst(42) where {
    prLib = library file 'prosti.wlib';
    };
```

U deklaraciji biblioteke se ne mora navoditi originalno ime biblioteke, već se ona može upotrebljavati pod drugim imenom, kao što se u prethodnom primeru biblioteka `prostiBrojevi` upotrebljava pod imenom `prLib`.

Uvođenjem biblioteke u globalni prostor imena programske celine, u taj prostor se uvode sva imena iz javnog prostora imena biblioteke. Uvođenje biblioteke u globalni prostor imena se vrši pomoću direktive `#using`[10]. Ova direktiva se mora nalaziti na samom početku teksta programa ili drugog modula u obliku:

```
#using [file | ( extern [<ime>] )] <naziv>
```

gde je značenje parametara zapisa isto kao u slučaju deklaracije biblioteke. Jedina razlika je što se u okviru direktive `#using` mora navesti naziv datoteke sa definicijom biblioteke.

Poput prethodnog primera i naredni program izračunava 42. prost broj, ali ovaj put uz uvođenje biblioteke u globalni prostor imena:

```
#using 'prosti.wlib'

ntiProst(42);
```

Ako se u nazivu biblioteke, bilo pri deklarisanju ili uvođenju biblioteke, ne navede putanja, upotrebljava se podrazumevana putanja. Podrazumevana putanja do direktorijuma sa bibliotekama definiše se na nivou servisa (videti *Dodatak D - Parametri servisa*, na strani 91.). Ako se

[10] Direktiva using, opisana u narednim odeljcima, još nije implementirana.

u nazivu biblioteke ne navede nastavak (oznaka tipa), onda se podrazumeva da se radi o sistemskoj (ugrađenoj) biblioteci. Ako se u deklaraciji izostavi naziv datoteke, traži se najpre podrazumevana biblioteka sa nazivom koji odgovara deklarisanom imenu biblioteke, pa onda datoteka sa tim nazivom i nastavkom `wlib` u podrazumevanom direktorijumu:

```
library primer {
    studenti = library; // 'studenti.wlib' u podrazumevanom dir.
    Report = library;   // sistemska biblioteka 'Report'
};
```

Funkcijske datoteke

Posebna forma biblioteka su *funkcijske datoteke*. Funkcijska datoteka sadrži definiciju tačno jedne funkcije. Referisanjem na funkcijsku datoteku može se u drugim programskim celinama deklarisati i upotrebljavati funkcija koja je u njoj definisana. Podrazumevani nastavak za funkcijske datoteke je `wfnc`.

Sadržaj funkcijske datoteke ima formu:

```
<funkcijski_modul> ::=
    [ function [<ime>] <formalni_parametri> = ]
    <telo_funkcije> ;
```

Ako funkcija ima argumente, neophodno je navesti ključnu reč `function` i listu formalnih parametara. Ime funkcije, kao i u slučaju imena biblioteke, ima samo dokumentacionu namenu. U narednom primeru je predstavljen sadržaj jedne funkcijske datoteke:

```
function max3( x, y, z ) = max( max(x,y), z )
where {
    max(x,y) = if x>y then x else y;
    };
```

Da bi se funkcija definisana u nekoj funkcijskoj datoteci mogla upotrebljavati u programu, neophodno je deklarisati funkciju. Deklaracija funkcije je po formi veoma slična deklaraciji biblioteke:

```
<deklaracija_funkcije> ::=
    <ime> [<formalni_parametri>] = function [file] <naziv> ;
```

Pri deklarisanju funkcije se njeni argumenti navode opciono. Ukoliko se navedu, njihov broj mora odgovarati stvarnom broju argumenata bibliotečke funkcije. Deklarisana funkcija se upotrebljava na potpuno isti način kao uobičajeno definisane funkcije. Primer upotrebe funkcije iz prethodnog primera je:

```
max3( 12, 34, 24 )
where {
    max3 = function file 'max3.wfnc';
    }
```

Standardne biblioteke

Definicija programskog jezika WAFL sadrži i definicije nekoliko standardnih biblioteka koje moraju biti sadržane u svakoj implementaciji programskog jezika. Naravno, svaka implementacija

može obuhvatiti i neke druge biblioteke. Međutim, činjenica da je biblioteka sadržana u implementaciji ne implicira da su imena iz te biblioteke vidljiva u programima.

Pregled svih standardnih biblioteka sadrži *Dodatak B - Standardna bibliotek*, na strani 81.

Podrazumevane biblioteke

Da bi programeri mogli prema potrebi zameniti ugrađene biblioteke svojim ili nekim drugim bibliotekama u kojima su isti elementi drugačije definisani, neophodno je da na nivou aplikacije[11] bude moguće definisanje *skupa podrazumevanih biblioteka*. Sve biblioteke iz skupa podrazumevanih biblioteka neke aplikacije automatski se uvode u globalni prostor imena svakog programa koji pripada toj aplikaciji. Takve biblioteke nije potrebno uvoditi direktivom `#using`.

Način podešavanja parametara na nivou aplikacije opisan je u dodatku *Parametri servisa* na strani 91.

Biblioteke pisane na drugim programskim jezicima[12]

Dopuštena je upotreba biblioteka pisanih na drugim programskim jezicima, kao što su C i C++. Time se omogućava pisanje efikasnih biblioteka prilagođenih specifičnim primenama i okruženjima.

Upotreba takvih biblioteka mora biti u skladu sa upotrebom biblioteka pisanih na programskom jeziku WAFL, uz razliku u ključnoj reči koja u deklaraciji biblioteke stoji iza ključne reči `library`. Umesto opcione ključne reči `file` mora stajati ključna reč `extern` koja označava da se radi o biblioteci koja nije pisana na programskom jeziku WAFL. Konkretna implementacija može uvesti i posebne oznake za različite programske jezike. Sam način pisanja takvih biblioteka nije definisan programskim jezikom WAFL već se prepušta da bude određen svakom konkretnom implementacijom.

Sledi primer deklaracije biblioteke napisane na programskom jeziku C++ i prevedene u formu izvršnog programa:

```
imageLib = library extern cpp "image.exe";
```

2.3.8. Izuzeci[13]

...

2.4. Strukture podataka

Svi podaci u programskom jeziku WAFL dele se na *privremene podatke* i *trajne podatke*. Privremeni podaci su svi podaci kojima se operiše tokom izračunavanja programa. Trajni podaci predstavljaju podatke pohranjene u okviru posebnih sistema za upravljanje resursima, kao što su *baze podataka* i *sistemi datoteka*. Ovo poglavlje je posvećeno privremenim podacima i sistemu datoteka, dok se pitanja upotrebe baza podataka razmatraju nešto kasnije.

---

[11] U ovom kontekstu aplikacija (Veb servis) se smatra skupom programa (generatora Veb stranica).

[12] Podrška za biblioteke pisane na drugim jezicima još nije implementirana.

[13] Rad sa izuzecima nije implementiran. Gde god se pominju izuzeci, to se može zanemariti.

2.4.1. Tipovi podataka

Tipove podataka u programskom jeziku WAFL možemo podeliti na: *proste tipove podataka*, *primitivne kolekcije* i *apstraktne tipove podataka*.

- Prosti tipovi podataka, tipovi `int`, `float`, `bool` i `string`, obrađeni su na samom početku izlaganja definicije programskog jezika WAFL.
- Primitivne kolekcije podataka predstavljaju jednostavne kolekcije vrednosti istog tipa. Programski jezik WAFL uključuje tri tipa primitivnih kolekcija: *liste*, *nizove* i *kataloge*.
- Apstraktni tipovi podataka su u programskom jeziku WAFL realizovani upotrebom principa objektno orijentisanog programiranja i definišu se u vidu *klasa*.

2.4.2. Označavanje tipova podataka

U programima na programskom jeziku WAFL ne navode se tipovi podataka, osim u slučaju pravljenja objekata i u izrazima za hvatanje izuzetaka. U ovom odeljku se definiše način označavanja tipova da bi se kasnije upotrebljavao:

- za izveštavanje o zaključenim tipovima od strane implementacije programskog jezika;
- u tekstovima koji se odnose na programski jezik WAFL, uključujući i ovaj rad.

Imena prostih tipova podataka zapisuju se malim slovima: `int`, `float`, `bool` i `string`. Imena kolekcija i standardnih klasa počinju velikim slovom. Oznaka tipa funkcije ima oblik:

```
t1 * ... * tn -> tr
```

gde su `t1` do `tn` tipovi argumenata, a `tr` tip rezultata. Tip funkcije koja nema argumente je:

```
() -> tr
```

Tip nazivamo *prosto polimorfnim* ako se odnosi na vrednost bilo kog u datom kontekstu dopuštenog tipa. Tip je polimorfan, ali ne i prosto polimorfan, ako je bar delimično poznata njegova struktura. Oznake prosto polimorfnih tipova podataka imaju formu:

```
'x
```

gde je *x* neko slovo ili reč, i nazivaju se *tipskim promenljivim*.

Tip nekog imena zapisujemo u obliku:

```
<ime> : <tip>
```

U skladu sa time, tip polimorfne funkcije `f` definisane sa:

```
f(x,y,z) = if x then y else z;
```

čiji su drugi i treći argument i rezultat su prosto polimorfni i istog su tipa, zapisuje se kao:

```
f : bool * 'a * 'a -> 'a
```

Poseban oblik polimorfnih tipova su *zbirni tipovi*. Zbirnim tipovima nazivamo polimorfne tipove koji ne predstavljaju bilo koji tip (kao u slučaju prosto polimorfnih tipova), već samo konačan skup tipova. Zbirni tipovi se zapisuju navođenjem skupa, koji predstavlja domen polimorfnog tipa, ispred oznake samog polimorfnog tipa:

```
{ <tip>, <tip> {,<tip>} }['<ime>]
```

Na primer, tip funkcije `asInt`, za konverziju necelobrojnih prostih tipova u cele brojeve, može da se predstavi kao:

```
{float, bool, string}['a] -> int
```

Pri tome se smatra da su tipovi `{<tip>}` i `<tip>` međusobno ekvivalentni.

Za neke skupove tipova koji se često pojavljuju uvode se posebne oznake. Tako oznaka `Numeric` predstavlja skup `{int, float}`, a oznaka `Value` predstavlja skup tipova `{int, float, string}`. Na primer, tipovi ugrađenih operatora `+` i `-` su:

```
+ : Value['a] * Value['a] -> Value['a]
- : Numeric['a] * Numeric['a] -> Numeric['a]
```

Funkcija `g`, u narednom primeru, je polimorfna, pri čemu prva dva argumenta imaju jedan polimorfan tip (određen kao `Value['a]` zbog tipa operatora poređenja), a treći argument i rezultat drugi polimorfni tip (određen kao `Numeric['b]` zbog tipa operatora množenja):

```
g(a,b,c) = if a>b then c else c*c;
g : Value['a] * Value['a] * Numeric['b] -> Numeric['b]
```

Pri upotrebi polimorfne funkcije u programu ustanovljava se tip njene instance. Recimo, u narednom programu, funkcija `g` je polimorfna i ima isti polimorfan tip kao u prethodnom primeru, ali njena instanca koja se poziva u glavnom izrazu programa ima nepolimorfan *instanciran tip*: `int * int * float -> float`:

```
g(5,6,7.5) where { g(a,b,c) = if a>b then c else c*c; }
```

Ako se prilikom ustanovljavanja tipova u glavnom izrazu programa neka polimorfna funkcija ne može instancirati na odgovarajući način, tj. potreban instancirani tip nije specijalizacija njenog polimorfnog tipa, to predstavlja grešku koja se mora prepoznati u fazi prevođenja programa. Na primer, bilo bi neispravno upotrebljavati funkciju `g` na sledeći način, jer prva dva argumenta moraju biti istog tipa:

```
g(5,6.3,7.5) where { g(a,b,c) = if a>b then c else c*c; }
```

2.4.3. Lista

U većini funkcionalnih programskih jezika, pa i u programskom jeziku WAFL, liste predstavljaju osnovni tip kolekcije. Lista je uređena struktura podataka koja može sadržati više elemenata istog tipa. Tip elemenata liste nije ni na koji način ograničen, pa je oznaka opšteg tipa liste `List['a]`.

Lista je prazna ako i samo ako nema nijedan element. Za označavanje prazne liste koristi se konstanta `nil`. Konstanta `nil` generalno ima tip `List['a]`, ali u konkretnim situacijama se njen

tip može suziti. Neprazna lista ima *glavu* i *rep*. Glavu liste predstavlja njen prvi element, dok je rep liste lista koja se sastoji od svih njenih elemenata osim prvog. Poredak elemenata u repu liste odgovara poretku elemenata u listi.

Lenja lista je lista čiji neki deo nije izračunat. Lenje liste se upotrebljavaju u cilju povećavanja efikasnosti u slučajevima kada je relativno velika verovatnoća da neće svi elementi liste biti upotrebljeni u izračunavanju. Primer lenje liste je lista koja predstavlja rezultat upita nad bazom podataka, jer se u daljem izračunavanju vrlo često upotrebljava samo deo rezultata upita, a ne čitav rezultat.

Naredna tabela predstavlja standardne funkcije i operatore za rad sa listama:

Funkcija	Vrednost i tip
`hd(l)`	glava liste: `List['a]->'a`
`tl(l)`	rep liste: `List['a]->List['a]`
`:`	Operator za formiranje liste. Levi operand predstavlja glavu, a desni rep rezultujuće liste. Operator `:` je desno asocijativan. Njegov tip je: `'a * List['a] -> List['a]`
`empty(l)`	proverava da li je lista `l` prazna: `List['a]->bool`
`length(l)`	izračunava dužinu liste `l`: `List['a] -> int`
`filter(l,f)`	izračunava listu elemenata liste `l` koji zadovoljavaju uslov `f`, uz očuvanje poretka elemenata: `List['a] * ('a->bool) -> List['a]`
`map(l,f)`	izračunava listu vrednosti funkcije `f` primenjene redom na elemente liste `l`: `List['a] * ('a->'b) -> List['b]`
`forall(l,f)`	Konjunkcija elemenata liste `l.map(f)`. Rezultat je `true` ako i samo ako svi elementi liste `l` zadovoljavaju uslov `f`. Nakon što se pronađe prvi element liste za koji `f` ima vrednost `false`, prestaje se sa izračunavanjem. Tip je: `List['a] * ('a->bool) -> bool`
`exists(l,f)`	Disjunkcija elemenata liste `l.map(f)`. Rezultat je `true` ako i samo postoji element liste `l` koji zadovoljava uslov `f`. Nakon što se pronađe prvi element liste za koji `f` ima vrednost `true`, prestaje se sa izračunavanjem. Tip je:: `List['a] * ('a->bool) -> bool`
`aggregate(l,f,n)`	agregira od kraja prema početku sve elemente `x` liste `l` funkcijom $S_{i+1} = f(x, S_i)$, gde je $S_0 = n$: `List['a] * ('a*'b -> 'b) * 'b -> 'b`
`leftAggregate(l,f,n)`	agregira od početka prema kraju sve elemente `x` liste `l` funkcijom $S_{i+1} = f(x, S_i)$, gde je $S_0 = n$: `List['a] * ('a*'b -> 'b) * 'b -> 'b`
`subList(l,n,m)`	Preskače prvih `n-1` elemenata liste `l` i izdvaja samo narednih `m` elemenata. Ako je `m=0`, onda se izdvaja do kraja liste. Ako je `n<0`, onda se broji od kraja, pa izraz `sublist(l,-3,3)` izdvaja listu koja sadrži samo poslednja tri elementa liste `l`. Tip funkcije `subList` je:

Funkcija	Vrednost i tip
	`List['a] * int * int -> List['a]`
`longerThan(l,n)`	Proverava da li lista `l` ima više od `n` elemenata. Ako je `l` lenja lista, izračunava samo prvih `n+1` elemenata liste. Tip funkcije `longerThan` je: `List['a] * int -> bool`
`forced(l)`	Rezultat funkcije `forced` je lista `l`. Pri tome, ako je lista `l` lenja, svi njeni elementi se izračunavaju. Namenjena je za skraćivanje trajanja zauzeća resursa u slučajevima kada se zna da će u izračunavanju koje će duže potrajati biti upotrebljeni svi elementi lenje liste[14]. Tip ove funkcije je: `List['a] -> List['a]`
`listAsString(l)`	Izračunava tekstualnu reprezentaciju liste `l`. Tip funkcije je: `List['a] -> string`

Tabela 14: Standardne funkcije nad listama

Konstantna lista označava se u formi:

```
[ [<izraz> { , <izraz> }] ]
```

pri čemu svi elementi moraju biti istog tipa. U skladu sa ovakvim načinom zapisivanja, prazna lista se može zapisati i kao `[]`. Zapisi `nil` i `[]` su potpuno ravnopravni. Pokušaj pristupanja glavi ili repu prazne liste dovodi do izuzetka `EEmptyList`. Zbog toga što je operator `:` desno asocijativan, naredna dva zapisa liste su ekvivalentna:

```
[ 1, 2, 3, 4, 5 ]
1 : 2 : 3 : 4 : 5 : nil
```

U narednom primeru se izračunava suma prvih 75 prirodnih brojeva:

```
aggregate( prvihN(75), add, 0 )
where {
    add(x,y) = x+y;
    prvihN(n) = if n<1 then []
                else n : prvihN(n-1);
    }
```

2.4.4. Niz

Osnovni nedostatak liste je prilična neefikasnost ako je potrebno pristupati elementima preko reda, recimo na osnovu mesta na kome se nalaze u listi. *Niz* je tip kolekcije koji omogućava efikasniji indeksni pristup. Oznaka opšteg tipa niza je `Array['a]`.

Standardna biblioteka sadrži jedan operator i nekoliko funkcija za rad sa nizovima:

Funkcija	Vrednost i tip
`asList(a)`	Izračunava listu sastavljenu, redom, od elemenata niza `a`.

[14] Važnost ove funkcije opisana je u odeljku posvećenom upotrebi baza podataka u programskom jeziku WAFL (*2.7 Baze podataka*, na strani 56).

Funkcija	Vrednost i tip
	Tip funkcije je: `Array['a] -> List['a]`
`asArray(l)`	Izračunava niz sastavljen, redom, od elemenata liste `l`. Tip funkcije je: `List['a] -> Array['a]`
`a[i]`	Izračunava `i`-ti element niza `a`: `Array['a] * int -> 'a`
`array(n,x)`	Pravi niz dužine `n` (`n` mora biti veće od nule), čiji svi elementi imaju vrednost `x`: `int * 'a -> Array['a]`
`setEl(a,i,x)`	Izračunava niz istog tipa i veličine kao niz `a`, čiji su svi elementi osim `i`-tog jednaki odgovarajućim elementima niza `a`, a `i`-ti element ima vrednost `x`. Tip funkcije je: `Array['a] * int * 'a -> Array['a]`
`addEl(a,x)`	Izračunava niz istog tipa kao niz `a`, ali duži za jedan element. Svi elementi novog niza, osim poslednjeg, jednaki su odgovarajućim elementima niza `a`, a poslednji element ima vrednost `x`. Tip funkcije je: `Array['a] * 'a -> Array['a]`
`length(a)`	Izračunava dužinu niza `a`: `Array['a] -> int`
`subArray(a,n,m)`	Izračunava niz sastavljen, redom, od elemenata niza `a` sa indeksima `n` do `n+m-1`. Argument `m` mora biti veći od nule. Tip funkcije je: `Array['a] * int * int -> Array['a]`
`emptyArray()`	Izračunava prazan niz. Tip funkcije je: `() -> Array['a]`

Tabela 15: Funkcije i operatori nad nizovima

Konstantan niz zapisuje se u formi:

```
{ <izraz> { , <izraz> } }
```

pri čemu svi elementi moraju biti istog tipa. Prazan niz se ne može zapisati, ali se može napraviti funkcijom `emptyArray`.

Indeksni operator omogućava pristup elementima niza putem indeksa njihove pozicije u nizu. Indeks mora biti ceo broj u ispravnom opsegu. Indeks prvog elementa niza je `0`, a indeks poslednjeg elementa je za jedan manji od broja elemenata niza. Pokušaj pristupanju elementu niza sa neispravnim indeksom proizvodi izuzetak `EInvalidIndex`.

2.4.5. Katalog

Katalog je uopštenje niza čiji tip indeksa ne mora biti samo celobrojni tip. Oznaka opšteg tipa kataloga je `Map['a]['b]`, gde je `'a` tip indeksa a `'b` tip elemenata kataloga. Nizovi se u svim aspektima upotrebe mogu smatrati specijalizacijom kataloga, tj. katalozima tipa: `Map[int]['b]`.

Katalozi se upotrebljavaju u situacijama kada je potrebno elementima kolekcije pristupati po ključu koji nije redni broj. U radu sa bazama podataka i pri pisanju Veb aplikacija najčešće se upotrebljavaju katalozi čiji su i ključevi i vrednosti tipa `string`.

Zbog toga što nisu uvek unapred poznati ključevi kataloga, ključevima i vrednostima u katalogu moguće je pristupati i prema njihovom *rednom broju* u katalogu. Kako je raspored elemenata u katalogu zavisan od implementacije, redni broj elemenata u katalogu se može razlikovati između implementacija, ali je neohodno da vrednost sa nekim rednim brojem odgovara ključu sa istim rednim brojem. Redni brojevi elemenata kataloga imaju vrednosti u opsegu od nula do broja za jedan manjeg od broja elemenata kataloga.

Standardna biblioteka definiše jedan operator i nekoliko funkcija za rad sa katalozima:

Funkcija	Vrednost : Tip
`createMap(ak,av)`	Izračunava katalog u kome ključevima iz niza `ak` redom odgovaraju vrednosti iz niza `av`. Tip funkcije `createMap` je: `Array['a] * Array['b] -> Map['a]['b]`
`c[k]`	izračunava element kataloga `c` kome odgovara ključ `k`: `Map['a]['b] * 'a -> 'b`
`setEl(c,k,x)`	Izračunava katalog istog tipa i veličine kao katalog `c`, čiji su svi elementi osim elementa sa ključem `k` jednaki odgovarajućim elementima kataloga `a`, a element sa ključem `k` ima vrednost `x`. Ako u katalogu `c` ne postoji element sa ključem `k`, tada je rezultat katalog koji sadrži sve neizmenjene elemente kataloga `c` i još element sa ključem `k` i vrednošću `x`. Tip funkcije je: `Map['a]['b] * 'a * 'b -> Map['a]['b]`
`addEl(c,k,x)`	Izračunava katalog istog tipa kao katalog `c`, ali sa jednim elementom više. Novi katalog sadrži sve elemente kataloga `c` i još jedan novi element sa ključem `k` i vrednošću `x`. Ako u katalogu `c` postoji element sa ključem `k` dolazi do izuzetka `EExistingKey`. Tip ove funkcije je: `Map['a]['b] * 'a * 'b -> Map['a]['b]`
`length(c)`	Izračunava broj elemenata kataloga `c`. Tip funkcije je: `Map['a]['b] -> int`
`key(c,i)`	Izračunava ključ elementa kataloga `c` sa rednim brojem `i`. Tip ove funkcije je: `Map['a]['b] * int -> 'a`
`value(c,i)`	Izračunava vrednost elementa kataloga `c` sa rednim brojem `i`. Tip funkcije je: `Map['a]['b] * int -> 'b`
`keys(c)`	Izračunava niz koji se sastoji od svih ključeva kataloga `c`. Ključevi su uređeni po njihovom rednom broju, u rastućem redosledu. Tip funkcije `keys` je: `Map['a]['b] -> Array['a]`
`values(c)`	Izračunava niz koji se sastoji od svih vrednosti kataloga `c`. Vrednosti su uređene po njihovom rednom broju, u rastućem redosledu. Tip funkcije `values` je: `Map['a]['b] -> Array['b]`
`emptyMap()`	Izračunava prazan katalog. Tip funkcije `emptyMap` je: `() -> Map['a]['b]`

Funkcija	Vrednost : Tip
`findKey(c,k)`	Izračunava redni broj elementa sa ključem `k` u katalogu `c`. Ako u katalogu ne postoji element sa ključem `k`, rezultat je `-1`. Tip je: `Map['a]['b] * 'a -> int`
`findValue(c,x)`	Izračunava redni broj elementa sa vrednošću `x` u katalogu `c`. Ako u katalogu ne postoji element sa vrednošću `x`, rezultat je `-1`. Tip je: `Map['a]['b] * 'b -> int`
`hasKey(c,k)`	Proverava da li postoji elementa sa ključem `k` u katalogu `c`. Tip je: `Map['a]['b] * 'a -> bool`
`hasValue(c,x)`	Proverava da li postoji elementa sa vrednošću `x` u katalogu `c`. Tip je: `Map['a]['b] * 'b -> bool`

Tabela 16: Funkcije i operatori nad katalozima

Konstantan katalog zapisuje se u formi:

```
[ < ključ>, <vrednost> > { , < ključ>, <vrednost> > } ]
```

pri čemu svi ključevi moraju biti međusobno istog tipa i sve vrednosti moraju da imaju međusobno isti tip. Prazan katalog se ne može zapisati, ali se može napraviti funkcijom `emptyMap`.

Pokušaj pristupanja elementu kataloga uz navođenje ključa koji ne postoji u katalogu dovodi do izuzetka `EInvalidKey`. Pokušaj formiranja kataloga sa nizovima različite dužine, ili sa nizom ključeva u kome se ponavlja neki ključ, dovodi do izuzetka `EIncompatibleKeys`. Pokušaj dodavanja elementa sa ključem koji već postoji dovodi do izuzetka `EExsistingKey`.

2.4.6. Datoteke[15]

Rad sa datotekama u programskom jeziku WAFL zamišljen je tako da se na funkcionalnom nivou operacije koje su uobičajene za datoteke u što većoj meri prenesu na niske. Implementacija podsistema za rad sa datotekama mora podržavati predstavljene koncepte na efikasan i pouzdan način.

Funkcije za rad sa datotekama dele se na:

- funkcije za pristupanje sadržaju datoteka;
- funkcije za manipulisanje datotekama i
- funkcije za manipulisanje direktorijumima.

Funkcije za pristupanje sadržaju datoteka su:

Funkcija	Opis i tip
`fileRead(f)`	Izračunava nisku koja predstavlja sadržaj datoteke sa nazivom `f`. Tip funkcije `fileRead` je: `string -> string`
`fileWrite(f,txt)`	Zamenjuje sadržaj datoteke `f` niskom `txt`. Ako je pisanje uspelo rezultat je `true`, a inače je `false`. Tip funkcije `fileWrite` je:

[15] Podrška za rad sa datotekama još nije implementirana. Biće implementirana u vrlo skoroj budućnosti.

Funkcija	Opis i tip
	`string * string -> bool`
`fileReadRecords(f,n)`	Izračunava listu niski koje redom predstavljaju slogove veličine `n` bajtova iz datoteke `f`. Tip funkcije `fileReadRecords` je: `string * int -> List[string]`
`fileReadRecordsA(f,n)`	Izračunava niz niski koje redom predstavljaju slogove veličine `n` bajtova iz datoteke `f`. Tip funkcije `fileReadRecordsA` je: `string * int -> Array[string]`
`fileWriteRecord(f,n,txt)`	Zamenjuje sadržaj `n`-tog sloga datoteke `f` niskom `txt`. Ako je pisanje uspelo rezultat je `true`, a inače je `false`. Tip funkcije `fileWriteRecord` je: `string * int * string -> bool`
`fileAppend(f,txt)`	Dopisuje nisku `txt` na kraj datoteke `f`. Ako je pisanje uspelo rezultat je `true`, a inače je `false`. Tip funkcije je: `string * string -> bool`

Tabela 17: Funkcije za pristupanje sadržaju datoteka

Funkcije `fileWrite`, `fileWriteRecord` i `fileAppend` smeju se upotrebljavati samo u okviru transakcija. Transakcije su detaljnije opisane u odeljku *2.7.4 Transakcije* na strani 66.

Funkcije za manipulisanje datotekama su:

Funkcija	Opis i tip
`fileExists(f)`	Proverava da li postoji datoteka sa nazivom `f`. Tip funkcije je: `string -> bool`
`fileGetAttrs(f)`	Izračunava atribute datoteke `f`. Rezultat je niska koja opisuje atribute. Tip funkcije je: `string -> string`
`fileSetAttrs(f,a)`	Postavlja atribute `a` datoteke `f`. Rezultat je `true` ako je postavljanje uspešno okončano. Tip funkcije je: `string * string -> bool`
`fileCopy(s,t)`	Kopira datoteku `s` u datoteku `t`. Rezultat je `true` ako je kopiranje uspešno okončano. Tip funkcije je: `string * string -> bool`
`fileMove(s,t)`	Premešta datoteku `s` u datoteku `t`. Rezultat je `true` ako je premeštanje uspešno okončano. Tip funkcije je: `string * string -> bool`
`fileRemove(f)`	Briše datoteku `f`. Rezultat je `true` ako je brisanje uspešno okončano. Tip funkcije je: `string -> bool`

Tabela 18: Funkcije za manipulisanje datotekama

Funkcije `fileSetAttrs`, `fileCopy`, `fileMove` i `fileRemove` se smeju upotrebljavati samo u okviru transakcija.

Funkcije za manipulisanje direktorijumima su:

Funkcija	Opis i tip
`dirFiles(d)`	Formira listu naziva svih datoteka u direktorijumu `d`. Tip funkcije `dirFiles` je: `string -> List[string]`
`dirSubdirs(d)`	Formira listu naziva svih poddirektorijuma direktorijuma `d`. Tip funkcije je: `string -> List[string]`
`dirCreate(d)`	Pravi direktorijum `d`. Rezultat je `true` ako je direktorijum uspešno napravljen. Tip funkcije je: `string -> bool`
`dirRemove(d)`	Uklanja direktorijum `d`. Rezultat je `true` ako je direktorijum uspešno uklonjen. Tip funkcije je: `string -> bool`

Tabela 19: Funkcije za manipulisanje direktorijumima

Funkcije `dirCreate` i `dirRemove` se smeju upotrebljavati samo u okviru transakcija. Implementacija programskog jezika WAFL mora obezbediti transakcione mehanizme za rad sa datotekama.

Izuzeci pri radu sa datotekama

Navedene funkcije za rad sa datotekama mogu se podeliti u dve grupe prema tome da li u slučaju neuspeha imaju neki specifičan rezultat ili proizvode izuzetak.

Funkcije koje imaju rezultat logičkog tipa koja ukazuje da li je funkcija uspešno izračunata ne proizvode izuzetak u slučaju greške. Međutim, istinitosna vrednost `false`, koju takve funkcije imaju za rezultat u slučaju greške, ponekad nije dovoljno opisna. Zbog toga se svaka od njih može upotrebljavati u još jednom obliku koji podrazumeva da se u slučaju greške proizvodi izuzetak, a u slučaju da je sve u redu dobija se kao rezultat vrednost `true`. Novi oblik se dobija dopisivanjem slova `E` na kraj naziva funkcije. Novi oblik funkcije zadržava isti tip. Na primer, takav oblik funkcije `fileWrite` predstavlja funkcija `fileWriteE`.

Funkcije koje nemaju rezultat koji ukazuje na uspešnost operacije proizvode izuzetak u slučaju greške.

Standardna biblioteka obuhvata četiri tipa izuzetaka koji se proglašavaju pri radu sa datotekama. U narednoj tabeli navedene su funkcije pri čijem se izračunavanju pojavljuju odgovarajući tipovi izuzetaka:

Klasa	Opis izuzetka
`FileError`	fileExists, fileGetAttrs, fileSetAttrsE, fileCopyE, fileMoveE, fileRemoveE, dirFiles, dirSubdirs, dirCreateE, dirRemoveE
`FileOpenError`	fileRead, fileReadRecords, fileReadRecordsA, fileWriteE, fileWriteRecordE, fileAppendE
`FileReadError`	fileRead, fileReadRecords, fileReadRecordsA,
`FileWriteError`	fileWriteE, fileWriteRecordE, fileAppendE

Tabela 20: Funkcije koje izazivaju izuzetke pri radu sa datotekama

Zaključavanje datoteka

Kada se čitaju van transakcija, datoteke se otvaraju u deljivom režimu koji omogućava najviši nivo konkurentnog rada na konkretnom operativnom sistemu. Osnovni razlog za to je izbegavanje suvišnog zaključavanja podataka u situacijama kada se pretpostavlja da programi uglavnom čitaju podatke.

Nasuprot tome, pri bilo kakvom pristupanju u okviru transakcija, datoteke se otvaraju u ekskluzivnom režimu, koji obezbeđuje najviši nivo izolovanosti programa.

Tekstualne datoteke sa separatorima

Iako je staranje o podacima na kojima počiva Veb aplikacija ili dinamička Veb lokacija poželjno prepustiti nekom SUBP, u slučaju manje složenih lokacija podaci se često čuvaju u datotekama. Jedan od uobičajenih formata za čuvanje uređenih podataka u datotekama jesu tekstualne datoteke sa separatorima. Obično se uvodi jedan simbol kao separator podataka (često je to upravo znak za novi red `'\n'`) i jedan simbol kao separator vrednosti atributa u okviru podataka (za razdvajanje atributa se često upotrebljava simbol `'|'`).

Programski jezik WAFL omogućava rad sa takvim podacima, ali podrška nije obuhvaćena bibliotekom funkcija za rad sa datotekama već bibliotekom funkcija za rad sa niskama. Funkcija `strSplit` izdvaja iz niske listu podniski međusobno razdvojenih datim separatorom, a funkcija `strJoin` od takve liste niske formira polaznu nisku njihovim spajanjem. Pomenute funkcije su opisane u odeljku *Niske* sa početkom na strani 15. Da bi se obezbedio efikasan rad sa ovakvim datotekama poželjno je da ove funkcije budu implementirane kao lenje, tj. da ne izračunavaju više nego što je zaista potrebno.

2.5. Klase[16]

...

2.6. Podrška za Veb

Prilagođenost programskog jezika WAFL pisanju programa za automatsko generisanje dokumenata na Vebu ogleda se kroz više aspekata. U ovom odeljku posvećena je pažnja konceptu tekstualnih šablona i funkcijama standardne biblioteke koje omogućavaju jednostavniji pristup određenim informacijama specifičnim za okruženje u kome se generišu dokumenti na Vebu. U posebnom odeljku, pod naslovom *Dijalozi*, obrađena su pitanja interaktivne komunikacije sa korisnikom.

2.6.1. Aplikativni model Veba

Podrška aplikativnom modelu Veba u programskom jeziku WAFL počiva na pojmovima *servisa*, *sesije*, *stranice*, *posete*, *formulara* i *dijaloga*:

- Čitava Veb aplikacija napisana na programskom jeziku WAFL predstavlja jedan *servis*. Servis se sastoji od većeg broja programa koji služe za ostvarivanje dijaloga i automatskog generisanja stranica.

[16] Podrška za rad sa klasama još nije implementirana.

- Svaki dokument generisan nekim programom koji pripada WAFL servisu naziva se *stranica*. Stranice se nakon generisanja prosleđuju korisniku i aplikacija prelazi u stanje čekanja na sledeći zahtev korisnika.
- Svaki par (*zahtev*,*odgovor*), koji čine *zahtev* koji korisnik upućuje Veb lokaciji i *odgovor* koji od nje dobija, predstavlja jednu *posetu* Veb lokacije od strane korisnika.
- Skup svih poseta Veb lokacije od strane jednog korisnika u nekom vremenskom intervalu predstavlja jednu *sesiju*. Sesija na određen način predstavlja jednu instancu servisa koja ostvaruje razmenu informacija sa jednim korisnikom. Sesija ne predstavlja obavezno uređen niz poseta, jer korisnik može istovremeno koristiti servis u više prozora, čime se struktura sesije prilično komplikuje.
- *Dijalog* je programski upravljan deo sesije koji se sastoji od uređenog niza poseta. Dijalozi se upotrebljavaju u situacijama kada je radi uspešnog ispunjavanja nekog zahteva korisnika neophodno stupiti ponovo u interakciju sa njim (radi obezbeđivanja još nekih podataka ili izmene datih podataka i sl.) i to u garantovanom redosledu komunikacije.
- U programskom jeziku WAFL *formularom* se nazivaju podaci koje korisnik pošalje uz zahtev kao sadržaj nekog HTML formulara.

2.6.2. Parametri okruženja

Svaki program jednog WAFL servisa može pristupati parametrima koji opisuju stanje ili tok sesije, a koji su podeljeni u tri skupa parametara: *paremetri servisa*, *parametri sesije* i *parametri formulara*. Vrednosti svih parametara su tipa `string`. Svaki od skupova parametara ima tip `Map[string][string]` (katalog kome su i indeksi i vrednosti tipa `string`).

Parametri servisa

Parametri servisa su zajednički za sve sesije i sve programe koji se izvršavaju u okvirima jednog servisa. Svi parametri servisa su konstantni i ne mogu se programski menjati.

Za pristupanje parametrima servisa na raspolaganju su dve funkcije standardne biblioteke:

Funkcija	Vrednost
`Service()`	katalog svih parametara servisa
`ServiceValue(x)`	vrednost parametra servisa čiji je naziv jednak niski `x`

Tabela 21: Standardne funkcije za pristupanje vrednostima parametara servisa

Parametre servisa i njihove vrednosti definiše administrator servisa. Neophodno je da pristup definicijama parametara ima samo administrator, jer se njihovim menjanjem može značajno uticati na rad servisa. Čak i sam uvid u definicije parametara može biti problematičan, jer se među njima, između ostalog, nalaze i korisničko ime i lozinka za pristupanje bazi podataka.

Opis predloženog načina za definisanje parametara servisa, kao i spisak obaveznih parametara kojima se definiše servis, mogu se pronaći u dodatku *Parametri servisa* na strani 91[17].

[17] Aktuelna implementacija ne omogućava pristup svim parametrima servisa. U bližoj budućnosti će biti implementirano u skladu sa definicijom programskog jezika.

Parametri sesije

Parametri sesije su zajednički za sve programe koji se izvršavaju u okviru jedne sesije jednog servisa. Parametri sesije se mogu programski menjati, ali tako da su pri tome programi međusobno potpuno izolovani. Svakom programu se garantuje da će od početka do kraja izračunavanja imati na raspolaganju iste vrednosti parametara sesije, osim ako ih sam ne promeni.

Za pristupanje parametrima sesije na raspolaganju su dve funkcije standardne biblioteke:

Funkcija	Vrednost
`Session()`	katalog svih parametara sesije
`SessionValue(x)`	vrednost parametra sesije čiji je naziv jednak niski `x`

Tabela 22: Standardne funkcije za pristupanje vrednostima parametara sesije

Menjanje parametara sesije je moguće samo u okviru transakcionih (i akcionih) funkcija. Transakcione funkcije su opisane u odeljku odeljku *2.7.4 Transakcije* na strani 66.

Parametri formulara

Parametri formulara su lokalnog karaktera. Svakom izračunavanju programa odgovara tačno jedan skup parametara formulara. Oni su konstantni, pa se ne mogu programski menjati.

Za pristupanje parametrima formulara na raspolaganju su dve funkcije standardne biblioteke:

Funkcija	Vrednost
`Form()`	katalog svih parametara formulara
`FormValue(x)`	vrednost parametra formulara čiji je naziv jednak niski `x`

Tabela 23: Standardne funkcije za pristupanje vrednostima parametara formulara

Parametri formulara odgovaraju sadržaju formulara koji je korisnik popunio prilikom formiranja zahteva. Takođe, umesto putem HTML formulara, parametri formulara se mogu kodirati neposredno u okviru dela URL-a koji predstavlja upit.

Funkcije za pristupanje delovima HTTP zahteva

Često je radi automatskog formiranja URL-a nekog resursa (ili drugog programa) neophodno da program ima pristup pojedinim elementima HTTP zahteva. Kako se podrazumeva da je protokol HTTP i da je upit pročitan i pohranjen u skupu parametara formulara, preostalo je da se obezbedi pristup nazivu domena, putanji do programa i eventualno produženoj putanji. To se ostvaruje putem naredne tri standardne funkcije čije su vrednosti tipa `string`:

Funkcija	Vrednost
`httpHost()`	naziv domena (hosta) kome je HTTP zahtev upućen
`httpScript()`	naziv programa (sa putanjom)
`httpPathInfo()`	produžena putanja HTTP zahteva

Tabela 24: Funkcije za pristupanje delovima HTTP zahteva

Ostale funkcije za rad sa skupovima parametara

Standardna biblioteka obuhvata još nekoliko funkcija za rad sa skupovima parametara: `UpdateValueSet`, `JoinValueSets` i `EncodeUrlQuery`:

- Iako su skupovi parametara u osnovi nepromenljivi (ako za trenutak zanemarimo transakcione funkcije), ponekad je neophodno da se izračuna skup parametara koji bi se dobio promenom (ili dodavanjem) nekog parametra u poznatom skupu parametara. To omogućava funkcija `UpdateValueSet`;
- Funkcija `JoinValueSets` ima za dva argumenta skupove parametara. Rezultat je skup parametara koji predstavlja uniju skupova argumenata. Ako se neki parametar pojavljuje u oba skupa parametara, u rezultatu će se pojaviti sa onom vrednošću koju ima u prvom skupu;
- Funkcija `EncodeUrlQuery` formira nisku koja predstavlja upit kodiran u skladu sa definicijom upitnog dela URL-a. Argument je katalog parametara, a vrednost je niska.

Funkcija	Vrednost i tip
`UpdateValueSet(s,n,v)`	Katalog parametara koji se sastoji od kataloga `s` dopunjenog parametrom sa imenom `n` i vrednošću `v`; ako u katalogu `s` postoji parametar sa imenom `n`, njegova vrednost u rezultujućem katalogu će biti `v`. Tip je: `Map[string][string] * string * string -> Map[string][string]`
`JoinValueSets(s1,s2)`	Katalog parametara koji sadrži sve parametre iz kataloga `s1` i `s2`, pri čemu parametri koji se pojavljuju i u `s1` i u `s2` imaju vrednost kao u `s1`. Tip je: `Map[string][string] * Map[string][string] -> Map[string][string]`
`EncodeUrlQuery(s)`	Formira nisku u formatu URL upita na osnovu datog skupa parametra `s`. Tip je: `Map[string][string] -> string`

Tabela 25: Funkcije za rad sa skupovima parametara

2.6.3. Tekstualni šabloni

Tekstualan šablon je poseban način zapisivanja programa, funkcije ili dela izraza čija je vrednost tipa `string`. Tekstualni šabloni su predviđeni za jednostavnije definisanje programa, funkcija ili delova izraza koji se sastoje od konstantnog teksta sa manjim brojem nekonstantnih segmenata.

Tekstualan šablon se može pojaviti na svakom mestu gde i uobičajeni izrazi tipa `string`. Za definisanje funkcije pomoću tekstualnog šablona upotrebljava se uobičajena definicija, pri čemu telo definicije predstavlja zapis tekstualnog šablona. Tekstualan šablon se zapisuje u formi:

```
[ html | xml ] template <zapis_šablona> <#>
```

Oznaka `<#>` je oznaka za kraj šablona. U okviru zapisa tekstualnog šablona mogu se upotrebljavati *oznake za umetanje izraza* u okviru kojih se mogu pojaviti proizvoljni izrazi čija je vrednost tipa `string`. Oznake za umetanje izraza imaju sledeći oblik:

```
<# <izraz> #>
```

U narednom primeru funkcija `pozdrav` formira tekst pozdrava korisniku čije ime je jedini argument funkcije. Telo funkcije predstavlja tekstualni šablon:

```
pozdrav(ime) = html template
    Zdravo <i><# ime #></i>!
    <br>Kako ti se dopada WAFL?
    <#>;
```

Tekstualan šablon je ekvivalentan izrazu višestrukog sabiranja (dopisivanja) niski, pri čemu se kao sabirci naizmenično pojavljuju konstantne niske koje predstavljaju delove šablona i izrazi koji su umetnuti u šablon. Tako je naredni primer ekvivalentan prethodnom:

```
pozdrav(ime) =
    "Zdravo <i>" + ( ime )  + "</i>!"
    "<br>Kako ti se dopada WAFL?";
```

Rezultat izračunavanja šablona je niska koja se ne smatra šablonom, pa će svako pojavljivanje oznaka za umetanje izraza u samom rezultatu biti ignorisano.

Sintaksa tekstualnih šablona i oznaka za umetanje izraza prilagođena je upotrebi u dokumentima zapisanim na HTML-u i XML-u. Tekstualni šabloni čija je vrednost HTML dokument ili deo dokumenta zapisanog na HTML-u nazivaju se i *HTML šablonima*. Kako su tekstualni šabloni namenjeni pre svega za definisanje HTML šablona, u daljem tekstu se najčešće ne pravi razlika između ova dva naziva. Opcione ključne reči `html` i `xml` ispred obavezne ključne reči `template` za sada imaju samo dokumentacionu namenu. Zbog toga što je moguće da se u nekoj reviziji programskog jezika WAFL podrži još neki tip šablona dodavanjem novih opcionih oznaka ispred ključne reči `template`, ili da se uvedu neke razlike među tipovima šablona, preporučuje se da se oznaka tipa šablona uvek navodi.

Definisanje funkcija pomoću HTML šablona se može izvoditi na dva načina. Prvi način je navođenjem šablona kao tela definicije funkcije, kao u prethodno navedenom primeru. Drugi način definisanja funkcija pomoću šablona je njihovo izdvajanje u posebne datoteke, sa podrazumevanim nastavkom `htpl`. Takav pristup predstavlja posebnu vrstu funkcijskih datoteka koje se nazivaju *datoteke šablona*.

Sadržaj datoteke šablona ima sledeći oblik:

```
[ html | xml ] template [<naziv>] ([<ime>{,<ime>}])
<zapis_šablona>
```

Neophodno je da se u prvom redu datoteke šablona nalazi deklaracija parametara. Na kraju datoteke se ne zapisuje oznaka za kraj šablona. Naziv ima samo dokumentacionu primenu. Šabloni definisani u datotekama šablona se upotrebljavaju na potpuno isti način kao i funkcije definisane u funkcijskim datotekama. U šablonu se ne smeju upotrebljavati imena koja nisu javna ili deklarisana u listi parametara.

Ako bi se, na primer, funkcija iz prethodnih primera definisala u datoteci šablona, njen sadržaj bi mogao biti sledeći:

```
html template pozdrav(ime)

Zdravo <i><# ime #></i>!
```

```
<br>Kako ti se dopada WAFL?
```

Funkcija definisana u datoteci šablona upotrebljava se na potpuno isti način kao funkcija definisana u funkcijskom modulu. Tako se prethodni šablon u programima može upotrebljavati deklarisanjem na sledeći način:

```
pozdrav(ime) = function file 'pozdrav.whtpl';
```

Osnovna namena tekstualnih šablona, i posebno datoteka šablona, jeste razdvajanje programerskog od dizajnerskog dela posla pri definisanju izgleda dokumenata. Zbog toga se toplo preporučuje da se u fazi projektovanja aplikacije sve funkcije koje se definišu pomoću šablona oblikuju tako da imaju dovoljno argumenata da se u okviru oznaka za umetanje izraza ne moraju upotrebljavati složeni izrazi, već samo elementarni izrazi u kojima se pojavljuju formalni argumenti funkcije koja je definisana šablonom. Tek uz striktno poštovanje navedene preporuke može se precizno postaviti granica između programerskog i dizajnerskog dela posla. Osnovni razlog zbog koga je omogućena upotreba proizvoljno složenih izraza je pružanje slobodnog izbora programeru i razvojnom timu. Upotreba složenijih izraza može biti opravdana ako se kao argumenti prenose kolekcije objekata (lista, niz, katalog ili neka klasa) ili funkcije.

Iako je sintaksa tekstualnih šablona definisana tako da bude prilagođena prvenstveno HTML i XML dokumentima, ona jednako dobro odgovara i drugim tipovima dokumenata koji počivaju na tekstualnom zapisu, kao što su SGML, TeX, RTF i drugi.

Biblioteke tekstualnih šablona

Biblioteka tekstualnih šablona je biblioteka funkcija koja je posebna samo po tome što su funkcije definisane u obliku umetnutih tekstualnih šablona. Podrazumevani nastavak za datoteku koja predstavlja biblioteku umetnutih HTML šablona je `whtlib`. Takva biblioteka se upotrebljava na potpuno isti način kao i bilo koja druga biblioteka.

Primer biblioteke umetnutih šablona je modul `studenti.htlib`:

```
library studentiTpl
{
stranica(x) = template <html>
    <head><title>Studenti</title></head>
    <body> Tabela studenata <p>
    <table><# x #></table>
    </body></html><#>;

zaglavljeTabele = template <tr>
    <td>Indeks</td><td>God.</td>
    <td>Ime</td><td>Prezime</td><td>Predmeti</td>
    </tr><#>;

redTabeleStudenata(red) = template <tr>
    <td><# red["INDEKS"] #></td>
    <td><# red["GOD"] #></td>
    <td><# red["IME"] #></td>
    <td><# red["PREZIME"] #></td>
    <td><# red["PREDMETI"] #></td>
    <tr><#>;
};
```

Ekvivalentna biblioteka se može definisati i na osnovu posebnih datoteka sa pojedinačnim šablonima, kao u narednom primeru:

```
library studentiTpl
{
stranica = function file 'stranica.whtpl';
zaglavljeTabele = function file 'zaglavlje.whtpl';
redTabeleStudenata = function file 'redtabele.whtpl';
};
```

Kao što je već napomenuto, takva biblioteka se upotrebljava na potpuno isti način kao i bilo koja druga biblioteka. Na primer[18]:

```
//  program generiše izveštaj o predmetima položenim od strane
//  studenata koji su upisali fakultet date godine
opis::stranica(
    opis::zaglavljeTabele +
    studenti(FormValue("godina"))
        .map(redSaPredmetima)
        .formatRows( opis::redTabeleStudenata )
    )
where {
    opis = library 'studenti.whtlib';

    //  funkcija dodaje redu atribut "PREDMETI"
    redSaPredmetima( red ) = red.addEl("PREDMETI", pred(red))
    where {
        pred(red) = predmeti(
                        red["INDEKS"].asInt(),
                        red["GOD"].asInt()
                        ).formatRowsSep(predRed,", ");
        predRed(red) = red["NAZIVPRED"];
        };

    studenti( godina ) = query {
            select indeks, god, ime, prezime
            from student
            where god = :godina
            };

    predmeti( indeks, god ) = query {
            select pl.naziv
            from student s, plans pl, polagao po
            where pl.sifpred = pl.sifpred
              and pl.statg = s.statg
              and pl.sifprof = s.sifprof
              and po.indeks = s.indeks
              and po.god = s.god
              and po.ocena > 5
              and s.indeks = :indeks
              and s.god = :god
            };
};
```

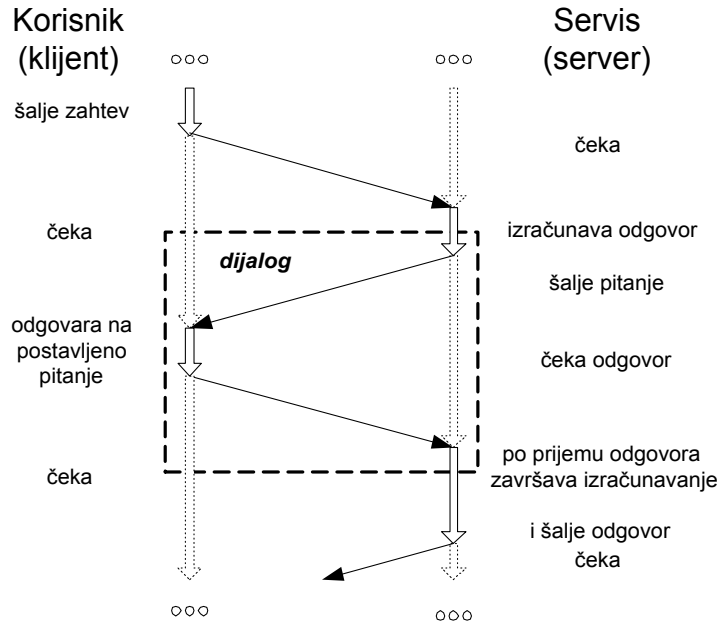
Preporučuje se da se svi šabloni potrebni za generisanje jedne stranice, ili grupe srodnih stranica, ili za definisanje nekih srodnih elemenata korisničkog interfejsa, pišu u vidu biblioteke tekstualnih šablona. Intenzivna primena biblioteka šablona radi grupisanja srodnih šablona, doprinosi modularnosti i uspešnijem razdvajanju dizajna korisničkog interfejsa od poslovne logike servisa.

[18] U ovom primeru se pojavljuje upotreba baze podataka, što je opisano tek nešto kasnije, u odeljku 2.7 *Baze podataka* na strani 56.

2.6.4. Dijalozi

Dijalog predstavlja programski upravljanu komunikaciju sa korisnikom. Za razliku od uobičajene komunikacije korisnika Veba i Veb servisa, kojom upravlja korisnik birajući željene resurse, dijalogom upravlja servis zahtevajući određene podatke od korisnika da bi se moglo nastaviti sa izračunavanjem osnovnog zahteva.

Dijalozi u programskom jeziku WAFL imaju za cilj da se kroz komunikaciju sa korisnikom obezbede dodatni podaci koji su neophodni za izračunavanje koje je u toku. Dok uobičajena forma komunikacije putem protokola HTTP podrazumeva da korisnik postavlja zahteve a servis isporučuje odgovore na te zahteve, u slučaju dijaloga je servis taj koji postavlja korisniku zahtev da saopšti određene podatke a korisnik na zahtev odgovara isporučivanjem tih podataka[19].



Slika 1: Shema interakcije između korisnika i servisa tokom dijaloga

Osnovni elementi dijaloga su *pitanje* i *odgovor*. Pitanje je niska (podatak tipa `string`) sa pitanjem koje se prosleđuje korisniku. Odgovor stiže u obliku skupa imenovanih niski, tj. kataloga tipa `Map[string][string]`. U slučaju Veb servisa pitanje je HTML stranica koja sadrži formular koji korisnik mora da popuni i pošalje i/ili veći broj hipertekstualnih veza sa vrednostima parametara kodiranim u odgovarajućim URL-ovima. Odgovor je skup vrednosti upisanih u formular ili predefinisanih u okviru URL-a odabrane veze.

Svaki dijalog je obuhvaćen izračunavanjem sistemske funkcije `ask`. Argument funkcije `ask` je pitanje (tipa `string`), a rezultat je odgovor na postavljeno pitanje (tipa `Map[string][string]`). Tokom izračunavanja funkcije `ask` izračunavanje programa se privremeno obustavlja zbog čekanja

[19] Primer situacije u kojoj je potrebno ostvariti dijalog sa korisnikom je, recimo, naručivanje nekog artikla posredstvom Veb aplikacije, a da korisnik prethodno nije identifikovan. Tada u okviru izračunavanja programa koji izvodi naručivanje, najpre od korisnika mora biti zahtevano da se identifikuje, a da se time ne prekine izračunavanje programa. Nakon identifikovanja korisnika nastavlja se dalje postupak naručivanja i formiranje stranice kojom se korisnik obaveštava o tome da li je naručivanje uspešno izvedeno. Drugačija implementacija po pravilu ima za posledicu deljenje poslova na veći broj manjih skriptova i prilično komplikuje kako razvoj tako i održavanje aplikacije.

na odgovor korisnika. Ako odgovor ne stigne u određenom vremenu (definisano na nivou implementacije ili parametara servisa), izračunavanje programa se prekida.

Dijalozi u Veb aplikacijama

U slučaju Veb aplikacije neophodno je raspoznavanje da li je korisnik isporučio odgovor na pitanje ili je postavio neki novi, potpuno nezavisan zahtev. Takođe, kada se prepozna da je u pitanju odgovor, neophodno je jednoznačno prepoznati sesiju i dijalog u okviru kojih je postavljeno odgovarajuće pitanje. To se postiže tako što se u okviru formulara kojim se postavlja pitanje kao akcija koja obrađuje sadržaj tog formulara navodi URL koji jednoznačno informiše servis o tome da od korisnika stiže odgovor na postavljeno pitanje, kao i koji korisnik, sesija i dijalog su u pitanju. Zbog toga je potrebno da se akcija formulara određuje automatski, pomoću funkcije `answerAction()`.

Funkcija `answerAction` nema argumenata i izračunava nisku koja predstavlja generički "naziv skripta" koji obezbeđuje nastavak dijaloga. Implementacija okruženja u kome radi WAFL servis mora taj "naziv skripta" prepoznati kao kodiranu oznaku nastavka dijaloga. Pri razvoju podsistema dijaloga neophodno je imati na umu da više poziva funkcije `answerAction` pri formiranju jednog pitanja može dati različite rezultate, ali da svi ti rezulati moraju ispravno identifikovati tekući dijalog. Moguća forma[20] niske koju generiše funkcija `answerAction` je:

```
answer.<kod>.wafl
```

gde je `<kod>` niska koja jednoznačno identifikuje dijalog o čijem se nastavku radi i u skladu je sa pravilima zapisivanja naziva skripta (tj. delova URL-a). Primer rezultata funkcije `answerAction` je:

```
answer.26260952.1024615.70796383.wafl
```

Program u narednom primeru postavlja korisniku pitanje kojim se od njega traži da upiše svoje ime. Nakon što korisnik upiše svoje ime i prosledi odgovor servisu, nastavlja se izvršavanje ovog istog programa tako što funkcija `ask` vraća taj odgovor kao svoj rezultat i on se upotrebljava u daljem izračunavanju:

```
pitanje()
    .ask()
    .pozdrav()

where {
    pitanje() = html template
        <html><body>
        Dobro došli na naš servis.
        <form action="<# answerAction() #>" method="post">
        Molimo upišite svoje ime:
        <input type="text" name="ime">
        <input type="submit" value="OK">
        </form></body></html>
        <#>;

    pozdrav(odgovor) = html template
```

[20] Nema potrebe da se u okviru definicije programskog jezika uvede obavezan oblik generičkog naziva skripta koji odgovara nastavku dijaloga. Navodi se samo jedan moguć oblik kao sugestija i radi približavanja problema čitaocu.

```
        <html><body>
        Umesto samo "Zdravo, svete!",
        sada možemo da kažemo
        <b>Zdravo korisniče <# odgovor["ime"] #>!</b>
        </body></html>
        <#>;
    }
```

Umesto putem formulara, pitanja se mogu postavljati i kao izbor jedne od ponuđenih hipertekstualnih veza, pri čemu se u vezama osim koda akcije mora pojaviti i neki parametar koji identifikuje odgovor. Naredni program nudi korisniku izbor poslastice i najavljuje mu šta će dobiti:

```
pitanje()
    .ask()
    .najava()

where {
    pitanje() = html template
        <html><body>
        Izaberite jedan od ponuđenih specijaliteta:<br>
        <a href="<# answerAction() #>?spec=1"> Krofne </a><br>
        <a href="<# answerAction() #>?spec=2"> Palačinke </a><br>
        <a href="<# answerAction() #>?spec=3"> Sladoled </a><br>
        <a href="<# answerAction() #>?spec=4"> Torta </a><br>
        </body></html>
        <#>;

    najava(odgovor) = html template
        <html><body>
        Po vašem izboru, <#
            if odgovor['spec']=='1'
                then 'stižu krofne sa džemom'
            else if odgovor['spec']=='2'
                then 'stižu palačinke sa čokoladom'
            else if odgovor['spec']=='3'
                then 'stiže sladoled od lešnika'
            else 'stiže VEEEELIKA torta'
        #>!
        </body></html>
        <#>;
    }
```

Ako je pri nekom izračunavanju potrebno od korisnika tražiti da popuni nekoliko formulara zaredom, za objedinjavanje odgovora se može upotrebiti funkcija `JoinValueSets`, opisane u odeljku *2.6.2 Parametri okruženja* na strani 45.

Ograničenja

Primetimo da postoji mogućnost da korisnik ne prati komunikaciju na zamišljen način:

- korisnik može odustati od postavljenog zahteva i nastaviti upotrebu servisa postavljanjem zahteva koji nemaju direktne veze sa iniciranim dijalogom, ili čak u potpunosti prekinuti upotrebu servisa;
- korisnik može privremeno obustaviti nastavak komunikacije u okviru dijaloga da bi prisupio nekim drugim resursima, bilo putem istog ili drugog servisa, a zatim nastaviti dijalog;
- korisnik se nakon odgovaranja na pitanje može vratiti na prethodnu stranicu, tj. na pitanje, u želji da da drugi odgovor.

Da bi dijalog i servis u okviru koga se dijalog formira radili ispravno, uvode se određena ograničenja na nivou korisnika i implementacije i preporuke na nivou razvoja WAFL aplikacije.

- Korisnik koji je postavljanjem nekog zahteva inicirao dijalog:
  - ne sme da se vraća na pitanje na koje je već odgovorio, a da pri tome ne ponovi kompletan zahtev tokom čijeg je izračunavanja došlo do uspostavljanja dijaloga (problem nastaje ako program za pregledanje Veba korisniku predstavi stranicu sa pitanjem pročitanu iz prethodno lokalno pohranjene kopije, umesto da je ponovo zahteva od servera, jer se na svako pitanje može dati tačno jedan odgovor i ponovno odgovaranje proizvodi grešku);
  - sme tokom jednog dijaloga postavljati druge zahteve servisu, tj. posećivati druge stranice formirane istim servisom, ali ne sme u istom servisu započeti nov dijalog; u slučaju započinjanja novog dijaloga ranije započet dijalog se mora automatski prekinuti, ili se korisniku mora ponuditi da najpre nastavi ili prekine dijalog koji je u toku.
- Implementacija mora obezbediti:
  - poštovanje ograničenja na nivou korisnika;
  - da započet dijalog može da se nastavi i nakon privremenog prekida;
  - da započet dijalog koji se ne nastavi nakon određenog vremena bude automatski prekinut radi oslobađanja rezervisanih resursa.
- Pri razvoju aplikacije preporučuje se da započet dijalog ne drži zaključane deljene podatke tokom čekanja na odgovor korisnika. Da bi to bilo ispoštovano dovoljno je da se interakcija sa korisnikom ne inicira tokom izvođenja transakcija, tj. da se funkcija `ask` ni neposredno ni posredno ne upotrebljava u definicijama transakcija.

Dijalozi u drugim okruženjima

Ako WAFL program radi u nekom drugom okruženju, a ne kao Veb aplikacija, tada priča o pitanjima i odgovorima ima nešto drugačiju prirodu. Implementacija je u drugim okruženjima često nešto jednostavnija, jer korisnik obično u okviru tekuće instance interfejsa kojim mu se postavlja pitanje ne može uraditi ništa drugo osim da odgovori na postavljeno pitanje ili da prekine rad.

U slučaju nekog grafičkog radnog okruženja, koje nije Veb, na implementaciji je da korisniku predstavi formular na odgovarajući način, kao i da prihvati upisani odgovor. Ukoliko se radi o nekom komandnom radnom okruženju, recimo komandnom interfejsu operativnog sistema, na implementaciji je da obezbedi odgovarajući tekstualni prikaz tog formulara, ili da umesto jednog formulara korisniku postavlja niz pitanja, za svako polje formulara po jedno.

U slučaju okruženja koje, za razliku od Veb aplikacija, ne podrazumeva fizičku i vremensku distancu između korisnika i WAFL servisa, može biti od koristi da se umesto popunjavanja formulara od korisnika zahteva da odgovara na jedno po jedno pitanje. To se postiže funkcijom `input`, čiji je argument niska koja predstavlja tekst pitanja, a rezultat niska koja predstavlja odgovor na postavljeno pitanje. Tako se, na primer, u narednom primeru definiše funkcija koja izračunava srećan broj aktuelnog korisnika:

```
srećanBroj() = input("Upišite svoj srećan broj").asInt();
```

2.6.5. *MIME resursi*[21]

U opštem slučaju ne postoji nikakva pretpostavka u odnosu na tip rezultata programa. Međutim, u slučaju Veb aplikacija i dinamičkih Veb lokacija situacija je nešto drugačija. U najvećem broju slučajeva rezultat će predstavljati HTML dokument. U preostalim slučajevima uglavnom će se raditi o nekom drugom tipu dokumenta uobičajenom na Vebu, tj. o dokumentu kojim će imati neki drugi MIME tip. Zbog toga ćemo u slučaju Veb aplikacija podrazumevati da je tip rezultata programa `MimeResource`.

Ukoliko rezultat programa nije objekat klase `MimeResource` postupa se na sledeći način:

1. Ako rezultat programa nije ni primitivnog tipa ni objekat klase `MimeResource`, prijavljuje se greška. Posebno, ako implementacija obezbedi mehanizme za konvertovanje neprimitivnih tipova (liste, katalozi, nizovi ili klase) u niske, tada ne mora da se prijavljuje greška, već se rezultat može konvertovati u nisku (tipa `string`), nakon čega podleže narednim koracima.
2. Ako je rezultat programa vrednost nekog primitivnog tipa osim `string`, rezultat se konvertuje u tip `string` i zatim podleže narednim koracima.
3. Ako je podrazumevani MIME tip "`text/html`", a rezultat programa niska tipa `string`, koja ne predstavlja kompletan HTML dokument (tj. nije ograđena oznakom `<HTML>`), tada se ta niska kompletira do HTML dokumenta primenom funkcije definisane u funkcijskoj datoteci određenoj parametrom servisa `defaultWrapper`, pa dobijeni rezultat podleže narednom koraku obrade. Ako nije definisan parametar servisa `defaultWrapper`, ili je na početku programa navedena direktiva `#nowrap`, rezultat se ne dopunjava ali podleže narednom koraku.
4. Ako je rezultat programa vrednost tipa `string`, tada se od te vrednosti automatski formira objekat klase `MimeResource` podrazumevanog MIME tipa i taj objekat se vraća kao rezultat izračunavanja programa. Podrazumevani MIME tip se definiše vrednošću parametra servisa `defaultMimeType`.

Nakon primene ovih koraka ili je dobijen objekat klase `MimeResource` koji predstavlja rezultat, ili je u međuvremenu došlo do greške.

Klasa `MimeResource` definiše na sledeći način:

```
class MimeResource
members {
    type;
    content;
    }
methods {
    // konstruktor koji izračunava novi MIME resurs
    // datog tipa i sadržaja
```

[21] Implementacija je izvedena bez upotrebe klasa. Postoje funkcije koje omogućuju praktično istovetan način upotrebe. Postoje odgovarajuće funkcije `mimeResource` (tipa `String*String -> MimeResource`), `mimeType` (tipa `MimeResource -> String`) i `mimeContent` (tipa `MimeResource -> String`).

```
// primetimo posredni zahtev da oba člana budu tipa string
static MimeResource(type,res) = constructor(type+'',res+'');
// metod koji izračunava tip MIME resursa
Type() = type;
// metod koji izračunava sadržaj MIME resursa
Content() = content;
};
```

2.7. Baze podataka

U analizi karakteristika problema za čije se rešavanje projektuje programski jezik WAFL, naglašen je značaj upotrebe baza podataka u programima i potreba obezbeđivanja elemenata jezika koji omogućavaju što prirodniju i jednostavniju komunikaciju sa sistemima za upravljanje bazama podataka.

U programskom jeziku WAFL upotreba sistema za upravljanje bazama podataka je podeljena na tri osnovna nivoa:

- uspostavljanje komunikacije sa sistemom za upravljanje bazama podataka;
- postavljanje upita;
- izvođenje transakcija.

Uspostavljanjem komunikacije sa sistemom za upravljanje bazama podataka upravlja administrator servisa definisanjem odgovarajućih parametara servisa. Postavljanjem upita i izvođenjem transakcija upravlja programer na jeziku sistema za upravljanje bazama podataka, a pomoću odgovarajućih elemenata programskog jezika WAFL. Pri tome se o ostvarivanju transakcionih mehanizama stara implementacija.

2.7.1. Komunikacija sa sistemom za upravljanje bazama podataka

Komunikacija servisa pisanog na programskom jeziku WAFL sa sistemom za upravljanje bazama podataka počiva na sledećim osnovnim principima:

- uspostavljanje komunikacije servisa sa sistemom za upravljanje bazama podataka odvija se automatski, na osnovu vrednosti odgovarajućih parametara servisa;
- servis može uspostavljati komunikaciju (konekcije) samo sa jednom bazom podataka;
- servis može (automatski i prema potrebi) uspostaviti više konekcija sa bazom podataka;
- servis može čuvati trenutno neupotrebljavane konekcije (da se u slučaju ponovne potrebe za njima ne bi čekalo na njihovo uspostavljanje) i automatski ih zatvarati kada njihov broj pređe granicu definisanu odgovarajućim parametrom servisa;
- svi upiti se, inicijalno, postavljaju sa nivoom izolovanosti *Read Uncommited* [22], ali se nivo izolovanosti pojedinačnog upita može po potrebi promeniti;

[22] Nivoi izolovanosti su predstavljeni u odeljku ***Error! Reference source not found. Error! Reference source not found.***, na strani **Error! Bookmark not defined.**.

- upiti sa istim nivoom izolovanosti, koji može biti *Read Commited* ili *Read Uncommited*, a koji se postavljaju iz različitih programa (ili različitih instanci istog programa), mogu deliti konekciju, čime se smanjuje opterećenje sistema za upravljanje bazama podataka;
- upiti sa nivoom izolovanosti *Serializable* ili *Repeatable Read* ne mogu deliti konekciju ni među različitim instancama istog programa, jer bi se time narušila njihova izolovanost;
- sve transakcije se, inicijalno, izvode sa nivoom izolovanosti *Read Commited*, ali se nivo izolovanosti može po potrebi promeniti;
- za svaku transakciju se obezbeđuje posebna konekcija sa odgovarajućim nivoom izolovanosti;
- upiti koji se postavljaju tokom izvođenja transakcije postavljaju se u okviru transakcije i sa nivoom izolovanosti transakcije.

Parametri servisa koji se tiču rada sa bazama podataka

Pri zadavanju vrednosti parametara servisa, administrator servisa definiše i način rada sa bazom podataka. U narednoj tabeli su predstavljeni standardni parametri servisa koji se odnose na rad sa bazom podataka:

Parametar	Značenje
dbAlias	Jednoznačna identifikacija baze podataka koju upotrebljava servis. Niska koja identifkuje bazu podataka može zavisiti od implementacije i okruženja.
dbUser	Korisnički nalog pod kojim servis upotrebljava bazu podataka.
dbPassword	Lozinka uz korisnički nalog pod kojim servis upotrebljava bazu podataka.
dbKeepSessions	Najveći dopušten broj neaktivnih konekcija koje se čuvaju radi uštede na ponovnom uspostavljanju konekcija.
dbQueryIsolation	Podrazumevani nivo izolovanosti upita van transakcija. Dopuštene vrednosti su `s`, `rr`, `rc`, `ru`. Parametar nije obavezan. Podrazumevana vrednost je `ru`.
dbTransIsolation	Podrazumevani nivo izolovanosti transakcija. Dopuštene vrednosti su `s`, `rr`, `rc`, `ru`. Parametar nije obavezan. Podrazumevana vrednost je `rr`.

Tabela 26: Standardni parametri servisa koji određuju način rada sa bazom podataka

Za razliku od ostalih parametara servisa, vrednost parametra `DbPassword` se ne može koristiti u programima. Iz bezbednosnih razloga rezultat izraza `ServiceValue( "DbPassword")` je uvek prazna niska. Pored navedenih parametara, implementacija prema konkretnim potrebama može uvesti i druge parametre koji preciznije određuju rad sa bazom podataka. (Videti *Dodatak D - Parametri servisa* na strani 91.)

2.7.2. Postavljanje upita

Upitna funkcija, ili *upit*, u programskom jeziku WAFL predstavlja posebnu funkciju čije izračunavanje se sastoji od izračunavanja upita nad bazom podataka i prevođenja tako dobijenog rezultata u formu koja odgovara funkcionalnom kontekstu programskog jezika WAFL.

Zaglavlje definicije upita je slično bilo kojoj drugoj definiciji, ali telo definicije ima specifičnu formu:

```
<definicija_upita> ::=
    <ime> [<formalni_parametri>] =
    [sql] query [ isolation ( s | rr | rc | ru ) ]
    { <upit_na_upitnom_jeziku> } ;
```

gde je `<upit_na_upitnom_jeziku>` zapis upita na upitnom jeziku sistema za upravljanje bazama podataka. Ako se izostavi opciona ključna reč `sql`, podrazumeva se da je upitni jezik SQL. Neophodno je da postoji podrška za upitni jezik SQL, dok konkretne implementacije mogu uvesti podršku za druge upitne jezike, a time i nove ključne reči kojima se ti jezici identifikuju. U daljem tekstu se podrazumeva upotreba upitnog jezika SQL pri postavljanju upita.

Nivo izolovanosti[23] upita se navodi opciono i može biti `s` (engl. *Serializable*), `rr` (engl. *Repeatable Read*), `rc` (engl. *Read Commited*) ili `ru` (engl. *Read Uncommited*). Ukoliko se nivo izolovanosti ne navede eksplicitno, podrazumeva se `ru` (engl. *Read Uncommited*), osim ako nije drugačije definisano parametrima servisa (*Dodatak D - Parametri servisa* na strani 91.). Ukoliko se upit izračunava u okviru transakcije, zanemaruje se nivo izolovanosti naveden za upit i upotrebljava se nivo izolovanosti koji važi za transakciju.

Ako upitna funkcija ima parametre, formalni parametri se mogu upotrebljavati u okviru zapisa upita u skladu sa pravilima upitnog jezika. U slučaju SQL-a formalni parametri se mogu pojavljivati u okviru upita kao host promenljive, pri čemu imenima formalnih parametara mora neposredno prethoditi znak ':'.

U narednom primeru se definiše upit koji izračunava podatke o studentima koji su upisali fakultet ne pre date godine:

```
studentiOd(godina) = query {
    select indeks, god, ime, prezime
    from student
    where god >= :godina
    order by god, indeks
    };
```

Rezultat upita predstavlja listu redova. Svaki red za sebe predstavlja katalog atributa čiji su indeksi i vrednosti tipa `string`. Tip rezultata upita možemo formalno zapisati kao `List[Map[string][string]]`. Kako je rezultat upita lista, nad njim se mogu izračunavati sve funkcije definisane za rad nad listama. Atributima jednog reda upita pristupa se na način uobičajen za kataloge.

[23] Nije implementirana podrška za izbor nivoa izolovanosti. Sve se odvija sa istim nivoom izolovanosti, onako kako je definisano u okviru podešavanja BDE aliasa.

Lenjost rezultata upita

Izračunavanjem upita se ne preuzimaju od SUBP svi redovi rezultata, već se pravi lenja lista redova, koja se izračunava prema potrebi. Tako se, u slučaju da rezultat upita čini 100 redova, a program koristi samo prvih 10, od SUBP ne traži 90 nepotrebnih redova. Time se obezbeđuje manje opterećenje SUBP, komunikacionih linija i sistema na kome se izračunava WAFL program.

U nekim situacijama lenjost liste koja predstavlja rezultat upita može napraviti probleme koji se razrešavaju primenom funkcije `forced` za izračunavanje lenje liste:

- Ako se u izračunavanju upotrebljava lenja lista koja predstavlja rezultat upita, tada se poslednji redovi tog rezultata neće preuzeti od SUBP-a sve do pred sam kraj izračunavanja. Ako je izračunavanje složeno, pa samim tim i dugotrajno, a želi se rasteretiti SUBP od držanja otvorenog kursora, tada se forsiranjem izračunavanja liste preuzimaju svi podaci od SUBP i kursor se zatvara, a zatim se izračunavanje nastavlja nad izračunatom listom.
- Druga moguća primena je u slučaju da se pri izračunavanju izvode transakcije koje utiču na rezultat upita čiji je rezultat lenja lista koja se upotrebljava u tom izračunavanju. Tada je radi ispravnog toka transakcije neophodno da se čitav rezultat upita preuzme od SUBP pre početka transakcije.

Dopunski atributi

Pored atributa koji čine rezultat upita, a čiji se nazivi po pravilu zapisuju velikim slovima, svakom redu se dodaje dopunski atribut `counter`, čija je vrednost redni broj reda u rezultatu upita. Implementacija može svaki red upita dopuniti još nekim atributima, pri čemu svi moraju imati nazive zapisane isključivo malim slovima.

2.7.3. Generisanje izveštaja

Za generisanje izveštaja na osnovu rezultata upita mogu se upotrebljavati već definisane funkcije za rad sa listama. Tako se u narednom primeru generiše izveštaj o studentima upotrebom već navedenog upita:

```
studentiOd(1995)
    .izveštaj()

where {
    //  f.ja izdvaja podatke o studentima upisanim od 1995.
    studentiOd(godina) = query {
        select indeks, god, ime, prezime
        from student
        where god >= :godina
        order by god, indeks
        };

    //  f.ja generiše izveštaj na osnovu rezultata upita
    izveštaj(rez) = html template
        <html><head><title>Studenti</title></head><body>
        <#
            if rez.empty() then "Nema studenata"
            else (
                // ovo je primer šablona u šablonu
                html template
```

```
                    Tabela studenata <p>
                    <table>
                    <tr>
                        <td><b>Indeks</b></td>
                        <td><b>God.</b></td>
                        <td><b>Ime</b></td>
                        <td><b>Prezime</b></td>
                    </tr>
                    <# rez.map(red).sums() #>
                    </table><#>
                )
        #>
        </body><html><#>;

    //  formira jedan red izveštaja na osnovu jednog reda upita
    red(r) = html template <tr>
        <td><# r["INDEKS"] #></td>
        <td><# r["GOD"] #></td>
        <td><# r["IME"] #></td>
        <td><# r["PREZIME"] #></td>
        </tr><#>;

    //  f.ja sumira listu niski
    sums(l) = if l.empty() then "" else l.hd() + l.tl().sums();
    }
```

Funkcije za generisanje izveštaja

Zbog česte potrebe za generisanjem ovakvih i sličnih izveštaja na osnovu rezultata upita ili liste drugih vrednosti, definiše se standardna biblioteka funkcija za formiranje izveštaja. Funkcije za formiranje izveštaja ne rade samo sa rezultatima upita nego i sa bilo kojim drugim listama:

Funkcija	**Opis**
`formatRows(l,red)`	Izračunava izveštaj koji se sastoji od pojedinačnih elemenata liste `l` formatiranih pomoću funkcije `red`. Isto kao: `l.map(red).aggregate(strCat,"")`.
`formatRowsSep(l,red,sep)`	Izračunava izveštaj koji se sastoji od pojedinačnih elemenata liste `l` formatiranih pomoću funkcije `red`, pri čemu su svaka dva elementa međusobno razdvojena separatorom `sep`.
`formatReport( l, zaglavlje,red,kraj, prazna )`	Izračunava izveštaj koji se sastoji od zaglavlja `zaglavlje`, pojedinačnih elemenata liste `l` formatiranih pomoću funkcije `red` i kraja izveštaja `kraj`. Ako je lista `l` prazna, rezultat je argument `prazna`.
`formatSubReport( l, zaglavlje,red,kraj, prazna, start,n )`	Izračunava izveštaj koji se sastoji od zaglavlja `zaglavlje`, `n` pojedinačnih elemenata liste `l`, uz izostavljanje prvih `start` elemenata, formatiranih pomoću funkcije `red` i kraja izveštaja `kraj`. Ako je lista `l` prazna, ili nema više od `start` elemenata, rezultat je argument `prazna`. Isto kao: `l.subList(start,n).formatReport(z,r,k,p)`

Tabela 27: Funkcije za generisanje izveštaja

Upotreba ovih funkcija prilično pojednostavljuje pisanje i održavanje programa za generisanje izveštaja. Naredni primer je ekvivalentan prethodnom, ali je pregledniji i jednostavniji za održavanje:

```
studentiOd(1995).izveštaj()

where {
    //  f.ja izdvaja podatke o studentima upisanim od 1995.
    studentiOd(godina) = query {
        select indeks, god, ime, prezime
        from student
        where god >= :godina
        order by god, indeks
        };

    //  f.ja generiše izveštaj na osnovu rezultata upita
    izveštaj(tab) = html template
        <html><head><title>Studenti</title></head><body>
        <#
            tab.formatReport( zaglavlje, red, "</table>",
                              "Nema studenata!" )
        #></body><html><#>;

    //  zaglavlje izveštaja
    zaglavlje = html template
        Tabela studenata <p>
        <table>
        <tr>
            <td><b>Indeks</b></td>
            <td><b>God.</b></td>
            <td><b>Ime</b></td>
            <td><b>Prezime</b></td>
        </tr><#>;

    //  formira jedan red izveštaja na osnovu jednog reda upita
    red(r) = html template <tr>
        <td><# r["INDEKS"] #></td>
        <td><# r["GOD"] #></td>
        <td><# r["IME"] #></td>
        <td><# r["PREZIME"] #></td>
        </tr><#>;
    }
```

Funkcije za generisanje izveštaja sa navigacijom

Ako se radi o velikim izveštajima, često je potrebno da se podele na više dokumenata sa po manje podataka. Opisana funkcija `formatSubReport` omogućava formiranje dela izveštaja, ali posao oko prelaženja sa jednog na drugi deo izveštaja i dalje je na programeru. Da bi se i taj deo posla pojednostavio definisana je *biblioteka funkcija za generisanje izveštaja sa navigacijom*.

Biblioteka počiva na sledećim konceptima i principima:

- sve delove izveštaja generiše isti skript, razlika je samo u parametrima koji određuju koji deo izveštaja je potrebno predstaviti;
- navigaciona linija predstavlja skup hipertekstualnih veza na isti skript, ali sa parametrima koji sugerišu da je potrebno predstaviti drugi deo izveštaja;
- na istom dokumentu (stranici) može biti više izveštaja čija se navigacija mora odvijati međusobno potpuno nezavisno; zbog toga svaki izveštaj mora imati svoj identifikacioni broj;
- programeru se prepušta da definiše mesto pojavljivanja, sadržaj i izgled navigacione linije, ali mu se nudi i jedan predefinisan izgled.

Navigaciona linija predstavlja skup veza koji omogućava izbor prethodnog, narednog ili nekog drugog dela izveštaja. Ima tri celine koje se mogu prikazivati po izboru:

- levu navigaciju, koja omogućava pregledanje prethodnog dela izveštaja;
- desnu navigaciju, koja omogućava pregledanje narednog dela izveštaja;
- indeksnu navigaciju, koja omogućava izbor dela izveštaja prema rednom broju.

Funkcija `navigationBar` omogućava formiranje navigacione linije za neki izveštaj na najopštiji način. Manje izbora, ali zbog toga i jednostavniju upotrebu, nude funkcije `defaultNavigationBar` i `simpleNavigationBar`.

Funkcija / Parametar	**Opis**
`navigationBar(`	Funkcija formira navigacionu liniju.
`id,`	Identifikator izveštaja na koji se odnosi navigaciona linija.
`l,`	Lista čiji se sadržaj predstavlja izveštajem.
`od,`	Redni broj prvog elementa liste u tekućem delu izveštaja.
`broj,`	Broj elemenata liste predstavljenih u delovima izveštaja.
`levaVeza,`	Izgled veze na prethodni deo izveštaja, tipa `string`. Ako je prazna niska, iz navigacione linije se izostavlja veza na prethodni deo izveštaja.
`desnaVeza,`	Izgled veze na naredni deo izveštaja, tipa `string`. Ako je prazna niska, iz navigacione linije se izostavlja veza na naredni deo izveštaja.
`indeksVezaFnc,`	Funkcija koja za celobrojni argument formira nisku koja predstavlja izgled veze na deo izveštaja koji odgovara datom rednom broju.
`maxIndeks` `)`	Najveći broj delova izveštaja koji može da se dobije posredstvom indeksne navigacije, tipa `int`. Ako ovaj broj nije veći od nule, indeksna navigacija se izostavlja.
`defaultNavigationBar(`	Funkcija formira navigacionu liniju sa sve tri celine. Podrazumevani izgled veze na prethodni deo izveštaja je '<', veze na naredni '>', a indeksna navigacija se sastoji od dekadnog zapisa rednih brojeva najviše 20 delova izveštaja.
`id,`	Identifikator izveštaja na koji se odnosi navigaciona linija.
`l,`	Lista čiji se sadržaj predstavlja izveštajem.
`od,`	Redni broj prvog elementa liste u tekućem delu izveštaja.
`broj` `)`	Broj elemenata liste predstavljenih u delovima izveštaja.
`simpleNavigationBar(`	Funkcija formira navigacionu liniju bez indeksne navigacije.
`id,`	Identifikator izveštaja na koji se odnosi navigaciona linija.
`l,`	Lista čiji se sadržaj predstavlja izveštajem.
`od,`	Redni broj prvog elementa liste u tekućem delu izveštaja.

Funkcija / Parametar	Opis
broj,	Broj elemenata liste predstavljenih u delovima izveštaja.
levoDugme,	Izgled veze na prethodni deo izveštaja, tipa `string`. Ako je prazna niska, iz navigacione linije se izostavlja veza na prethodni deo izveštaja.
desnoDugme)	Izgled veze na naredni deo izveštaja, tipa `string`. Ako je prazna niska, iz navigacione linije se izostavlja veza na naredni deo izveštaja.

Tabela 28: Funkcije za generisanje navigacione linije

Nazivi parametara koji su rezervisani za upotrebu u navigacionim linijama imaju formu: `startX` i `rowsX`, gde je `X` ceo broj koji predstavlja identifikator izveštaja. Ne preporučuje se njihova upotreba u druge svrhe.

Za formiranje izveštaja sa odgovarajućom navigacijom namenjene su naredne funkcije:

Funkcija / Parametar	Opis
`formatReportWithNavBar(`	Funkcija formira izveštaj sa navigacijom.
`id,`	Identifikator izveštaja na koji se odnosi navigaciona linija.
`l,`	Lista čiji se sadržaj predstavlja izveštajem.
`zaglavlje,`	Funkcija koja formira zaglavlje. Ima argumenate: `id`, `l`, `od` i `broj`, koje može upotrebljavati pri formiranju navigacione linije. Značenje argumenata je, redom: identifikacija izveštaja, lista od koje se formira izveštaj, broj preskočenih i broj predstavljenih redova.
`red,`	Funkcija koja formatira izveštaj za jedan red. Ima jedan argument koji odgovara elementu liste čiji se sadržaj predstavlja izveštajem.
`kraj,`	Funkcija koja formira kraj izveštaja. Ima iste argumente kao i funkcija `zaglavlje`.
`prazna,`	Niska koja predstavlja tekst izveštaja u slučaju da je lista `l` prazna.
`defOd,`	Broj elemenata liste preskočenih u delu izveštaja od kog se počinje.
`defBroj` `)`	Broj elemenata liste predstavljenih u delovima izveštaja.

`formatReportWithHeaderNavBar(`
`id, l,`
`zaglavlje, red, kraj,`
`prazna,`
`defOd, defBroj`
`)`
	Funkcija formira izveštaj sa navigacijom u zaglavlju. Svi argumenti imaju isto značenje kao kod funkcije `formatReportWithNavBar`, osim argumenta `kraj` čija vrednost je niska koja predstavlja kraj izveštaja.

`formatReportWithFooterNavBar(`
`id, l,`
`zaglavlje, red, kraj,`
`prazna,`
`defOd, defBroj`
`)`
	Funkcija formira izveštaj sa navigacijom u dnu izveštaja. Svi argumenti imaju isto značenje kao kod funkcije `formatReportWithNavBar`, osim argumenta `zaglavlje` čija vrednost je niska koja predstavlja početak izveštaja.

Tabela 29: Funkcije za formiranje izveštaja sa automatizovanom navigacijom

Naredni primer predstavlja kompletan program koji uz pomoć bibilioteke za generisanje izveštaja sa navigacijom formira stranicu koja ispod dela izveštaja ima navigacionu liniju koja omogućava prikazivanje nekog drugog dela izveštaja:

```
studentiOd(1995).izveštaj()

where {
    //  f.ja izdvaja podatke o studentima upisanim od 1995.
    studentiOd(godina) = sql query {
        select indeks, god, ime, prezime
        from student
        where god >= :godina
        order by god, indeks
        };

    //  f.ja generiše izveštaj na osnovu rezultata upita
    izveštaj(upit) = html template
        <html><head><title>Studenti</title></head><body>
        <#
            formatReportWithFooterNavBar(
            1, upit,
                pocetak, red, kraj, "",
                0, 10
                )
        #></body><html><#>;
    where   {
        //  zaglavlje izveštaja
        pocetak = html template
            Tabela studenata <p>
            <table>
            <tr>
                <td><b>Indeks</b></td>
                <td><b>God.</b></td>
                <td><b>Ime</b></td>
                <td><b>Prezime</b></td>
            </tr><#>;

        //  f.ja generiše jedan red izveštaja
        red(r) = html template <tr>
            <td><# r["INDEKS"] #></td>
            <td><# r["GOD"] #></td>
            <td><# r["IME"] #></td>
            <td><# r["PREZIME"] #></td>
            </tr><#>;

        //  kraj izveštaja sadrži navigacionu liniju
        kraj(repIndex,q,start,range) = html template
            <tr><td colspan=4>
                <#
                    defaultNavigationBar(
                            repIndex, q, start, range
                            )
                #>
            </td></tr></table>
            <#>;
        };
    };
```

Rad sa drugim tipovima podataka

POSTOJI OVDE NEOPISANA DOPUNA U VEZI SA BLOB TIPOVIMA PODATAKA.

Svi atributi se podrazumevano tumače kao da su tipa `string`. Ako se želi drugačije tumačenje, to je neophodno eksplicitno zahtevati upotrebom odgovarajuće funkcije za konverziju. Ukoliko je, na primer, potrebno upotrebljavati kao broj vrednost numeričkog atributa `GOD` iz prethodnih primera, takva konverzija se može izvesti na sledeći način:

```
red(r) = ... r["GOD"].asInt() ...
```

Podrška za vrednost `NULL`, koja se može pojaviti u bazi podataka, ostvarena je kroz podršku vrednosti `NULL` u okviru tipa `string`. Kao što je u odeljku posvećenim prostom tipu `string` (*2.2.1 Prosti tipovi podataka,* strana 13.) već predstavljeno, postoje dve osnovne funkcije za rad sa vrednostima `NULL`: funkcija `isNull`, koja proverava da li je niska `NULL`, i funkcija `ifNull`, koja kao rezultat daje vrednost niske, ako nije `NULL`, ili alternativnu vrednost, ako jeste `NULL`.

Jedan složeniji primer upita

Opisanim skupom funkcija se ni na koji način ne ograničava struktura izveštaja. Naredni program formira tabelu sa podacima o studentima u kojoj se za svakog studenta formira po izveštaj o svim položenim ispitima:

```
studentiOd(1995).izveštaj()

where {
    studentiOd(godina) = sql query {
        select indeks, god, ime, prezime
        from student
        where god >= :godina
        order by god, indeks
        };

    izveštaj(tab) = html template
        <html><head><title>Studenti</title></head><body>
        <#  tab.formatReport( zaglavlje, red,
                              "</table>", "Nema studenata!" )
        #></body><html><#>;

    zaglavlje = html template
        Tabela studenata <p>
        <table>
        <tr>
            <td><b>Indeks</b></td>
            <td><b>God.</b></td>
            <td><b>Ime</b></td>
            <td><b>Prezime</b></td>
            <td><b>Položio</b></td>
        </tr><#>;

    red(r) = html template <tr>
        <td><# r["INDEKS"] #></td>
        <td><# r["GOD"] #></td>
        <td><# r["IME"] #></td>
        <td><# r["PREZIME"] #></td>
        <td><# predmeti( r["INDEKS"].asInt(),
                         r["GOD"].asInt()
                        ).formatRowsSep(predRed,", ")
                        #></td>
        </tr><#>;
```

```
predmeti(indeks,god) = query {
        select pl.naziv
        from student s, plan pl, polagao po
        where pl.sifpred = pl.sifpred
          and pl.statg = s.statg
          and pl.sifprof = s.sifprof
          and po.indeks = s.indeks
          and po.god = s.god
          and po.ocena > 5
          and s.indeks = :indeks
          and s.god = :god
        };

predRed(r) = r["NAZIVPRED"];
}
```

2.7.4. Transakcije[24]

Transakciona funkcija, ili *transakcija*, je funkcija koja proizvodi bočni efekat samo u slučaju uspešnog završetka. *Akciona funkcija*, ili *akcija*, je funkcija koja pravi bočni efekat ali ne sadrži transakcionu logiku. Zbog toga akciona funkcija može da se upotrebljava samo u okviru neke transakcione funkcije, bilo neposredno ili posredno. Ako se neka transakciona funkcija upotrebljava (bilo neposredno ili posredno) u okviru izračunavanja neke šire transakcione funkcije, tada se ona tretira kao akciona funkcija i njena transakciona logika se zanemaruje.

Telo akcione ili transakcione funkcije se sastoji od niza izraza, koji mogu biti:

- izrazi transakcionog jezika primenjenog SUBP;
- WAFL izrazi čija je vrednost logičkog tipa i
- specifični akcioni izrazi programskog jezika WAFL.

Izrazi koji čine telo akcione funkcije izračunavaju se redom, dok se svi ne izračunaju ili dok se kao rezultat nekog od izraza ne dobije vrednost `false`. Ako neki izraz kao rezultat da `false`, izrazi koji slede se ne izračunavaju i rezultat izračunavanja funkcije je takođe vrednost `false`. Ako svi izrazi kao rezultat daju `true`, tada je i vrednost akcione funkcije `true`. Isto važi i u slučaju transakcionih funkcija koje se izračunavaju u okviru neke šire transakcije, pa se stoga tretiraju kao akcione funkcije. Ako tokom izračunavanja akcije bude proglašen izuzetak, on se čuva sa strane i kao rezultat akcione funkcije se vraća vrednost `false`. Način pristupanja nastalom izuzetku opisan je u odeljku *2.7.6 Izuzeci pri radu sa bazama podataka* na strani 72.

Transakcione funkcije, koje se ne izračunavaju u okviru nekih drugih transakcija, imaju nešto drugačiji tok izračunavanja jer obuhvataju i transakcionu logiku:

- najpre se započinje transakcija;
- zatim se izrazi izračunavaju kao u slučaju akcije i privremena vrednost transakcije se određuje na isti način kao vrednost akcije;
- kada je privremena vrednost transakcije izračunata, okončava se transakcija, i to:

[24] Rad sa transakcijama nije u potpunosti podržan. Nisu podržane akcije.

- ako je privremena vrednost `false`, transakcija se poništava i konačna vrednost transakcione funkcije je `false`;
- ako je privremena vrednost `true`, pokušava se potvrđivanje transakcije:
  - ako potvrđivanje transakcije uspe, konačna vrednost funkcije je `true`;
  - ako potvrđivanje transakcije ne uspe, transakcija se poništava i konačna vrednost funkcije je `false`.

Transakcione i akcione funkcije se definišu slično ostalim funkcijama, s tim što tela definicija transakcione i akcione funkcije imaju odgovarajući oblik:

```
<definicija_transakcije> ::=
    <ime> [<formalni_parametri>] =
    [sql] transaction [ isolation ( s | rr | rc | ru ) ]
    { {<akcioni_izraz>} }
    [<where_podizraz>] ;

<definicija_akcije> ::=
    <ime> [<formalni_parametri>] = [sql] action
    { {<akcioni_izraz>} }
    [<where_podizraz>] ;

<akcioni_izraz> ::=
    <naredba_transakcionog_jezika>
    | <naredba_set> | <naredba_reset>
    | <izraz>
```

Pri tome:

- ključna reč `sql` je opciona i označava da se u telu funkcije kao transakcioni jezik upotrebljava SQL; ako se ne navede podrazumeva se SQL;
- konkretne implementacije mogu podržati i druge transakcione jezike uz dodavanje odgovarajućih ključnih reči za njihovo označavanje;
- ukoliko se nivo izolovanosti ne navede eksplicitno, podrazumeva se nivo izolovanosti *Read Commited*;
- `<naredba_transakcionog_jezika>` može biti bilo koja naredba transakcionog jezika koja ne predstavlja upit; u slučaju SQL-a to može biti bilo koja naredba SQL-a koja se može izvršiti nad bazom podataka i ne predstavlja upit;
- `<naredba_set>` predstavlja zapis naredbe `set` za dodeljivanje vrednosti parametru sesije, u formi:

  ```
  set <naziv> = <izraz>
  ```

  gde je `<naziv>` naziv parametra sesije, a `<izraz>` neki WAFL izraz čiji je rezultat tipa `string` ili nekog drugog prostog tipa;
- `<naredba_reset>` predstavlja zapis naredbe `reset` za uklanjanje suvišnog parametra sesije, u formi:

  ```
  reset <naziv>
  ```

  gde je `<naziv>` naziv parametra sesije;

- `<izraz>` može biti bilo koji izraz na programskom jeziku WAFL čija je vrednost logičkog tipa, tj. ima vrednost `true` ili `false`.

Naredbe `set` i `reset` menjaju parametare sesije. Ako izraz kojim je definisana vrednost koja se dodeljuje parametru sesije nije tipa `string`, njegova vrednost se pre dodeljivanja konvertuje u tip `string` funkcijom `asString`. Obe naredbe su obuhvaćene mehanizmom transakcije tako da se sve do potvrđivanja transakcije promene ne vide od strane drugih programa.

Kao i svakoj drugoj definiciji funkcije, definicijama transakcija i akcija odgovaraju po dva prostora imena:

- prostor imena definicije, kome pripadaju formalni parametri definicije;
- prostor imena izraza `where`, kome pripadaju sva imena definisana neposredno u okviru opcionog izraza `where`.

Kao host promenljive u izrazima koji predstavljaju naredbe jezika SQL, mogu se pojavljivati samo imena koja su parametri definicije, tj. imena iz prostora imena definicije. Imena iz prostora imena opcionog izraza `where` mogu se koristiti u WAFL izrazima koji čine transakcionu ili akcionu funkciju i u onim izrazima koji su sastavni deo izraza `set`.

Naredni primer ažurira prosečnu ocenu studentima koji su upisali fakultet pre 1998. godine:

```
ažuriranjeProseka()
where {
    ažuriranjeProseka = transaction {
        update studenti
        set prosek = ( select avg(ocena)
                       from polagao
                       where ocena>5
                         and indeks = studenti.indeks
                         and god = studenti.god )
        where god < 1998;
        };
    };
```

U narednom primeru se ilustruje upotreba promenljivih sesije. Program ažurira prosečnu ocenu datog studenta i pamti je na nivou sesije kao promenljivu `prosek`:

```
ažuriranje(indeks,god) = transaction {
    // ažuriramo prosek u bazi podataka
    ažuriranjeProseka(indeks,god);

    // postavljamo vrednost promenljive sesije "prosek"
    set prosek = čitanjeProseka(indeks,god);
    }
where {
    ažuriranjeProseka(indeks,god) = transaction {
        update studenti
        set prosek = ( select avg(ocena)
                       from polagao
                       where ocena>5
                         and indeks = :indeks
                         and god = :god )
        where indeks = :indeks
          and god = :god;
        };
```

```
čitanjeProseka(indeks,god) =
    prosekUpit(indeks,god).hd()["PROSEK"];

prosekUpit(indeks,god) = query {
        select prosek from studenti
        where indeks = :indeks and god = :god
        };
};
```

Iterativne transakcije

Iterativnim transakcijama ćemo nazivati transakcije koje se izvode nad većim brojem podataka ponavljanjem nekog postupka za svaki od podataka. Primetimo da na konceptualnom nivou pojam *iteracije* nema mnogo smisla u funkcionalnom programskom jeziku WAFL. Ipak, kako su transakcije posebna vrsta funkcija sa semantikom koja nalaže redosled izračunavanja, taj redosled izračunavanja može imati iterativnu prirodu.

Iterativne transakcije se na programskom jeziku WAFL implementiraju tako što se definiše akciona funkcija koja izračunava promenu jednog objekta, a zatim i transakciona funkcija u kojoj se obezbeđuje izračunavanje definisane akcione funkcije za sve objekte nad kojima je potrebno izvesti transakciju.

Naredni primer ažurira prosečnu ocenu studentima koji su upisali fakultet pre 1998. godine. Funkcija `studenti` predstavlja upit koji izdvaja podatke o svim studentima fakulteta, tj. izračunava listu čiji svaki element predstavlja katalog sa podacima o jednom studentu. Funkcija `uslov` proverava da li je student upisao fakultet pre 1998. godine. Funkcija `ažuriranjeJednog` ažurira podatke o jednom studentu, pri čemu akciona funkcija `ažuriranje0` neposredno izvodi bočni efekat menjanjem podataka u bazi podataka o studentu sa datim brojem indeksa i godinom upisa. Transakciona funkcija `ažuriranjeProseka` ažurira podatke o svim studentima tako što proverava da li je za svakog studenta koji zadovoljava `uslov` uspešno izvedena akciona funkcija `ažuriranjeJednog`. U slučaju uspešnog izračunavanja (tj. sa rezultatom `true`) akcione funkcije za svakog odgovarajućeg studenta, transakcija uspeva:

```
ažuriranjeProseka()
where {
    ažuriranjeProseka = transaction {
        studenti
            .filter( uslov )
            .forall( ažuriranjeJednog );
        };

    studenti = query { select * from studenti };

    uslov( student ) = student["GOD"].asInt() < 1998;

    ažuriranjeJednog( student ) =
            ažuriranje0( student["GOD"].asInt(),
                         student["INDEKS"].asInt() )
    where ažuriranje0( god, indeks ) = action {
        //  ova akcija sadrži samo jednu SQL naredbu
        update studenti
        set prosek = ( select avg(ocena)
                       from polagao
                       where ocena>5
                         and indeks = :indeks
                         and god = :god )
        where indeks = :indeks
          and god = :god;
```

```
        };
    };
```

Zaključavanje podataka i izolovanost transakcija

WAFL servis se izvršava u konkurentnim uslovima, jer je moguće da više korisnika istovremeno zahteva neke resurse. Postoji i mogućnost da oni zatraže ili pokušaju da menjaju iste podatke u isto vreme. Zbog toga je WAFL servis podložan problemima koji se pojavljuju u konkurentnim okruženjima. Resursi okruženja u kome se izračunavaju programi WAFL servisa, a koji su podložni problemima usled konkurentnosti, jesu, pre svega, oni elementi koji su promenljivi. Kako je već ranije rečeno, jedini potencijalno promenljivi elementi su:

- baza podataka,
- sistem datoteka;
- spoljni objekti i
- parametri sesija.

Uvođenjem koncepta transakcionih funkcija i lokalizovanjem svih promena podataka, bilo da se radi o podacima u bazi podataka, datotekama ili parametrima sesije, dobija se i na lokalizovanju problema zaključavanja podataka. Poseban aspekt problema predstavljaju upiti koji samo pristupaju podacima iz baze podataka, kao i funkcije koje samo čitaju sadržaj datoteka, ali je i tu suština problema na strani transakcije koja pokušava da menja te podatke.

Da bi se postigla što veća efikasnost aplikacija pisanih na programskom jeziku WAFL, a uz postizanje bezbednosti podataka i transakcija u smislu međusobne izolovanosti, definiše se skup pravila po kojima mora funkcionisati svaka implementacija programskog jezika WAFL. Neka od tih pravila su uobičajena za sisteme sa transakcijama, pa i za baze podataka, a neka su specifična i odražavaju specifičnosti programskog jezika:

1. sve do završetka transakcije, sve promene izvedene u njenim okvirima moraju biti nedostupne drugim programima;
2. ako transakcija uspe, sve promene izvedene u njenim okvirima, bilo na bazi podataka, na sistemu datoteka ili na parametrima sesije, moraju biti potvrđene i od tog trenutka dostupne svim programima te sesije i šire;
3. ako ne uspe potvrđivanje bilo kog tipa promena u okviru neke transakcije, tada se mora poništiti čitava transakcija;
4. ako transakcija ne uspe, sve promene izvedene u njenim okvirima, bilo na bazi podataka, na sistemu datoteka ili na parametrima sesije, moraju biti poništene i nedostupne svim programima;
5. podrazumevani nivo izolovanosti za operacije nad bazom podataka u okviru transakcije je *Repeatable Read*;
6. datoteke kojima se pristupa u okviru transakcija otvaraju se u ekskluzivnom režimu, koji obezbeđuje najviši nivo izolovanosti programa;
7. podrazumevani nivo izolovanosti za upite koji se postavljaju van transakcija je *Read Uncommited*, čime se izbegava suvišno zaključavanje podataka;

8. datoteke koje se čitaju van transakcija otvaraju se u deljivom režimu koji omogućava najviši nivo konkurentnosti za odgovarajući operativni sistem ali i garantuje njihovu nepromenljivost tokom čitanja.

Prva četiri pravila su uobičajena pravila za transakcije. Poslednja dva pravila se odnose na način upotrebe resursa van transakcija, a u cilju postizanja što veće efikasnosti. Niska izolovanost može imati za posledicu neispravne rezultate pri čitanju iz baze podataka ili sistema datoteka, ali se to može izbeći na dva osnovna načina:

- na nivou servisa se može definisati neki drugi podrazumevani nivo izolovanosti za upite i režim deljenja datoteka za čitanje (*Dodatak D - Parametri servisa* na strani 91.);
- za pojedinačne upite i pristupanja datotekama se može eksplicitno zahtevati neki viši nivo izolovanosti ili neki stroži režim deljenja datoteka.

Da bi svi ovi uslovi bili ispunjeni, implementacija transakcionog sistema mora uzeti u obzir sve specificnosti distribuiranih transakcija, jer ovde su u pitanju četiri osnovna međusobno distribuirana objekta: baza podataka, sistem datoteka, spoljni objekti i parametri sesije. Olakšica je što se pod određenim uslovima u okviru implementacije može podrazumevati da potvrđivanje promena parametara sesije ne može biti neuspešno, pa se problem svodi na bazu podataka i sistem datoteka, a u slučaju da se u transakciji menja samo jedan od ovih resursa, tada se problem dalje pojednostavljuje.

Narednim zahtevima se definišu uslovi koje moraju ispuniti programi pisani na WAFL-u da ne bi nepotrebno zadržavali katanace u većem broju ili dužem intervalu u odnosu na stvarne potrebe:

- ako se u programu koriste dijalozi, ne smeju se koristiti u okviru transakcija, jer će svi resursi koje je transakcija već zaključala ostati zaključani bar još do okončanja dijaloga, što znači da su nepotrebno zaključani tokom trajanja prenosa podataka i intervala u kome korisnik priprema podatke za obradu;
- ako se u programu koriste dijalozi i upiti ili čitanje datoteka, ne preporučuje se da se čitanje podataka bilo iz rezultata upita, bilo iz datoteke vrši delom pre započinjanja a delom posle okončanja dijaloga, jer se na taj način tokom očekivanja odgovora (na pitanje postavljeno u dijalogu) značajni resursi sistema (radna memorija, otvorene datoteke sistema datoteka, otvoreni kursori sistema za upravljanje bazama podataka) drže zaključani; ako su isti resursi potrebni i pre i posle komunikacije sa korisnikom, često je efikasnije ponoviti čitanje nego čuvati pročitane podatke, posebno ako se ima na umu potencijalno veliki broj istovremenih dijaloga.

2.7.5. Dinamički SQL[25]

Opisani načini izračunavanja rezultata upita i izvođenja SQL naredbi u transakcijama podrazumevaju da je u trenutku pisanja programa poznato koji se upiti i SQL naredbe

[25] Funkcije za dinamički SQL nisu implementirane. Biće implementirane u najskorijoj budućnosti.

izračunavaju. Ukoliko tekst upita ili naredbe nije poznat u vreme pisanja programa, opisan način rada to neće omogućiti. Zbog toga je uvedena podrška za *dinamički SQL*.

Dinamički SQL podrazumeva postavljanje upita i izvršavanje SQL naredbi čiji se tekst ne definiše u trenutku pisanja programa nego u toku njegovog izračunavanja. Tekst SQL naredbi koje se dinamički izračunavaju saopštava se u obliku niske, a parametri u obliku kataloga parametara. Način upotrebe parametara je isti kao u slučaju uobičajenih upita. Sledi pregled funkcija za rad sa dinamičkim SQL-om:

Funkcija	Opis

`dynamicSQLQuery(q,p)`	Izračunava SQL upit koji je zapisan u okviru niske `q` uz upotrebu parametara `p`. Rezultat upita je lista redova, kao u slučaju već opisivanih običnih upita. Tip funkcije:
	`string * map[string][string]`
	`-> List[Map[string][string]]`

`dynamicSQLCommand(q,p)`	Izračunava SQL naredbu definisanu niskom `q` i parametrima `p`. Rezultat je `true` ako je sve u redu ili `false` ako nije. Funkcija `dynamicSQLCommand` je akciona funkcija i ne sme se upotrebljavati van okvira neke transakcije. Tip funkcije je: `string * map[string][string] -> bool`

Tabela 30: Funkcije za dinamiči SQL

Funkcija `dynamicSQLCommand` ima i oblik `dynamicSQLCommandE` koji u slučaju neuspeha proizvodi odgovarajući izuzetak.

2.7.6. Izuzeci pri radu sa bazama podataka[26]

Izuzeci do kojih dolazi pri radu sa bazama podataka mogu se podeliti na one do kojih dolazi pri postavljanju upita i na one do kojih dolazi pri izvršavanju transakcija.

Pri postavljanju upita, bez obzira na to da li se radi o statičkom ili dinamičkom upitu, može doći do izuzetaka `SQLQueryException` i `DBException`.

Pri izvršavanju SQL naredbe koja je deo neke transakcije, u slučaju njenog neuspeha se kao rezultat vraća vrednost `false`. Međutim, objekat izuzetka se ipak formira ali se ne proglašava, već se samo čuva sa strane. Za pristupanje eventualno napravljenim izuzecima može se upotrebljavati funkcija:

```
transactionExceptions : () -> List[Exception]
```

Funkcija `transactionExceptions` nema parametara i ima za rezultat listu svih izuzetaka koji su nastali od početka pa do neuspešnog završetka poslednje izračunavane transakcije, u obrnutom redosledu u odnosu na redosled nastajanja.

Pri izvršavanju SQL narebi mogu se formirati izuzeci: `DBException`, `SQLException`, `SQLCmdException` i `SQLQueryException`. Pored njih, u listi izuzetaka koja predstavlja rezultat funkcije `transactionExceptions` mogu se i drugi tipovi izuzetaka.

[26] Nije implementiran rad sa izuzecima, pa zato nije implementiran ni ovaj segment.

2.8. Upotreba spoljnih objekata[27]

Pod spoljnim objektima podrazumevaju se objekti čijim pravljenjem, manipulisanjem i uklanjanjem ne upravlja neposredno program na programskom jeziku WAFL. Koncept upotrebe spoljnih objekata u programskom jeziku WAFL definisan je sa ciljem omogućavanja komunikacije sa objektima srednjeg sloja aplikacija na više nivoa, a pre svega za rad sa distribuiranim objektima posredstvom tehnologija kao što su CORBA i COM.

[27] Rad sa spoljnim objektima još nije implementiran.

Dodatak A - Formalna sintaksa

Sintaksa programskog jezika WAFL formalno je predstavljena primenom proširene BNF:

- terminali se zapisuju istaknuto;
- neterminali se zapisuju u obliku `<neterminal>`;
- znak `|` razdvaja alternativne elemente;
- male zagrade se upotrebljavaju za grupisanje elemenata;
- zapis `[<element>]` označava opciono (jedno ili nijedno) pojavljivanje elementa;
- zapis `{<element>}` označava nula ili više ponavljanja obuhvaćenog elementa;
- zapis `<neterminal> ::= izraz` označava da se sintaksa neterminala formalno definiše datim izrazom.

Formalna sintaksa programskog jezika WAFL[28]

Osnovu programiranja na programskom jeziku WAFL predstavlja pisanje modula koji čine aplikaciju. Sledi formalna sintaksa svih vrsta modula na programskom jeziku WAFL:

```
<WAFL_modul> ::=
    <programski_modul> | <klasni_modul>
    | <funkcijski_modul> | <bibliotečki_modul>
    | <šablonski_modul>

<programski_modul> ::= <telo_programa> [;]
<telo_programa> ::= <izraz_except>
<klasni_modul> ::= class <naziv_klase> <telo_klase>;
<funkcijski_modul> ::=
    [ function [<ime>] <formalni_parametri> = ]
    <telo_funkcije> ;
<bibliotečki_modul> ::=
    library [<ime>] { <definicija> {<definicija>} }
    [<where_podizraz>]
<šablonski_modul> ::=
    [ html | xml ] template [<ime>] <formalni_parametri>
    <zapis_šablona>

<definicija> ::=
    <definicija_funkcije> | <definicija_klase>
    | <definicija_upita>
    | <definicija_transakcije> | <definicija_akcije>
```

[28] Predstavljena je puna sintaksa programskog jezika WAFL, uključujući i neimplementirane elemente.

```
    | <deklaracija_biblioteke> | <deklaracija_funkcije>
    | <deklaracija_klase>

<definicija_funkcije> ::=
    <ime> [<formalni_parametri>] = <telo_funkcije> ;
<telo_funkcije> ::= <izraz_except>
<definicija_upita> ::=
    <ime> [<formalni_parametri>] =
    [sql] query [ isolation ( s | rr | rc | ru ) ]
    { <upit_na_upitnom_jeziku> } ;
<definicija_transakcije> ::=
    <ime> [<formalni_parametri>] =
    [sql] transaction [ isolation (s | rr | rc | ru ) ]
    { {<akcioni_izraz>} }
    [<where_podizraz>] ;
<definicija_akcije> ::=
    <ime> [<formalni_parametri>] = [sql] action
    { {<akcioni_izraz>} }
    [<where_podizraz>] ;
<deklaracija_funkcije> ::=
    <ime> [<formalni_parametri>] = function [file] <niska> ;
<deklaracija_biblioteke> ::=
    <ime> = library [[file | ( extern [<ime>] ) ] <niska>] ;
<formalni_parametri> ::= ( [ <ime> {, <ime> } ] )

<definicija_klase> ::=
    <naziv_klase> = class <telo_klase>;
<telo_klase> ::=
    [<deklaracija_nasleđivanja>]
    [members { <blok_def_članova> }]
    methods { <blok_def_metoda> }
    [where { <blok_pom_def_klase> }]
<deklaracija_nasleđivanja> ::= : <ime>

<blok_def_članova> ::= <definicija_člana>{<definicija_člana>}
<blok_def_metoda > ::= <definicija_metoda>{<definicija_metoda>}
<blok_pom_def_klase > ::= <pom_def_klase>{<pom_def_klase>}
<pom_def_klase> ::= <definicija_metoda> | <definicija_klase>

<definicija_člana> ::=
    <definicija_običnog_člana>
    | <definicija_statičkog_člana>
<definicija_običnog_člana> ::= <ime> ;
<definicija_statičkog_člana> ::= static <ime> = <izraz> ;
<definicija_metoda> ::= [static] <definicija_funkcije>

<deklaracija_klase> ::=
    <ime> = class [ file | ( extern [<ime>] )] <niska> ;

<akcioni_izraz> ::=
    <naredba_transakcionog_jezika>
    | <naredba_set> | <naredba_reset>
    | <izraza>
<naredba_set> ::= set <ime> = <izraz> ;
<naredba_reset> ::= reset <ime> ;

<izraz_except> ::= <izraz_where> [<except_podizraz>]
<except_podizraz> ::= except { <hvatač> { <hvatač>} }
<hvatač> ::= catch ( [[<ime>] <ime>] ) = <izraz_except> ;

<izraz_where> ::= <izraz> [<where_podizraz>]
<where_podizraz> ::= where { <definicija> {<definicija>} }
```

```
<izraz> ::= <operatorski_izraz> | <tekstualan_šablon>
<operatorski_izraz> ::=
    { <unarni_operator> } <prost_izraz>
    [ <binarni_operator> <operatorski_izraz> ]
<tekstualan_šablon> ::=
    [ html | xml ] template <zapis_šablona> <#>
<oznaka_za_umetanje_izraza> ::= <# <operatorski_izraz> #>

<prost_izraz> ::=
    <literalna_konstanta>
    | <zapis_liste>
    | <zapis_niza>
    | <zapis_kataloga>
    | <kvalifikovano_ime>
    | <izraz_u_zagradi>
    | <uslovni_izraz>
    | <izraz_sa_aplikacijom>

<izraz_u_zagradi> ::= ( <izraz> )
<zapis_liste> ::= [ [ <izraz> {, <izraz> } ] ]
<zapis_niza> ::= { <izraz> {, <izraz> } }
<zapis_kataloga> ::= [[<element_kataloga>{,<element_kataloga>}]]
<element_kataloga> ::= < <izraz>, <izraz> >

<izraz_sa_aplikacijom> ::= <nosilac_aplikacije>
    ( <aplikacija> | <indeksiranje> | <poziv_metoda> )
<nosilac_aplikacije> ::=
    <izraz_u_zagradi>
    | <kvalifikovano_ime>
    | <izraz_sa_aplikacijom>
<aplikacija> ::= ( [ <izraz> {, <izraz> } ] )
<indeksiranje> ::= [<izraz>]
<poziv_metoda> ::= .<kvalifikovano_ime><aplikacija>

<uslovni_izraz> ::= <izraz_if> | <izraz_switch>
<izraz_if> ::= if <izraz> then <izraz> else <izraz>
<izraz_switch> ::=
    switch( <izraz> )
    { <klauzula_case> {<klauzula_case>} <klauzula_default> }
<klauzula_case> ::=
    <nepotpuna_klauzula_case> {<nepotpuna_klauzula_case>}
    [<izraz>;]
<nepotpuna_klauzula_case> ::= case <literalna_konstanta> :
<klauzula_default> ::= default: <izraz>

<literalna_konstanta> ::=
    <niska> | <ceo_broj> | <realan_broj> | <logička_vrednost>
<niska> ::= <deo_niske>{{<prazan_prostor>}<deo_niske>}
<deo_niske> ::= ("{<znak_osim_">}") | ('{<znak_osim_'>}')
<logička_vrednost> ::= true | false
<realan_broj> ::= <ceo_broj>.[<niz_cifara>][<eksponent>]
<eksponent> ::= (e|E)<ceo_broj>
<ceo_broj> ::= [<znak_broja>]<niz_cifara>
<niz_cifara> ::= <cifra>{<cifra>}
<znak_broja> ::= + | -

<kvalifikovano_ime> ::= [<kvalifikovano_ime>::]<ime>
<ime> ::= <slovo>{<slovo>|<cifra>|_}

<unarni_operator> ::= - | not | ~
<binarni_operator> ::=
    + | - | * | / | % | == | = | != | <> |
```

```
    < | > | and | or | | | & | && | || | :

<komentar> ::= <c_komentar> | <c++_komentar>
<c_komentar> ::= /* { <znak> } */
<c++_komentar> ::= // { <znak> } <novi_red>

<znak> ::=
    <slovo> | <cifra> | <običan_simbol> | <poseban_simbol>
    | <prazan_prostor> | <escape_znak>
<znak_osim_"> ::=
    <slovo> | <cifra> | <običan_simbol> | ' | \
    | <prazan_prostor> | <escape_znak>
<znak_osim_'> ::=
    <slovo> | <cifra> | <običan_simbol> | " | \
    | <prazan_prostor> | <escape_znak>

<slovo> ::= <malo_slovo> | <veliko_slovo>
<veliko_slovo> ::= <veliko_slovo_ascii> | <veliko_slovo_lsz>
<malo_slovo> ::= <malo_slovo_ascii> | <malo_slovo_lsz>
<veliko_slovo_ascii> ::=
    A | B | C | D | E | F | G | H | I | J | K | L | M |
    N | O | P | Q | R | S | T | U | V | W | X | Y | Z
<malo_slovo_ascii> ::=
    a | b | c | d | e | f | g | h | i | j | k | l | m |
    n | o | p | q | r | s | t | u | v | w | x | y | z
<malo_slovo_lsz>[29] ::= ć | č | đ | š | ž
<veliko_slovo_lsz> ::= Ć | Č | Đ | Š | Ž

<cifra> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<običan_simbol>[30] ::=
    ! | # | $ | % | & | ( | ) | * | + | , | - | . | / | : |
    ; | < | = | > | ? | @ | [ | ] | ^ | _ | ` | { | | | } | ~
<poseban_simbol> ::= " | ' | \

<prazan_prostor> =
    <blanko> | < tabulator> | <novi_red> | <nova_strana>
<blanko>[31] ::= □
<tabulator> ::= □
<novi_red> ::= □
<nova_strana> ::= □

<escape_znak> ::=
    <escape_navodnik> | <escape_apostrof>
    | <escape_obrnuta_kosa_crta> | <escape_novi_red>
    | <escape_povratak_na_početak_reda>
    | <escape_horizontalni_tabulator>
    | <escape_vertikalni_tabulator> | <escape_nova_strana>
    | <escape_jedan_znak_unazad>
<escape_navodnik> ::= \"
<escape_apostrof> ::= \'
<escape_obrnuta_kosa_crta> ::= \\
```

[29] <malo_slovo_lsz> i <veliko_slovo_lsz> su skupovi malih i velikih slova u lokalnom skupu znakova. Predstavljena definicija se odnosi na srpsku latinicu.

[30] Pored navedenih simbola, običan simbol može biti i neki drugi simbol, specifičan za određeni lokalni skup znakova.

[31] <blanko>, <tabulator> i <novi_red> su rezervisani simboli razmaka, tabulacije i novog reda.

```
<escape_novi_red> ::= \n
<escape_povratak_na_početak_reda> ::= \r
<escape_horizontalni_tabulator> ::= \t
<escape_vertikalni_tabulator> ::= \v
<escape_nova_strana> ::= \f
<escape_jedan_znak_unazad> ::= \b
```

Neterminali koji nisu formalno definisani

U okviru formalne definicije sintakse nisu definisani neki neterminali. Oni se neformalno definišu na sledeći način:

- `<zapis_šablona>` predstavlja ispravan zapis šablona, kako je opisan u odeljku *2.6.3 Tekstualni šabloni* na strani 47. U najkraćem, to je segment HTML dokumenta u kome se mogu naći posebne oznake za umetanje izraza, formalno definisane kao neterminal `<oznaka_za_umetanje_izraza>`.
- `<upit_na_upitnom_jeziku>` predstavlja ispravan zapis upita na upitnom jeziku kojim je upit definisan. Podrazumevani upitni jezik je SQL, ali konkretne implementacije mogu podržati i druge upitne jezike.
- `<naredba_transakcionog_jezika>` predstavlja ispravan zapis naredbe transakcionog jezika. Podrazumevani transakcioni jezik je SQL, ali konkretne implementacije mogu podržati i druge transakcione jezike.

Formalna sintaksa zapisa tipova

U programskoj dokumentaciji se tipovi definisanih imena moraju zapisivati u skladu sa formalnom definicijom neterminala `<zapis_tipa_imena>`. Sledi formalna sintaksa zapisa tipova:

```
<zapis_tipa_imena> ::=
    <kvalifikovano_ime> : ( <tip> | <tip_metoda> )

<tip> ::=
    <prost_tip>
    | <prost_polimorfan_tip>
    | <zbirni_tip>
    | <kombinovan_tip>
    | <tip_kolekcije>
    | <tip_funkcije>
    | <tip_klase>

<prost_tip> ::= int | float | bool | string
<prost_polimorfan_tip> ::= '<slovo>
<zbirni_tip> ::= { <tip>, <tip>{, <tip>} }[<prost_polimorfan_tip>]
<kombinovan_tip> ::= <numerički_tip> | <vrednosni_tip>
<numerički_tip> ::= Numeric[<prost_polimorfan_tip>]
<vrednosni_tip> ::= Value[<prost_polimorfan_tip>]

<tip_kolekcije> ::= <tip_liste> | <tip_niza> | <tip_kataloga>
<tip_liste> ::= List[<tip>]
<tip_niza> ::= Array[<tip>]
<tip_kataloga> ::= Map[<tip>][<tip>]

<tip_funkcije> ::= ( <prazan_tip> | ( <tip>{*<tip>} ) ) -> <tip>
<prazan_tip> ::= ()
```

```
<tip_klase> ::= <ime>{[<tip_člana>]}
<tip_člana> ::= <tip> | this

<tip_metoda> ::= (<ime>) {* <tip> } -> <tip>
```

Rezervisane reči

Rezervisane reči programskog jezika WAFL mogu se podeliti na:

- ključne reči – reči koje imaju specifično značenje u sintaksi programskog jezika;
- rezervisana imena – imena koja imaju predefinisano značenje i ne mogu se koristiti za imenovanje definicija.

Ključne reči programskog jezika WAFL su:

```
action     and          case      class       default     else
except     extern       file      function    html        if
isolation  library      members   methods     not         or
query      rc           reset     rr          ru          s
set        sql          static    switch      template    then
this       transaction  where     xml
```

Rezervisana imena su:

```
base_method  catch    constructor   false    hd        nil
nilObject    this     throw         tl       true
```

Većinu ključnih reči nije dopušteno upotrebljavati kao imena. Ipak, neke ključne reči imaju posebno značenje samo u strogo definisanom kontekstu pa se ipak smeju upotrebljavati i kao imena. To su ključne reči:

```
case     default  extern   file   isolation   members   methods
rc       reset    rr       ru     s           set
```

Ključne reči `set` i `reset` se mogu upotrebljavati kao imena, ali će svako njihovo pojavljivanje na početku naredbe transakcije biti tumačeno kao početak naredbe `set`, odnosno `reset`.

Posebnu kategoriju rezervisanih reči čine rezervisani parametri URL upita. To su parametri oblika `start`*X* i `row`*X*, gde je *X* neki ceo broj. Ovi parametri su opisani u odeljku *Funkcije za generisanje izveštaja sa navigacijom* na strani 61.

Dodatak B - Standardna biblioteka

Ovim dodatkom se rezimiraju opisane funkcije i operatori. Sve funkcije su grupisane po bibliotekama. Za svaku od funkcija se navodi naziv, tip i vrsta, kao i biblioteka kojoj pripada. Vrsta funkcije može biti:

- *deterministička* (oznaka 'D') – u svim kontekstima za date argumente ima isti rezultat;
- *nedeterministička* (oznaka 'N') – rezultat ne zavisi samo od argumenata nego i od okruženja;
- *lokalno deterministička* (oznaka 'LD') – nedeterminističke funkcije koje se u okviru jednog izračunavanja programa ponašaju se kao determinističke;
- *akciona* (oznaka 'A') – funkcija može proizvesti bočni efekat.

Standardna biblioteka programskog jezika WAFL obuhvata biblioteke: Math, String, Conversion, List, Array, Map, Files, WebApplication, Dialog, Report, NavigationReport, DynamicSQL, ExternalObject i MimeTypes[32].

Operatori i neke osnovne funkcije za manipulisanje primitivnim tipovima podataka smatraju se za ugrađene i ne definišu se bibliotekama. Svi operatori i osnovne funkcije osim `throw` pripadaju kategoriji determinističkih potprograma. Funkcija `throw` ima osobine i determinističkih i akcionih funkcija.

Programski jezik WAFL raspolaže sa tri unarna operatora:

Operator	Tip
`-`	`Numeric['a] -> Numeric['a]`
`~`	`int -> int`
`! (not)`	`bool -> bool`

Binarni operatori u programskom jeziku WAFL su:

Operator	Tip
`+`	`Value['a] * Value['a] -> Value['a]`
`-`	`Numeric['a] * Numeric['a] -> Numeric['a]`

[32] U ovom dodatku su opisane i biblioteke koje još nisu implementirane. To je posebno naznačeno i uz svaku pojedinačnu neimplementiranu funkciju ili biblioteku.

Operator	Tip
*	Numeric['a] * Numeric['a] -> Numeric['a]
/	Numeric['a] * Numeric['a] -> Numeric['a]
%	int * int -> int
&	int * int -> int
|	int * int -> int
&& (and)	bool * bool -> bool
|| (or)	bool * bool -> bool
:	'a * List['a] -> List['a]
<	Value['a] * Value['a] -> Value['a]
<=	Value['a] * Value['a] -> Value['a]
>	Value['a] * Value['a] -> Value['a]
>=	Value['a] * Value['a] -> Value['a]
==	'a * 'a -> 'a
!= (<>)	'a * 'a -> 'a

Ugrađene osnovne funkcije su:

Funkcija	Tip
hd	List['a]->'a
tl	List['a]->List['a]
throw[33]	Exception -> 'a

Biblioteka Math

Standardna biblioteka `Math` sadrži osnovne matematičke funkcije. Sve funkcije ove biblioteke su deterministične. Ona obuhvata funkcije nad celim brojevima:

Funkcija	Tip
abs	int -> int
random	int -> int

i funkcije nad realnim brojevima:

Funkcija	Tip
abs	float -> float
sqrt	float -> float
pow	float * float -> float
exp	float -> float
ln	float -> float
log	float -> float
sin	float -> float
cos	float -> float
tan	float -> float
asin	float -> float
acos	float -> float
atan	float -> float
atan2	float * float -> float
sinh[34]	float -> float

[33] Funkcija nije implementirana.

Funkcija	Tip
cosh[35]	float -> float
tanh[36]	float -> float
round	float -> int
ceil	float -> int
floor	float -> int

Navedene funkcije su detaljnije opisane u odeljku *2.2.1 Prosti tipovi podataka*.

Biblioteka String

Standardna biblioteka `String` sadrži osnovne funkcije za rad sa niskama. Priložen je spisak funkcija i njihovih tipova. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
isNull	string -> bool
ifNull	string * string -> string
strLen	string -> int
strCat	string * string -> string
subStr	string * int * int -> string
strLeft	string * int -> string
strRight	string * int -> string
strLTrim	string -> string
strRTrim	string -> string
strTrim	string -> string
strPos	string * string -> int
strNextPos	string * string * int -> int
strLastPos	string * string -> int
strNextLastPos	string * string * int -> int
strPosI	string * string -> int
strNextPosI	string * string * int -> int
strLastPosI	string * string -> int
strNextLastPosI	string * string * int -> int
strLowerCase	string -> string
strUpperCase	string -> string
strReplace	string * string * string * int -> string
strReplaceI	string * string * string * int -> string
strReplaceAll	string * string * string -> string
strReplaceAllI	string * string * string -> string
strSplit	string * string -> List[string]
strJoin	List[string] * string -> string
strReverse	string -> string

Navedene funkcije su detaljnije opisane u odeljku *2.2.1 Prosti tipovi podataka*.

[34] Funkcija nije implementirana.

[35] Funkcija nije implementirana.

[36] Funkcija nije implementirana.

Biblioteka Conversion

Standardna biblioteka `Conversion` sadrži osnovne funkcije za konverziju prostih tipova podataka. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
`asInt`	`{ float, string, bool } -> int`
`asFloat`	`{ int, string, bool } -> float`
`asString`	`{ int, float, bool } -> string`
`asChar`	`{ int, float, bool } -> string`
`asBool`	`{ int, float, string } -> bool`

Navedene funkcije su detaljnije opisane u odeljku *2.2.1 Prosti tipovi podataka*.

Biblioteka List

Standardna biblioteka `List` sadrži funkcije za rad sa listama. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
`empty`	`List['a] -> bool`
`length`	`List['a] -> int`
`longerThan`	`List['a] * int -> bool`
`map`	`List['a] * ('a->'b) -> List['b]`
`forall`	`List['a] * ('a->bool) -> bool`
`exists`	`List['a] * ('a->bool) -> bool`
`aggregate`	`List['a] * ('a*'b -> 'b) * 'b -> 'b`
`leftAggregate`	`List['a] * ('a*'b -> 'b) * 'b -> 'b`
`filter`	`List['a] * ('a->bool) -> List['a]`
`subList`	`List['a] * int * int -> List['a]`
`forced`	`List['a] -> List['a]`
`listAsString`	`List['a] -> string`

Primetimo da funkcije `hd` i `tl` ne pripadaju biblioteci `List` jer predstavljaju osnovne funkcije koje su ugrađene u interpretator.

Navedene funkcije su detaljnije opisane u odeljku *2.4.3 Lista* na strani 36.

Biblioteka Array[37]

Standardna biblioteka `Array` sadrži funkcije za rad sa nizovima. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
`[]`	`Array['a] * int -> 'a`
`emptyArray`	`() -> Array['a]`
`array`	`int * 'a -> Array['a]`
`asList`	`Array['a] -> List['a]`
`asArray`	`List['a] -> Array['a]`
`setEl`	`Array['a] * int * 'a -> Array['a]`

[37] Implementirane su samo funkcije za konverziju niza u listu i obratno i operator za pristupanje elementima niza.

Funkcija	Tip
addEl	Array['a] * 'a -> Array['a]
length	Array['a] -> int
subArray	Array['a] * int * int -> Array['a]

Navedene funkcije su detaljnije opisane u odeljku *2.4.4 Niz* na strani 38.

Biblioteka Map

Standardna biblioteka `Map` sadrži funkcije za rad sa katalozima. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
[]	Map['a]['b] * 'a -> 'b
emptyMap[38]	() -> Map['a]['b]
createMap	Array['a] * Array['b] -> Map['a]['b]
setEl[39]	Map['a]['b] * 'a * 'b -> Map['a]['b]
addEl[40]	Map['a]['b] * 'a * 'b -> Map['a]['b]
length[41]	Map['a]['b] -> int
keys	Map['a]['b] -> Array['a]
key	Map['a]['b] * int -> 'a
values	Map['a]['b] -> Array['b]
value	Map['a]['b] * int -> 'b
findKey	Map['a]['b] * 'a -> int
findValue	Map['a]['b] * 'b -> int
hasKey	Map['a]['b] * 'a -> bool
hasvalue	Map['a]['b] * 'b -> bool
mapAsString	Map['a]['b] -> string

Navedene funkcije su detaljnije opisane u odeljku *2.4.5 Katalog* na strani 39.

Biblioteka Files[42]

Standardna biblioteka `Files` sadrži funkcije za rad sa datotekama.

Funkcija	Tip	Vrsta
fileRead	string -> string	N
fileWrite	string * string -> bool	A
fileReadRecords	string * int -> List[string]	N
fileReadRecordsA	string * int -> Array[string]	N
fileWriteRecord	string * int * string -> bool	A
fileAppend	string * string -> bool	A
fileExists	string -> bool	N
fileGetAttrs	string -> string	N

[38] Funkcija nije implementirana.

[39] Funkcija nije implementirana.

[40] Funkcija nije implementirana.

[41] Funkcija nije implementirana.

[42] Biblioteka nije implementriana.

Funkcija	Tip	Vrsta
`fileSetAttrs`	`string * string -> bool`	A
`fileCopy`	`string * string -> bool`	A
`fileMove`	`string * string -> bool`	A
`fileRemove`	`string -> bool`	A
`dirFiles`	`string -> List[string]`	N
`dirSubdirs`	`string -> List[string]`	N
`dirCreate`	`string -> bool`	A
`dirRemove`	`string -> bool`	A

Navedene funkcije su detaljnije opisane u odeljku *2.4.6 Datoteke* na strani 41.

Biblioteka WebApplication

Standardna biblioteka `WebApplication` sadrži funkcije specifične za razvoj Veb aplikacija. Obuhvata funkcije za rad sa parametrima formulara, sesije i servisa (*2.6.2 Parametri okruženja*, strana 45.):

Funkcija	Tip	Vrsta
`Service`	`() -> Map[string][string]`	LD
`ServiceValue`	`string -> string`	LD
`Session`	`() -> Map[string][string]`	N
`SessionValue`	`string -> string`	N
`Form`	`() -> Map[string][string]`	LD
`FormValue`	`string -> string`	LD
`httpHost`	`() -> string`	LD
`httpScript`	`() -> string`	LD
`httpPathInfo`	`() -> string`	LD

i funkcije za manipulisanje skupovima parametara (2.6.2 Parametri okruženja, strana 45.):

Funkcija	Tip	Vrsta
`UpdateValueSet`	`Map[string][string] * string * string -> Map[string][string]`	D
`JoinValueSets`[43]	`Map[string][string] * Map[string][string] -> Map[string][string]`	D
`EncodeUrlQuery`	`Map[string][string] -> string`	D

Biblioteka Dialog

Standardna biblioteka `Dialog` sadrži funkcije za podršku dijaloga. Navedene funkcije su opisane u odeljku *2.6.4 Dijalozi* na strani 51.

Funkcija	Tip	Vrsta
`ask`	`string -> Map[string][string]`	N
`answerAction`	`() -> string`	LD/N[44]

[43] Funkcija nije implementirana.

[44] Funkcija `answerAction` izračunava istu vrednost tokom dela izračunavanja jedne instance programa koji ne obuhvata izračunavanje funkcije `ask`. U delovima programa čije je izračunavanje

Funkcija	Tip	Vrsta
`input`[45]	`string -> string`	N

Biblioteka Report

Standardna biblioteka `Report` sadrži funkcije za formiranje izveštaja od liste dobijene postavljanjem upita nad bazom podataka ili neke druge liste odgovarajućeg tipa. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
`formatRows`	`List['a] * ('a->string) -> string`
`formatRowsSep`	`List['a] * ('a->string) * string -> string`
`formatReport`	`List['a] * string * ('a->string) * string * string -> string`
`formatSubReport`	`List['a] * string * ('a->string) * string * string * int * int -> string`

Navedene funkcije su detaljnije opisane u odeljku *2.7.3 Generisanje izveštaja* na strani 59.

Biblioteka NavigationReport

Standardna biblioteka `NavigationReport` sadrži funkcije za formiranje delova izveštaja sa linijom za navigaciju. Sve funkcije ove biblioteke su determinističke.

Funkcija	Tip
`navigationBar`	`int * List['a] * int * int * string * string * (int->string) * int -> string`
`defaultNavigationBar`	`int * List['a] * int * int -> string`
`simpleNavigationBar`	`int * List['a] * int * int * string * string -> string`
`formatReportWithNavBar`	`int * List['a] * (int*List['a]*int*int -> string) * ('a->string) * (int*List['a]*int*int -> string) * string * int * int -> string`
`formatReportWithHeaderNavBar`	`int * List['a] * (int*List['a]*int*int -> string) * ('a->string) * string * string * int * int -> string`
`formatReportWithFooterNavBar`	`int * List['a]`

vremenski razdvojeno izračunavanjem funkcije `ask` vrednosti funkcije `answerAction` se moraju razlikovati.

[45] Funkcija nije implementirana.

Funkcija	Tip
	* string * ('a->string) * (int*List['a]*int*int -> string) * string * int * int -> string

Navedene funkcije su detaljnije opisane u odeljku *2.7.3 Generisanje izveštaja* na strani 59.

Biblioteka DynamicSQL[46]

Standardna biblioteka `DynamicSQL` sadrži funkcije za podršku dinamičkog SQL-a.

Funkcija	Tip	Vrsta
`dynamicSQLQuery`	`string * map[string][string] -> List[Map[string][string]]`	N
`dynamicSQLCommand`	`string * map[string][string] -> bool`	A

Navedene funkcije su opisane u odeljku *2.7.5 Dinamički SQL* na strani 71.

Biblioteka ExternalObject[47]

Standardna biblioteka `ExternalObject` sadrži klasu `ExternalObject` za podršku komunikacije sa objektima srednjeg sloja aplikacije.

Funkcija	Tip	Vrsta
`Create`	`string * List[string] -> ExternalObject`	A
`Get`	`string * List[string] -> ExternalObject`	N
`Refer`	`string -> ExternalObject`	N
`Evaluate`	`(ExternalObject) * string * List[string] -> string`	A
`Execute`	`(ExternalObject) * string * List[string] -> bool`	A
`GetProperty`	`(ExternalObject) * string -> string`	N
`SetProperty`	`(ExternalObject) * string * string -> bool`	A

Metodi klase `ExternalObject` su detaljnije opisani u odeljku *2.8 Upotreba spoljnih objekata* na strani 73.

Biblioteka MimeTypes

Standardna biblioteka `MimeTypes` sadrži klasu `MimeResource` za podršku operacija sa MIME resursima. Svi metodi ove klase su deterministički.

Funkcija	Tip
`MimeResource`[48]	`string * string -> MimeResource`
`Type`[49]	`(MimeResource) -> string`

[46] Biblioteka nije implementirana.

[47] Biblioteka nije implementirana.

[48] Implementirana je ekvivalentna funkcija `mimeResource`.

Funkcija	Tip
`Content`[50]	`(MimeResource) -> string`

Namena biblioteke MimeTypes je opisana u odeljku *2.6.5 MIME resursi* na strani 55.

Biblioteka Transactions[51]

Standardna biblioteka `Transactions` sadrži funkciju za izračunavanje izuzetaka do kojih je došlo tokom neuspele transakcije, kao što je opisano u odeljku *2.7.6 Izuzeci pri radu sa bazama podataka* na strani 72.

Funkcija	Tip	Vrsta
`transactionExceptions`	`() -> List[Exception]`	`N`

[49] Implementirana je ekvivalentna funkcija `mimeType`.

[50] Implementirana je ekvivalentna funkcija `mimeContent`.

[51] Biblioteka nije implementirana.

Dodatak C - Imenovanje modula

Radi lakšeg snalaženja u kodu programa definiše se uputstvo za imenovanje modula, tj. datoteka koje čine WAFL program. Imenovanje modula prema priloženom uputstvu se preporučuje, ali nije obavezno.

Označavanje tipa modula se izvodi njegovim imenovanjem u obliku: `<naziv>.<nastavak>`, gde je `<naziv>` proizvoljna niska dopuštena od strane operativnog sistema, a `<nastavak>` se određuje prema priloženoj tabeli preporučenih nastavaka naziva modula.

Nastavak	Sadržaj modula
`wafl`	tekst programa
`wpage`	tekst programa koji generiše stranicu (može i `wafl`)
`wdlg`	tekst programa koji upravlja diajlogom (može i `wafl`)
`wfnc`	funkcijska datoteka
`wlib`	biblioteka funkcija
`whtpl`	HTML šablon
`whtlib`	bibilioteka HTML šablona
`wclass`	definicija klase
`wclslib`	biblioteka klasa

Tabela 31: Preporučeni nastavci naziva modula

Dodatak D - Parametri servisa

Konfigurisanje servisa se izvodi pomoću inicijalizacione datoteke servisa. Ona sadrži skup imenovanih vrednosti koje određuju konfiguraciju servisa. Može sadržati komentare zapisane u skladu sa sintaksom programskog jezika WAFL. Preporučuje se da inicijalizaciona datoteka servisa ima nastavak `wsvc`. [52]

Parametri servisa imaju jedinstveno ime i vrednost tipa `string`. Zapisuju se u obliku:

```
<ime> = <vrednost>
```

pri čemu `<ime>` može biti bilo koje ispravno ime programskog jezika WAFL, a `<vrednost>` bilo koja niska zapisana po pravilima programskog jezika WAFL. Ovde definišemo samo nekoliko parametara servisa koje moraju podržati sve implementacije. Implementacije po potrebi mogu dodavati nove parametre. Preporučuje se da imena parametara koji se uvode konkretnim implementacijama počinju znakom podvlačenja '_'.

Naziv	**Vrednost**
dbAlias	Lokalni naziv baze podataka. U slučaju ODBC-a upotrebljava se DSN. Parametar je obavezan ako se upotrebljava baza podataka.
dbUser	Korisničko ime pod čijim se nalogom pristupa bazi podataka. Parametar je obavezan ako se upotrebljava baza podataka.
dbPassword	Lozinka za pristup bazi podataka pod nalogom navedenim u *dbUser*. Parametar je obavezan ako se upotrebljava baza podataka.
dbKeepSessions	Najveći dopušten broj neaktivnih konekcija koje se čuvaju radi uštede na ponovnom uspostavljanju konekcija. Parametar nije obavezan. Podrazumevana vrednost je "0" – bez ograničenja.
dbQueryIsolation	Podrazumevani nivo izolovanosti upita. Dopuštene vrednosti su `s` (*Serializable*), `rr` (*Repeatable Read*), `rc` (*Read Commited*) i `ru` (*Read Uncommited*). Parametar nije obavezan. Podrazumevana vrednost je `ru`.
dbTransIsolation	Podrazumevani nivo izolovanosti transakcija. Dopuštene vrednosti su `s` (*Seria-*

[52] Implementirani su odgovarajući parametri servisa, ali se upotrebljavaju na izmenjen način. Opis sledi u posebnom dokumentu.

Naziv	Vrednost
	lizable), `rr` (*Repeatable Read*), `rc` (*Read Commited*) i `ru` (*Read Uncommited*). Parametar nije obavezan. Podrazumevana vrednost je `rr`.
defaultLib	Naziv datoteke sa spiskom naziva bibliotečkih modula čije se definicije uvode u prostor imena svih modula servisa koji nisu na spisku. Nazivi modula se navode po jedan u redu. Datoteka može sadržati komentare u stilu programskog jezika WAFL. Parametar je obavezan.
defaultMimeType	Podrazumevani MIME tip resursa koji se vraća kao rezultat. Upotrebljava se za dopunjavanje rezultata zaglavljem sa oznakom MIME tipa resursa, u slučaju kada rezultat nije objekat klase MimeResource. Parametar nije obavezan. Ukoliko se izostavi podrazumeva se vrednost "`text/plain`".
defaultWrapper	Naziv datoteke sa funkcijom (ili HTML šablonom) koja generiše podrazumevani okvir stranice. Funkcija mora imati tip `string -> string` i upotrebljava se za kompletiranje rezultata u slučaju kada rezultat rada programa nije ni MIME resurs ni niska koja počinje tagom `<HTML>`, a podrazumevani MIME tip je "`text/html`". Ako rezultat programa nije potrebno dopunjavati, to se označava direktivom `#nowrap` na samom početku programa. Parametar nije obavezan. Ukoliko se ne navede, neće se dopunjavati nekompletni rezultati.
defaultLibDir	Direktorijum u kome se traže biblioteke uz čiji naziv nije navedena putanja. Parametar je obavezan.
defaultFncDir	Direktorijum u kome se traže funkcijske datoteke. Parametar nije obavezan. Ako se ne navede upotrebljava se vrednost parametra `defaultLibDir`.
defaultTplDir	Direktorijum u kome se traže šabloni i biblioteke šabona. Parametar nije obavezan. Ako se ne navede upotrebljava se vrednost parametra `defaultLibDir`.
defaultClassDir	Direktorijum u kome se traže moduli sa definicijama klasa i biblioteka klasa. Parametar nije obavezan. Ako se ne navede upotrebljava se vrednost parametra `defaultLibDir`.
dlgResponseTime	Trajanje intervala (u sekundama) u kome mora stići odgovor na pitanje postavljeno u dijalogu, tj. vreme u kome mora biti dovršeno izračunavanje funkcije `ask`. Parametar nije obavezan. Ako se navede, njegova vrednost mora biti pozitivna. Podrazumevana vrednost je 180.

Tabela 32: Parametri servisa

Kako inicijalizaciona datoteka može sadržati i neke poverljive informacije, poput naloga i lozinke za pristupanje bazi podataka, preporučuje se oprezno davanje prava za pristupanje inicijalizacionoj datoteci.

Dodatak E - Tabele iz primera

U primerima u ovom radu je upotrebljavana baza podataka informacionog sistema fakulteta. Ovde je priložen pojednostavljen opis upotrebljavanih tabela, bez namere da se izlaže opis kompletnog informacionog sistema. Opisi tabela su zapisani u obliku SQL naredbi za njihovo pravljenje.

Tabela `Profil` sadrži podatke o profilima studija na fakultetu:

```
create table profil (
    statg       smallint not null,  --  godina donošenja statuta
    sifprof     char(5)  not null,  --  šifra profila
    naziv       char(20) not null,  --  naziv profila
    primary key (statg,sifprof)
    )
```

Tabela `Plans` sadrži podatke o predmetima koji ulaze u plan studija na profilima koji postoje na fakultetu:

```
create table plans (
    statg       smallint not null,  --  godina donošenja statuta
    sifprof     char(5)  not null,  --  šifra profila
    sifpred     char(5)  not null,  --  šifra predmeta
    naziv       char(20) not null,  --  naziv predmeta
    primary key (statg,sifprof,sifpred),
    foreign key (statg,sifprof)
        references profil on delete restrict
    )
```

Tabela `Student` sadrži podatke o studentima upisanim na fakultet:

```
create table student (
    indeks      smallint not null,  --  broj indeksa
    god         smallint not null,  --  godina upisa na fakultet
    ime         char(20) not null,  --  ime studenta
    prezime     char(20) not null,  --  prezime studenta
    sifprof     char(5)  not null,  --  šifra profila
    statg       smallint not null,  --  statut po kome je upisan
    email       char(50),           --  email adresa
    primary key (indeks, god),
    foreign key (statg, sifprof)
        references profil on delete restrict
    )
```

Tabela `Prijavio` sadrži podatke o prijavljenim ispitima:

```
create table prijavio (
    indeks      smallint not null,  --  indeks studenta
    god         smallint not null,  --  godina upisa studenta
    sifpred     char(5)  not null,  --  šifra predmeta
    rok         char(3)  not null,  --  ispitni rok
    irgod       smallint not null,  --  godina
    priput      smallint not null,  --  po koji put se prijavljuje
    primary key (indeks,god,sifpred,priput),
    foreign key (indeks,god)
        references student on delete cascade
    )
```

Tabela `Polagao` sadrži podatke o polaganim ispitima:

```
create table polagao (
    indeks      smallint not null,  --  indeks studenta
    god         smallint not null,  --  godina upisa studenta
    sifpred     char(5)  not null,  --  šifra predmeta
    polput      smallint not null,  --  po koji put se polaže
    ocenap      smallint not null,  --  ocena na pismenom delu i.
    ocenau      smallint,           --  ocena na usmenom delu isp.
    ocena       smallint,           --  ukupna ocena
    primary key (indeks,god,sifpred,polput),
    foreign key (indeks,god)
        references student on delete cascade,
    foreign key (indeks,god,sifpred,polput)
        references prijavio on delete restrict
    )
```

Tabela `Aktuelno` sadrži podatke o ispitnom roku za koji je u toku prijavljivanje ispita:

```
create table aktuelno (
    rok         char(3)  not null,  --  ispitni rok
    irgod       smallint not null,  --  godina
    primary key (rok,irgod)
    )
```

Rečnik pojmova

A

aplet - Aplikativni program na programskom jeziku Java koji se preuzima od Veb servera i izvršava na programu za pregledanje Veba.

aplikacija - Skup softverskih komponenti koji se upotrebljava za izvođenje nekog korisnički orijentisanog posla primenom računara.

aplikativni programski interfejs - Softverska komponenta nekog sistema čijom se upotrebom u aplikacijama omogućava upotreba tog sistema.

aplikativni server - Komponenta sistema sa arhitekturom na više nivoa, ili sistema za distribuirano izračunavanje, koja upravlja izvršavanjem ili predstavlja podršku izvršavanju aplikacija.

aplikativni SQL - Standardizovan način upotrebe upitnog jezika SQL u aplikacijama na drugim programskim jezicima.

arhitektura - Arhitektura predstavlja konceptualni nivo razvoja aplikacije.

ASP - Tehnologija kompanije Microsoft za razvoj Veb aplikacija i lokacija. Akronim od *Active Server Pages*.

B

BDE - Tehnologija kompanije Borland koja predstavlja aplikativni interfejs za pristup podacima pohranjenim na većem broju podržanih SUBP. Akronim od *Borland Data Engine*.

<bigwig> - Programski jezik za razvoj Veb aplikacija zasnovan na upravljanju sesijama.

bočni efekat - Pojava da izračunavanje nekog izraza uzrokuje promene u okruženju u kome se taj izraz izračunava. Programiranje na imperativnim programskim jezicima počiva na upotrebi bočnih efekata. Na deklarativnim programskim jezicima je moguće programiranje bez bočnih efekata.

broj porta - Sastavni deo kolekcije Internet protokola. Identifikator logičke veze između aplikacije i transportnog servisa.

C

CFML - Proširenje HTML-a koje omogućava razvoj Veb aplikacija putem šablona. Sastavni deo tehnologije *Cold Fusion*. Akronim od *ColdFusion Markup Language*.

CGI - Standardni interfejs za razmenu informacija između Veb servera i programa (skriptova) koji predstavljaju elemente Veb aplikacije. Akronim od *Common Gateway Interface*.

čisto funkcionalni program - Program napisan uz strogo pridržavanje principa funkcionalnog programiranja.

CLI - Standardizovan aplikativni interfejs za upotrebu relacionih baza podataka u aplikacijama. Specifikacija grupe *X/Open SQL Access Group*. Akronim od *Client Level Interface*.

ColdFusion - Tehnologija kompanije *Allaire* za razvoj Veb aplikacija putem šablona i skriptova umetnutih u šablone.

COM - Tehnologija kompanije Microsoft za razvoj distibuiranih aplikacija putem distribuiranih objekata. Akronim od *Component Object Model*.

CORBA - Tehnologija za razvoj distribuiranih aplikacija putem distribuiranih objekata. Akronim od *Common Object Request Broker Architecture*. Razvijena od strane OMG (*Object Management Group*).

D

deklarativni programski jezici - Programski jezici na kojima se problemi rešavaju njihovim preciznim opisivanjem, a ne opisivanjem postupaka za njihovo rešavanje. Dele se na logičke i funkcionalne programske jezike. (*videti* funkcionalni programski jezici, logički programski jezici, imperativni funkcionalni jezici)

deljenje podataka - Neki od postupaka za rešavanje problema kada više programa istovremeno upotrebljava neke podatke. Osnovni aspekti problema su očuvanje konzistentnosti podataka i svakog od programa.

dijalog - (1) Razmena informacija između dva subjekta. (2) U računarstvu predstavlja razmenu informacija između korisnika i sistema sa kojim sistem komunicira. (3) Dijalog u programskom jeziku WAFL predstavlja segment Veb sesije čijim se tokom programski upravlja (za razliku od uobičajenog upravljanja tokom sesije od strane korisnika).

dinamička Veb stranica - Stranica Veb lokacije čiji sadržaj nije nepromenljiv već se prilagođava zahtevu korisnika i/ili drugim okolnostima.

dinamički SQL - Način izvršavanja SQL naredbi koji počiva na njihovom generisanju ili pripremanju u vreme izvršavanja programa. Upotrebljava se u prilikama kada u fazi pisanja programa nije poznat SQL iskaz koji je potrebno izvršiti.

dinamičko vezivanje metoda - način povezivanja imena metoda sa njegovom definicijom koji omogućava da se u fazi izračunavanja (ili izvršavanja) programa odabire odgovarajuća definicija metoda na osnovu stvarnog tipa objekta za koji se metod poziva.

distribuirani sistem za upravljanje datotekama - Sistem za upravljanje datotekama koje su pohranjene na više fizički razdvojenih lokacija.

DTD - Skup pravila koja određuju strukturu konkretne klase SGML ili XML dokumenata. Akronim od *Document Type Definition*.

dugme - Element grafičkog korisničkog interfejsa čijim se "pritiskom" (tj. izborom mišem) inicira neka aktivnost.

DŽ

džep - Prostor u koji se nešto umeće. U programskom jeziku <bigwig> džepovi predstavljaju prostor u koji se umeću vrednosti parametara.

F

funkcija višeg reda - Funkcija koja kao svoje argumente ili rezultat ima neke funkcije. Jedan od osnovnih principa funkcionalnih programskih jezika je mogućnost upotrebe funkcija višeg reda.

funkcionalni programski jezici - Deklarativni programski jezici u kojima se problem opisuje definisanjem funkcionalnih zavisnosti argumenata i rezultata. U radu su opisani funkcionalni programski jezici ML i Haskell. Programski jezik WAFL je funkcionalni programski jezik. (*videti* deklarativni programski jezici, imperativni programski jezici)

G

generator dinamičke stranice - Program ili neko drugo sredstvo za generisanje sadržaja dinamičkih Veb stranica.

generički tip podataka - tip podataka čija definicija zapravo predstavlja definiciju jedne klase (grupe) srodnih tipova podataka.

glavna stranica - (*videti* uvodna stranica)

GIF - Jedan od najzastupljenijih digitalnih formata za zapisivanje slika. Predstavlja jedan od dva osnovna oblika digitalnih slika na Vebu. Akronim od *Graphical Interchange Format*. (*videti* JPEG)

G mašina - Posebna vrsta graf redukcione mašine.

graf redukciona mašina - Mašina koja izvršava redukciju grafa funkcionalnog izraza.

grafički korisnički interfejs - Interfejs za komunikaciju korisnika i programa zasnovan na grafičkom predstavljanju zadataka i rezultata rada.

grafičko radno okruženje - Okruženje u kome korisnik upravlja nekim alatom ili radom nekog sistema posredstvom grafičkog korisničkog interfejsa.

Guide - Sistem za upravljanje dinamičkim lokacijama koji na osnovu skriptova upravlja generisanjem dinamičkih stranica iz šablona.

H

Haskell - Jedan od najzastupljenijih savremenih funkcionalnih programskih jezika.

hipertekst - Način predstavljanja informacija pomoću aktivnih veza između delova informacija. Čitalac (korisnik) može upravljati redosledom pregledanja informacija.

hipertekstualna veza - Element hiperteksta koji predstavlja aktivnu vezu sa drugom informacijom. Čitalac (korisnik) može upotrebiti hipertekstualnu vezu da bi pristupio odgovarajućoj informaciji.

host promenljiva - Ime koje se pojavljuje u okviru upita (ili naredbe) na SQL-u, a odnosi se na promenljivu jezika u koji je upit (ili naredba) ugnežden.

HTML - Jezik za opis dokumenata definisan prvenstveno kao podrška hiperteksta na računarskim sistemima. Jedna od osnovnih tehnologija kojima je definisan Veb. Akronim od *Hypertext Markup Language*.

HTML formular - Element HTML dokumenta koji korisniku (čitaocu) omogućava da upiše podatke i prosledi ih Veb lokaciji (aplikaciji).

HTML oznaka - Osnovna sintaksna konstrukcija jezika HTML koja se upotrebljava za određivanje funkcije ili načina vizualnog predstavljanja elemenata HTML dokumenta.

HTTP - Protokol koji se upotrebljava na Internetu za razmenu i prikazivanje hiperteksta. Jedna od osnovnih tehnologija kojima je definisan Veb.

I

ime računara - Naziv po kome se računar raspoznaje u lokalnoj mreži ili na Internetu.

imperativni programski jezici - Programski jezici na kojima se problemi rešavaju opisivanjem postupaka za njihovo rešavanje. Tipični predstavnici imperativnih programskih jezika su *C*, *C++* i *Pascal*. (*videti* deklarativni programski jezici, funkcionalni programski jezici)

interaktivna komunikacija sa korisnikom - Komunikacija koju program ostvaruje sa korisnikom tokom izvršavanja (ili izračunavanja) u cilju obezbeđivanja informacija neophodnih za dobijanje rezultata.

Internet - Globalna mreža međusobno povezanih računarskih mreža koje razmenjuju informacije primenom kolekcije tzv. Internet protokola i dopuštaju javni pristup.

IP - Sastavni deo kolekcije Internet protokola. Protokol koji usmerava podatke kroz mrežu ili međusobno povezane mreže i služi kao posrednik između protokola viših slojeva mreže i fizičke mreže. Akronim od *Internet Protocol*.

IP broj - Jedinstvena 32-bitna adresa koja određuje položaj svakog računara ili drugog uređaja na Internetu.

ISO - Međunarodna organizacija za ustanovljavanje standarda i podršku njihovog razvoja. Akronim od *International Organization for Standardization*.

izuzetak - Neuobičajen uslov detektovan tokom izvršavanja programa. Obično predstavlja neposrednu ili posrednu posledicu neispravnosti u ulaznim podacima ili samom programu.

J

Java - Objektno orijentisan programski jezik koji se, po definiciji, prevodi na prenosivi interpretativni kod koji omogućava interakciju među međusobno udaljenim objektima. Razvijen je od strane kompanije *Sun Microsystems Inc.*

JavaScript - Programski jezik za pisanje skriptova koji podseća na programski jezik Java.

JDBC - Aplikativni programski interfejs veoma sličnih karakteristika kao ODBC, ali posebno razvijen za upotrebu u aplikacijama na programskom jeziku Java. Akronim od *Java Database Connectivity*.

JPEG - Standardizovan format za zapisivanje komprimovavnih slika sa velikim brojem boja. Predstavlja jedan od dva (*videti* GIF) osnovna tipa grafičkih dokumenata na Vebu. Akronim od *Joint Photographic Experts Group*, što je naziv komiteta koji je ustanovio ovaj standardni format.

JVM - Softverska implementacija procesora koji izvršava prevedeni Java kod. Akronim od *Java Virtual Machine*.

K

klijent - Računarski sistem ili program koji zahteva usluge od drugog računarskog sistema ili programa, koji se naziva *server*. Više klijenata može koristiti usluge istog servera. (*videti* klijent/server, server)

klijent/server - Model komunikacije u distribuiranoj obradi podataka po kome jedan program šalje zahteve drugom programu i čeka na odgovor. Programi mogu biti izvršavani na različitim fizičkim lokacijama. (*videti* klijent, server)

kolačić - Informacija koju Veb server pohranjuje na strani korisnika prilikom posećivanja neke Veb lokacije. Upotrebljava se za praćenje specifičnosti korisnika i podataka koje je korisnik naveo pri pregledanju lokacije. Upotreba kolačića omogućava unapređivanje interaktivnosti Veb lokacije.

komandna linija - (1) Linija na korisnikovom ekranu u kojoj se upisuje tekst komande. (2) Parametar pri pokretanju programa koji sadrži tekst komande kojom je program pokrenut.

konkurentnost - (*videti* paralelizam)

konkurentno izvršavanje - (*videti* paralelno izvršavanje)

L

lenjo izračunavanje - Izračunavanje vrednosti izraza primenom normalnog redosleda redukcije. Obično počiva na primeni lenjih konstruktora i deljenju podataka u okolini.

logički programski jezici - Deklarativni programski jezici u kojima se problem opisuje definisanjem odnosa objekata koji učestvuju u obradi. Najvažniji predstavnik je programski jezik *Prolog*. (*videti* deklarativni programski jezici, imperativni programski jezici)

M

makro - (*videti* makroinstrukcija)

makroinstrukcija - Konstrukcija programskog jezika ili jezika za opis podataka koja se zamenjuje definisanom sekvencom konstrukcija istog jezika. Može imati parametre koji utiču na rezultujuću sekvencu.

Mawl - Programski jezik za razvoj Veb lokacija putem opisivanja sesija.

MHTML - Proširenje HTML-a koje omogućava definisanje HTML dokumenata u obliku interfejsa za programe pisane na programskom jeziku Mawl.

MIME - Standard za identifikovanje tipa objekta koji se prenosi putem Interneta. Akronim od *Multipurpose Internet Mail Extension*.

ML - Jedan od najzastupljenih savremenih funkcionalnih programskih jezika.

N

nivo izolovanosti - Stepen izolovanosti procesa koji upotrebljava bazu podataka od drugih procesa koji upotrebljavaju istu bazu podataka.

O

ODBC - Standardni aplikativni programski interfejs za pristupanje podacima kako u relacionim tako i u ne-relacionim bazama podataka. Zasnovan je na CLI. Razvijen je od strane kompanija *DEC*, *Lotus*, *Microsoft* i *Sybase*. Akronim za *Open Database Connectivity*. (*videti* CLI)

oslonac - (*videti* hipertekstualna veza)

oznaka - (*videti* HTML oznaka)

P

paralelizam - Istovremeno korišćenje resursa (uređaja, datoteka, baze podataka i drugih) od strane više subjekata (programa, procesa, uređaja i drugo).

paralelno izvršavanje - Paralelizam pri izvršavanju više programa ili procesa.

početna stranica - (*videti* uvodna stranica)

polje za označavanje - Element grafičkog korisničkog interfejsa, koji može da se označi ili ne označi.

poseta - (1) Na Vebu, pregledanje jedne stranice Veb lokacije od strane posetioca. (2) U programskom jeziku WAFL, poseta je prost segment sesije, tj. par (zahtev, odgovor) koji ne pripada nijednom dijalogu. (*videti* sesija, posetilac Veb lokacije, dijalog)

posetilac Veb lokacije - Subjekat koji Veb serveru isporučuje zahteve i preuzima stranice lokacije koje predstavljaju odgovore na te zahteve.

produžena putanja - Deo URL putem koga se mogu prenositi parametri generatorima dinamičkih stranica.

program - (1) Niz instrukcija predviđenih za izvršavanje od strane računara. (2) Algoritam zapisan na nekom programskom jeziku.

programska okolina - Okruženje u kome se izvršava program. Putem parametara programske okoline programima se mogu saopštavati uslovi u kojima se izvršavaju.

programski jezik - Jezik za zapisivanje programa.

proste stranice - Dinamičke stranice Veb lokacije pri čijem generisanju ne dolazi do bočnih efekata. (*videti* bočni efekat)

protokol - Pravila koja određuju redosled razmenjivanja zahteva i odgovora i njihova značenja,

koja se koriste pri komunikaciji, razmeni podataka i sinhronizovanju stanja učesnika u komunikaciji.

prozor - Područje na ekranu u kome se ostvaruje prikaz koji se odnosi na neku aplikaciju. Različitim aplikacijama može odgovarati više prozora na ekranu.

putanja - (1) Kod sistema datoteka, jedan ili više naziva direktorijuma uz koje stoji naziv jednog objekta. (2) Redosled obilaženja čvorova pri prenošenju podataka kroz mrežu.

Python - Objektno orijentisani programski jezik nove generacije razvijen za pisanje skriptova i aplikacija.

R

radio polje - Poseban oblik polja za označavanje. Navodi se uvek u grupi u kojoj najviše jedno radio polje može biti označeno. (*videti* polje za označavanje)

razvojno okruženje - Okruženje u kome se izvodi razvoj aplikacije. Često se razlikuje od okruženja u kome se aplikacija izvršava.

robusnost programa - Sposobnost programa da odreaguje na neispravne ulazne podatke i neispravnosti u izvršnom okruženju.

RSUBP - Sistem za upravljanje bazama podataka koji počiva na upotrebi relacionog modela podataka. Akronim od *Relacioni sistem za upravljanje bazama podataka*.

S

SECD mašina - Mašina za izvršavanje funkcionalnih programa. Ime je dobijeno izdvajanjem prvih slova četiri osnovna elementa mašine: *Stack*, *Environment*, *Control* i *Dump*.

server - Funkcionalna jedinica (računar, program i sl.) koja pruža usluge jednom ili većem broju klijenata. (*videti* klijent/server, klijent)

server za datoteke - Računar ili uređaj velikog kapaciteta za pohranjivanje datoteka koji većem broju drugih uređaja ili računara pruža usluge pristupanja datotekama.

serverske oznake - Posebne oznake u proširenom HTML-u koje Veb server zamenjuje odgovarajućim sadržajem. (*videti* HTML oznaka)

servisna logika - Skup pravila po kojima se izvodi usluga (servis). Predstavlja srednji sloj pri razvoju servisa.

sesija - (1) U mrežnim arhitekturama, sve aktivnosti koje se odvijaju radi ostvarivanja komunikacije između funkcionalnih jedinica: uspostavljanje veze, održavanja veze i raskid veze. (2) Logička veza između dve jedinice na mreži koja može biti aktivirana, upravljana i deaktivirana, već prema zahtevu. Svaka sesija je jedinstveno identifikovana u okviru svih pojedinačnih razmena informacija koje se odvijaju tokom sesije. (3) Na Vebu, skup svih poseta jedne Veb lokacije od strane jednog posetioca u nekom vremenskom intervalu.

SGML - Standardizovan metajezik za definisanje jezika za opis dokumenata. Počiva na strukturiranju sadržanih informacija, a ne njihovog prikaza. Akronim od *Standard Generalized Markup Language*.

sidrište - (*videti* hipertekstualna veza)

sistemi za rad sa distribuiranim objektima - Sistemi koji omogućavaju razvoj distribuiranih aplikacija obezbeđivanjem međusobne komunikacije objekata koji su locirani na različitim računarima u mreži.

skript - Računarski program koji se interpretira.

SQL - Standardizovan upitni jezik za relacione SUBP-e. Akronim od *Struktuirani upitni jezik* (engl. *Structured Query Language*).

srednji sloj - Kod višeslojne arhitekture aplikacije, sloj koji nema neposrednog dodira ni sa korisnikom ni sa resursima.

standardni izlaz - Kod programskih jezika, podrazumevani resurs putem koga se isporučuju izlazni podaci. Po pravilu se određuje pri pokretanju programa od strane korisnika programa ili operativnog sistema.

standardni ulaz - Kod programskih jezika, podrazumevani resurs putem koga se preuzimaju ulazni podaci. Po pravilu se određuje pri pokretanju programa od strane korisnika programa ili operativnog sistema.

stanje sesije - Na Vebu, skup parametara koji opisuju realizovani tok sesije i podatke koje je korisnik saopštio Veb servisu.

statička Veb stranica - Veb stranica čiji se sadržaj ne menja programski.

stranice sa bočnim efektima - Dinamičke stranice Veb lokacije pri čijem generisanju dolazi do bočnih efekata. (*videti* bočni efekat)

STG mašina - Vrsta G-mašine bez kičme i oznaka tipova čvorova.

SUBP - Akronim od *Sstem za upravljanje bazama podataka*.

Š

šablon stranice - Uzorak po kome se generiše dinamička Veb stranica. Obično ima parametre koji utiču na rezultat.

T

TCP/IP - Protokol za kontrolu prenosa podataka i Internet protokol, koji zajedno predstavljaju pouzdanu vezu između aplikacija kroz međusobno povezane raznorodne mreže. Ujedno i kolekcija transportnih i aplikativnih protokola na Internetu. Akronim od *Protokol za kontrolu slanja / Internet protokol* (engl. *Transmission Control Protocol/Internet Protocol*).

trajni podaci - Podaci, programi i drugi oblici informacija čije trajanje može da se nastavi i po okončanju rada programa koji im pristupa.

transakcija - Logička jedinica posla pri radu sa podacima.

U

ugneždeni SQL - (*videti* aplikativni SQL)

umetnuti skript - Skript koji je umetnut u neki program ili dokument. U kontekstu generisanja dinamičkih Veb stranica, skript je obično umetnut u šablon stranice.

UNIX-oliki operativni sistemi - Operativni sistemi oblikovani po uzoru na operativni sistem UNIX.

upit - (1) Naredba upitnog jezika kojom se zahtevaju određeni podaci. (2) Deo URL-a koji se upotrebljava za prenošenje parametara generatoru dinamičke Veb stranice.

upravljanje sesijom - Upravljanje tokom interaktivne komunikacije sa posetiocem (korisnikom) Veb lokacije.

uvodna stranica - Stranica koja je zamišljena kao polazna tačka pri posećivanju Veb lokacije.

V

VBScript - Verzija programskog jezika Visual Basic prilagođena za razvoj dinamičkih Veb lokacija. Upotrebljava se u okviru tehnologije ASP. (*videti* ASP)

Veb - Mreža servera koji sadrže programe i podatke. Znatan deo podataka je u obliku hiperteksta.

Veb aplikacija - Aplikacija čiji korisnički interfejs počiva na upotrebi Veba. Jezgro aplikacije predstavlja nadgradnju Veb servera, a korisnički interfejs se sastoji od Veb stranica koje se korisniku predstavljaju posredstvom programa za pregledanje Veba.

Veb klijent - Program koji koristi usluge Veb servera.

Veb korisnik - Korisnik Veb aplikacije ili posetilac Veb lokacije.

Veb lokacija - Veb server kojim se upravlja od strane jednog subjekta i sadrži informacije u hipertekstualnom obliku.

Veb server - Server koji je spojen na Internet i pruža usluge isporučivanja Veb stranica.

Veb servis - (*videti* Veb usluga)

Veb sesija - Logička veza između Veb klijenta i Veb servera koja ima početak, tok i kraj. (*videti* sesija)

Veb stranica - Bilo koji dokument koji je pristupačan na Vebu posredstvom URL-a.

Veb usluga - Usluga koja se isporučuje posredstvom Veba.

veza - (*videti* hipertekstualna veza)

vredno izračunavanje - Izračunavanje vrednosti izraza primenom aplikativnog redosleda redukcije.

W

WWW - Akronim od *World Wide Web*. (*videti* Veb)

X

XML - Standardni metajezik za definisanje jezika za opis dokumenata koji je izveden iz SGML-a i predstavlja njegov podskup. Akronim od *Extensible Markup Language*.

Indeks

Literatura

[Alla:1999] **Developing Web Applications with ColdFusion**, *Allaire Corporation*, 1999.

[Arno1996] Ken Arnold, James Gosling, **The Java Programming Language**, *Java Series, Sun Microsystems*, 1996. ISBN 0-201-63455-4

[Bakk2000] Stig Sather Bakken et.al, **PHP Manual**, *PHP Documentation Group*, 2000.

[Bell:1998] **Mawl 2.1 Tutorial**, *Mawl Software Distribution*, *Bell Laboratories*, *Lucent Technologies*, 1998.

[Booth1996] Simon P. Booth, Simon B. Jones, **Are Ours Really Smaller Than Theirs?**, u Phil Trinder, ed., **Glasgow Workshop on Functional Programming**, *Ullapool, Scotland, July 1996. Department of Computing Science, University of Glasgow*, 1996.

[Brab2000] Claus Brabrand, Anders Mller, Anders Sandholm, Michael I. Schwartzbach, **Designing a language for developing interactive Web services**, 2000. (*http://www.brics.dk/bigwig/research/publications/*)

[Brown1996] M. R. Brown, **FastCGI Specification**, *Open Market, Inc., Cambridge*, 1996.

[Card1985] Luca Cardeli, Peter Wegner, **On Understanding Types, Data Abstraction, and Polymorphism**, *Computing Surveys, Vol 17 n.4*, 1985.

[Card1987] Luca Cardelli, **Basic Polymorphic Typechecking**, *Science of Computer Programming, Vol.8, No.2*, 1987.

[Card1991] Luca Cardeli, **Typeful Programming**, u E. J. Neuhold, M. Paul, eds., **Formal Description of Programming Concepts**, *Springer-Verlag*, 1991.

[Coll1994] Graham Collins, **Supporting Formal Reasoning About Standard ML**, Tech Report, *LFCS, University of Edinburgh*, 1994.

[Cous1987] G. Cousineau, P.-L. Curien, M. Mauny, **The Categorical Abstract Machine**, *Science of Computer Programming*, *8(2)*, 1987.

[Date2000] C. J. Date, **An Introduction to Database Systems (7th edition)**, *Addison-Wesley*, 2000.

[Davi1994] Andrew Davison, **Teaching C in a Functional Style**, Technical report 94/1, *University of Melbourne, Department of Computer Science*, 1994.

[Ghez1997] Carlo Ghezzi, Mehdi Jazayeri, **Programming Language Concepts (3rd edition)**, *John Wiley & Sons*, 1997. ISBN 0-201-63455-4

[Gieg1999] Robert Giegerich, Ralf Hinze, Stefan Kurtz, **Straight to the Heart of Computer Science via Functional Programming**, u Matthias Felleisen, Michael Hanus, Simon Thompson, eds., **Proceedings of the Workshop on Functional and Declarative Programming in Education, FDPE'99**, *Paris, France, Technical report of Rice University, Rice COMP TR99-346*, 1999.

[Giesl1995] Jurgen Giesl, **Termination Analysis for Functional Programs using Term Orderings**, *Pr. 2nd Int. Stat. Analysis Symp., LNCS 983*, Glasgow, Scotland, 1995.

[Hami1996] Graham Hamilton, Rick Cattell, **JDBC: A Java SQL API v.1.10**, *JavaSoft, Sun Microsystems, Inc.*, 1996.

[Hass2000] A. E. Hassan, R. C. Holt, **A Reference Architecture for Web Servers**, *WCRE 2000: Working Conference on Reverse Engineering, Brisbane, Australia*, Nov 6, 2000.

[Hein1997] Nick Heinle, **JavaScript – When HTML Is Not Enough**, *WWW Journal, II/2 – "Scripting Languages"*, 1997.

[Hend1980] P. Henderson, **Functional Programming: Application and Implementation**, *Prentice Hall International*, 1980.

[Hudak1989] Paul Hudak, **Conception, Evolution, and Application of Functional Programming Languages**, *ACM Computing Surveys*, Vol.21, No.3, 1989.

[Hudak1992] Paul Hudak, Simon Peyton Jones, Philip Wadler, eds., **Report on the Programming Language Haskell**, *ACM SIGPLAN Notices, vol. 27, no. 5*, 1992.

[Hugh1990] John Hughes, **Why Functional Programming Matters**, u D. Turner, ed., **Research Topics in Functional Programming**, *Addison Wesley*, 1990.

[Gilm1997] S. Gilmore, **Programming in Standard ML'97: A tutorial introduction**, Technical Report ECS-LFCS-97-364, *Laboratory for Foundations of Computer Science*, 1997.

[IBM:1998] **IBM DB2 Universal Database Version 5.2: SQL Reference**, *IBM Corporation*, 1998.

[IBM:1999] **IBM Dictionary of Computing: DRAFT**, *IBM Corporation*, 1999.

[IEEE:754] **IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985**, *Institute of Electrical and Electronics Engineers*, 1985.

[ISO:8879] **ISO 8879:1986, Information processing – Text and office systems – Standard Generalized Markup Language (SGML)**, *ISO* 1986.

[ISO:9075] **ISO/IEC 9075-3:1999 Information technology – Database languages – SQL – Part 3: Call-Level Interface (SQL/CLI)**, *ISO/IEC*, 1999.

[ISO:9636] **ISO/IEC 9636:1991 Information technology – Computer graphics – Interfacing techniques for dialogues with graphical devices (CGI) – Functional specification**, *ISO/IEC*, 1999.

[John1984] Thomas Johnsson, **Efficient Compilation of Lazy Evaluation**, u **Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction**, *SIGPLAN Notices, Vol. 19*, 1984.

[John1998] Ben Johnson, **Perl Tutorial**, *The National Center for Supercomputing Applications, University of Illinois*, 1998.

[Jones1987] Simon L. Peyton Jones, **The Implementation of Functional Programming Languages**, *Prentice Hall*, 1987.

[Jones1992] Simon L. Peyton Jones, **Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine**, *Journal of Functional Programming 2(2)*, April 1992.

[Jones1993] Simon L. Peyton Jones, Philip Wadler, **Imperative Functional Programming**, *ACM Simposium on Principles of Programming Languages (POPL), Charlestown*, 1993.

[Joos1993] Stef Joosten (ed.), Klaas Van Den Berg, Gerrit Van Der Hoeven, **Teaching functional programming to first-year students**, *Journal of Functional Programming 3(1):49-65*, 1993.

[Kels1998] R. Kelsey at all, **Revised Report on the Algorithmic Language Scheme**, *SIGPLAN Notices*, 1998.

[Kern1988] Brian W. Kernighan, Dennis M. Ritchie, **The C Programming Language**, *Prentice-Hall*, 1988.

[Knuth1984] D. E. Knuth, **The TEX Book**, *Addison Wesley, Reading, MA*, 1984.

[Koop1995] P. Koopman, V. Zweije, **Functional programming in a basic database course**, u P. H. Hartel, M. J. Plasmeijer, eds., **Functional programming languages in education**, *FPLE, Nijmegen, The Netherlands, LNCS 1022, Springer-Verlag, Heidelberg*, 1995.

[Lazar1997] Z. Peter Lazar, Peter Holfelder, **Web Database Connectivity with Scripting Languages**, *WWW Journal, II/2 – "Scripting Languages"*, 1997.

[Leij1999] Daan Leijen, Erik Meijer, **Domain Specific Embeded Compilers**, *2nd USENIX Conference on Domain-Specific Languages (DSL), Austin, USA, October*, 1999.

[Levy1998] Michael Levy, **Guide User Manual**, *University of Victoria*, 1998. (*http://www.csr.uvic.ca/~mlevy/Guide/guideman.pdf*)

[Lipp1998] Stanley B. Lippman, Josee Lajoie, **C++ Primer, 3rd Edition**, *Addison Wesley Longman, Inc.*, 1998.

[Malk1999] Saša Malkov, **Infomacioni sistem Morava.Com: Tehnička dokumentacija**, 1999.

[McCa1963] J. McCarthy, **A basis for a mathematical theory of computation**, u P. Braffort, D. Hirschberg, eds., **Computer Programming and Formal Systems**, *North-Holland, Amsterdam*, 1963.

[McCa1978] J. McCarthy, **History of Lisp**, u **Preprints of Proceedings of ACM SIGPLAN History of Programming Languages Conference**, *SIGPLAN Notices, Vol. 13*, 1978.

[Meij2000] Erik Meijer, **Server side web scripting in Haskell**, *Journal of Functional Programming, 10(1):1-18*, 2000.

[Miln1978] R. Milner, **A Theory of Type Polymorphism in Programming**, *Journal of Computer and System Science, Vol.17*, 1978.

[Mitić1995] Nenad Mitić, **Funkcijski interfejs ka relacionim bazama podataka i primene**, doktorska disertacija, *Matematički fakultet Univerziteta u Beogradu*, 1995.

[MS:1999] **Rich Text Format (RTF) Specification, version 1.6**, u **MSDN Library**, *Microsoft Corp.*, 1999. (*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnrtfspec/html/rtfspec.asp*)

[MS:1999a] **Active Server Pages Guide**, u **MSDN Library**, *Microsoft Corp.*, 1999. *(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iisref/html/psdk/asp/aspguide.asp*)

[MS:2000] **ISAPI and the Web Application Architecture**, u **MSDN Library**, *Microsoft Corp.*, 2000. (*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iisref/html/psdk/asp/isgu80v9.asp*)

[NCC:1999] **NSAPI Programmer's Guide for Enterprise Server 4.0**, *Netscape Communications Corp.*, 1999. (*http://docs.iplanet.com/docs/manuals/enterprise/40/nsapi/contents.htm*)

[NCC:1999c] **Persistent Client-State Http Cookies: Preliminary Specification**, *Netscape Communications Corp.*, 1999. (*http://home.netscape.com/newsref/std/cookie_spec.html*)

[Nunez1995] Manuel Nunez, Pedro Palao, Ricardo Pena, **A Second Year Course on Data Structures Based on Functional Programming**, u P. H. Hartel, M. J. Plasmeijer, eds., **Functional programming languages in education**, *FPLE, Nijmegen, The Netherlands, LNCS 1022, Springer-Verlag, Heidelberg*, 1995.

[Owens1999] Christopher Owens, **Proving Correctness of Modular Functional Programs**, Technical Report ECS-LFCS-99-416, *Laboratory for Foundations of Computer Science*, 1999.

[Pavl1996] Gordana Pavlović-Lažetić, **Osnove relacionih baza podataka**, *VESTA i Matematički fakultet, Beograd*, 1996.

[Pyth:FTP] *Corporation for National Research Initiatives*: (*ftp://ftp.python.org*)

[Ragg1997] Dave Ragget, **Client-Side Scripting and HTML**, *WWW Journal, II/2 – "Scripting Languages"*, 1997.

[Ross1997] Guido van Rossum, **Scripting the Web with Python**, *WWW Journal, II/2 – "Scripting Languages"*, 1997.

[Ross2000] Guido van Rossum, **Python Reference Manual R.1.5.2**, *Corporation for National Research Initiatives*, 2000.

[Sabry1992] Amr Sabry, Matthias Felleisen, **Reasoning About Programs in Continuation-Passing Style**, *ACM Conference on LISP and Functional Programming*, 1992..

[Stro1997] Bjarne Stroustrup, **The C++ Programming Language, 3rd Edition**, *Addison Wesley Longman, Inc*, 1997.

[Thom1995] Simon Thompson, Steve Hill, **Functional programming through the curriculum**, u Pieter H. Hartel, Rinus Plasmeijer, eds., **Functional Programming Languages in Education**, *LNCS 1022, Springer-Verlag*, 1995.

[Turn1979] D. A. Turner, **A new implementation technique for applicative languages**, *Software Practice and Experience 9*, 1979.

[Wadl1998] Philip Wadler, **An Angry Half-Dozen**, *ACM SIGPLAN Notices, 33(2):2530*, 1998.

[W3C:1996] N. Freed, N. Borenstein, **Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types**, Technical Report 2046, Draft Standard, *World Wide Web Consortium*, 1996.

[W3C:1997] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners Lee, **Hypertext Transfer Protocol – HTTP/1.1**, *World Wide Web Consortium*, 1997. (*http://www.w3.org/Protocols/rfc2068/rfc2068*)

[W3C:1998] T. Berners-Lee et.al: **Uniform Resource Identifiers (URI) Generic Syntax**, *World Wide Web Consortium*, *RFC 2396*, 1998.

[W3C:1999] D. Raggett, A. Le Hors, I. Jacobs: **HTML 4.01 Specification**, *World Wide Web Consortium*, 1999. *(http://www.w3.org/TR/1999/REC-html401-19991224*)

[W3C:2000] **Extensible Markup Language (XML) 1.0 (Second Edition)**, *W3C Recommendation*, 2000. (*http://www.w3.org/TR/2000/REC-xml-20001006*)