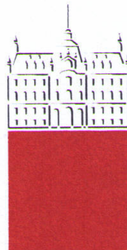
Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si



Št. naloge: 00528/2010

Datum: 01.09.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **UROŠ PERČIČ**

Naslov: **VIZUALNO OKOLJE ZA UČENJE PROGRAMSKEGA JEZIKA C**

A VISUAL ENVIRONMENT FOR LEARNING C

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

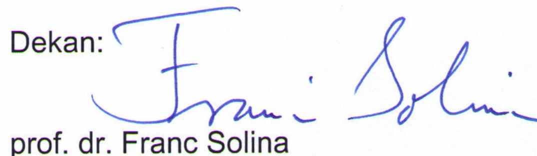
Tematika naloge:

V diplomskem delu preučite razvojna orodja, ki podpirajo vizualizacijo programskih struktur in delovanja programov in so v prvi vrsti namenjena poučevanju programiranja.
Na podlagi analize tovrstnih orodij razvijte program, ki bo namenjen učenju programskega jezika C. Program naj deluje kot interpreter, ki omogoča koračno izvajanje kode, spreminjanje stanja med izvajanjem (vrednosti spremenljivk, vsebina sklada klicev funkcij) in grafičen prikaz gradnje linearnih seznamov in binarnih dreves na enostaven in pregleden način.

Mentor:

doc. dr. Matija Marolt

Dekan:



prof. dr. Franc Solina

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Uroš Perčič

Vizualno okolje za učenje programskega jezika C

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Matija Marolt

Ljubljana, 2010

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Namesto te strani vstavite original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Uroš Perčič,

z vpisno številko 63050306,

sem avtor diplomskega dela z naslovom:

Vizualno okolje za učenje programskega jezika C

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Matija Marolt

  in somentorstvom

  /

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne ____________________ Podpis avtorja: ________________________

Zahvala

Zahvaljujem se mentorju, doc. dr. Matiji Maroltu, brez katerega nasvetov izdelava te diplomske naloge ne bi bila mogoča. Prav tako sem hvaležen staršem, sorodnikom in vsem ostalim, ki so kakorkoli pripomogli pri nastajanju tega dela.

Kazalo

Povzetek 1

Abstract 2

1 Uvod 3
- 1.1 Motivacija in cilji 3
- 1.2 Pregled poglavij 4

2 Pregled področja 5
- 2.1 jGRASP 5
- 2.2 Memview 7
  - 2.2.1 Sklad klicev funkcij 8
  - 2.2.2 Statičen spomin 8
  - 2.2.3 Objektni spomin 9
  - 2.2.4 Ime in vidljivost 9
- 2.3 DrJava 9
  - 2.3.1 Vmesnik za neposredno interakcijo 9
    - 2.3.1.1 Testiranje in razhroščevanje 10
    - 2.3.1.2 Delo s knjižnicami 10
  - 2.3.2 Urejevalnik 11
  - 2.3.3 Integrirani razhroščevalnik 12
- 2.4 BlueJ 12
- 2.5 CLIP 12
  - 2.5.1 Interaktivni način 13
- 2.6 VIP 13

3 Uporabljeni koncepti in orodja 14
- 3.1 Flex 14
  - 3.1.1 Splošno o programu 14
  - 3.1.2 Vhodna specifikacija 16
  - 3.1.3 Regularni izrazi 16
  - 3.1.4 Akcije 17
- 3.2 Bison 17
  - 3.2.1 Splošno o programu 17
  - 3.2.2 Kontekstno neodvisna gramatika 17
    - 3.2.2.1 LALR(1) gramatika 18
- 3.3 Ogrodje Qt 18

4 Implementacija 20
- 4.1 Splošen opis programa 20
- 4.2 Funkcionalnost programa 21
  - 4.2.1 Menujska vrstica 22
  - 4.2.2 Orodna vrstica 23
  - 4.2.3 Urejevalnik izvorne kode 23
  - 4.2.4 Orodja za pregled notranjega stanja programa 24
    - 4.2.4.1 Sporočila 24

4.2.4.2 Konzola.....25
4.2.4.3 Sklad klicev funkcij.....25
4.2.4.4 Spremenljivke.....26
4.2.4.5 Globalne spremenljivke.....27
4.2.4.6 Grafični prikaz podatkovne strukture.....27
4.3 Izvorna koda.....29
4.3.1 Vgrajeni operatorji.....29
4.3.2 Podatkovni tipi.....30
4.3.3 Vgrajene funkcije.....30
4.3.4 Kontrolne strukture.....30
4.3.5 Pisanje novih funkcij.....30
4.3.6 Omejitve pri pisanju izvorne kode.....31
4.4 Podrobnejši opis razhroščevalnika.....31
4.4.1 Regularni izrazi za leksikografski analizator.....31
4.4.2 BNF gramatika za sintaktični analizator.....33
4.4.3 Postopek razhroščevanja.....34
5 Zaključek.....35
5.1 Rezultati.....35
5.2 Možnosti za nadaljnje delo.....35
5.2.1 Odpravljanje omejitev izvorne kode.....35
5.2.2 Dodatne funkcionalnosti.....36
5.2.3 Prilagajanje aplikacije.....36
5.3 Pomisleki.....36
Seznam slik 38
Viri 39

Seznam uporabljenih kratic in simbolov

API	Application Programming Interface
ASD	Abstraktno Sintaksno Drevo
ASCII	American Standard Code for Information Interchange
BNF	Backus-Naur Form
CSD	Control Structure Diagram
LALR	Look-Ahead Left to right Rightmost derivation
MDI	Multiple Document Window
REPL	Read Evaluate Print Loop
SQL	Structured Query Language
UML	Unified Modeling Language
VHDL	VHSIC Hardware Description Language
XML	Extensible Markup Language

Povzetek

Ljudje imajo ob prvem stiku s programiranjem pogosto težave pri razumevanju nekaterih osnovnih konceptov. Rezultati študij kažejo, da uporaba orodij, ki nudijo vizualno reprezentacijo različnih pogledov na program, lahko v veliki meri pripomorejo k večji razumljivosti programov in lažjemu učenju programiranja. Zaradi pomanjkanja tovrstnih orodij, ki bi bili namenjeni uporabi programskega jezika C, je namen te diplomske naloge razvoj aplikacije, ki bi nudila nekaj pripomočkov za lažje učenje tega jezika in bi se uporabljala pri razlagi snovi.

Pri delu je bila za izdelavo sintaktičnega analizatorja uporabljena kombinacija programov Flex in Bison, za izdelavo grafičnega vmesnika pa ogrodje Qt. Aplikacija je namenjena za operacijski sistem Microsoft Windows, prevesti pa jo je možno tudi na drugih operacijskih sistemih, kot sta npr. GNU/Linux in Mac OS, saj je napisana v programskem jeziku C++.

Rezultat dela je razhroščevalnik, ki interpretira uporabnikovo izvorno kodo in navidezno zažene program. V primeru sintaktičnih napak se izpišejo jasni in preprosti opisi napak. Uporabnik lahko izvaja ukaz za ukazom, pri tem pa opazuje spreminjanje stanja v programu (vrednosti spremenljivk, vsebina sklada klicev funkcij). Program omogoča tudi grafičen prikaz nekaterih podatkovnih struktur (linearnih seznamov, binarnih dreves) na enostaven in pregleden način.

Ključne besede:

programiranje, programski jezik C, razhroščevalnik, učenje, vizualizacija programov

Abstract

People often experience difficulties with understanding some basic consepts when dealing with computer programming for the first time. Studies show that the use of visual tools for depicting different aspects of a computer program can aid in better understanding and easier learning process. Due to the lack of such programs for use with the C programming language, the goal of this assignment is to develop one that could provide functions for easier learning and would be used to assist in teaching introductory programming classes.

The combination of Flex and Bison programs was used for implementing the source code parser and the Qt framework was used for creating the application's graphical user interface. The application was built for the Microsoft Windows operating system and it can also be compiled on several different platforms like GNU/Linux and Mac OS, due to the fact it is written in the C++ programming language.

The result is a debugger which interprets the user's source code and seemingly runs the program. In the event of syntactic errors short and meaningful messages are displayed. The user can execute one command at a time and observe changes to the program's internal state while doing so (variable values, contents of the call stack). The debugger is also able to present certain data structures (linked lists, binary trees) in a simple graphical way.

Keywords:

programming, C programming language, debugger, learning, program visualization

Poglavje 1

1 Uvod

Priprava strukturiranega dokumenta, kot je npr. skripta HTML, ni programiranje, kljub temu, da je za to potrebno določeno znanje. Prav tako to ni "programiranje" videorekorderja, saj gre v tem primeru zgolj za vnaprej določeno zaporedje izbir parametrov delovanja. Znotraj področja, ki ga obravnava ta diplomska naloga, programiranje pomeni ustvarjanje seznama zaporednih ukazov za neko napravo. Programer na vsakem koraku izbira med ukazi in izkorišča njihovo medsebojno interakcijo za dosego svojega cilja. Kljub temu, da so ukazi napisani zaporedno, ne pomeni, da se v takem vrstnem redu tudi izvajajo. Programer mora vseskozi imeti v mislih obnašanje naprave in učinke posameznih ukazov. Programiranje je vezano na konkretne stvari, zato ne zahteva nadpovprečnih sposobnosti abstrakcije, vendar je kljub temu potrebnega nekaj truda, da si ustvarimo "mentalno sliko" programa. Nezmožnost tega je glavni vzrok težav, s katerimi se soočajo začetniki. Dijaki in študenti imajo pogosto težave, ko se prvič seznanijo s programiranjem v enem od programskih jezikov. Rezultati študij [1] kažejo, da jim vizualna reprezentacija računalniškega spomina lahko pomaga pri dojemanju pomena ukazov, objektov, struktur in povezav med njimi.

1.1 Motivacija in cilji

Obstaja veliko integriranih razvojnih okolij in orodij, ki imajo vgrajene dodatne funkcije za pomoč začetnikom pri programiranju. Večinoma so realizirane kot interaktivne grafične sheme in številske predstavitve različnih aspektov programa, ki se dinamično spreminjajo glede na stanje programa. Taka orodja se s pridom uporabljajo v izobraževalnih ustanovah za lažjo in bolj praktično demonstracijo delovanja programov. Programski jezik Java je med bolj priljubljenimi jeziki za poučevanje, saj omogoča spoznavanje z nekaterimi naprednejšimi programskimi koncepti (objektno usmerjeno programiranje, generične strukture, delo z grafiko, omrežno povezovanje itd.) na relativno preprost način. Posledično je večina orodij namenjena delu s tem (in nekaterimi drugimi višjenivojskimi) programskim jezikom. Programski jezik C je zaradi nekaterih nižjenivojskih konceptov manj priljubljen med začetniki. Tudi orodij za lažje razumevanje tega jezika ni ravno na pretek. Diplomska naloga zato predvideva izgradnjo razhroščevalnika (debuggerja) za programski jezik C, ki bi bil namenjen poučevanju jezika C in bi vseboval posebne funkcije, ki prikazujejo obnašanje sistema med izvajanjem prevedenega programa (vsebina pomnilnika in sklada, vrednosti spremenljivk, ...). Izvajanje ukazov pomeni spreminjanje notranjega stanja naprave, to pa je najlažje predstavljivo s števili in različnimi shemami. Pri izdelavi teh funkcij sem se zgledoval po obstoječih rešitvah, ki so opisane v nadaljevanju. Dodal sem še možnost

grafičnega izrisa struktur in kazalcev, če ti predstavljajo linearni eno- ali dvosmerni seznam oziroma binarno drevo.

1.2 Pregled poglavij

V drugem poglavju so nanizane obstoječe rešitve in njihovi opisi. Pri izdelavi razhroščevalnika sem se v veliki meri zgledoval po njih.

Tretje poglavje opisuje koncepte in orodja, ki sem jih uporabil pri izdelavi. Opisani sta orodji za gradnjo leksikografskega in sintaktičnega analizatorja (Flex in Bison), ki sem ju uporabil za kreiranje izvorne kode, ki realizira bistven del razhroščevalnika – analizator izvorne kode. Opisano je tudi ogrodje Qt, s pomočjo katerega sem zgradil grafični uporabniški vmesnik aplikacije.

Četrto poglavje vsebuje opis aplikacije in način uporabe z uporabniškega vidika. Podrobneje so opisana vsa orodja za pregled nad stanjem programa. Na koncu poglavja sledi še podrobnejši opis delovanja programa oz. nekateri implementacijski detajli.

V petem poglavju je napisan zaključek s sklepnimi mislimi. Opravljeno delo je kritično ocenjeno, navedene so tudi ideje za nadaljno delo.

Poglavje 2

2 Pregled področja

Pri implementaciji razhroščevalnika sem nekaj idej dobil med pregledovanjem programov, namenjenih nudenju razvojnega okolja s poudarkom na pedagoških funkcijah. Programi so opisani v podpoglavjih. Po pregledu teh programov sem se odločil, da svoj razhroščevalnik implementiram v jeziku C++ in s pomočjo knjižnice Qt.

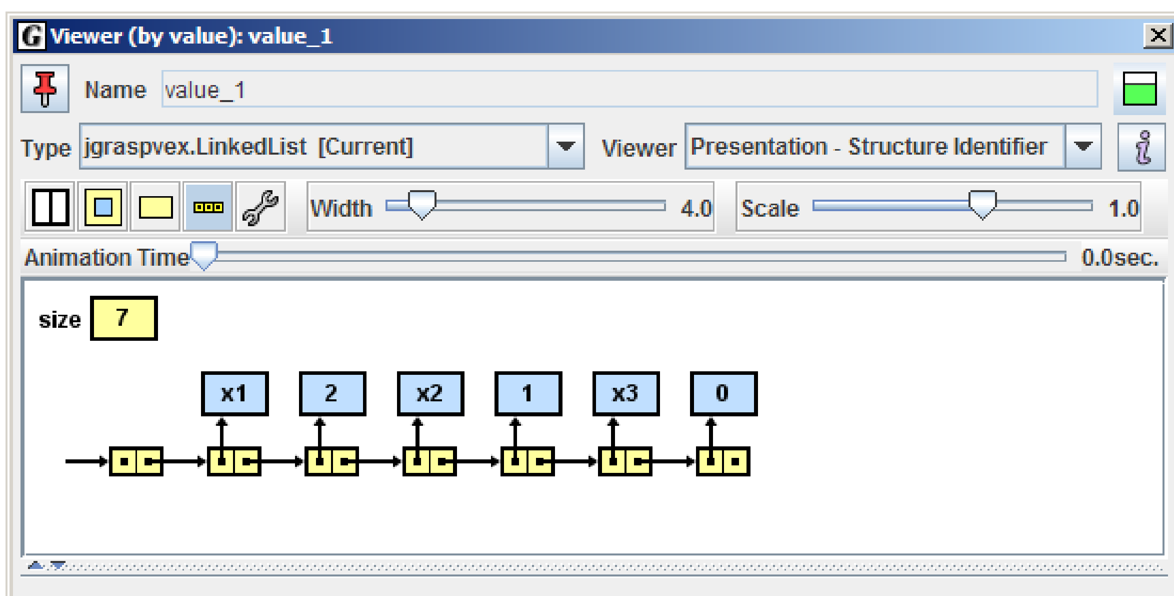
2.1 jGRASP

JGRASP je lahko integrirano razvojno okolje (IDE), namenjeno ustvarjanju različnih vizualizacij, katerih namen je izboljšava razumljivosti izvorne kode programov. Program je napisan v Javi, zaradi česar ga lahko poganjamo na vseh platformah, ki imajo nameščen program Java Virtual Machine. Podprti programski jeziki so Java, C, C++, Objective-C, Ada in VHDL. JGRASP je najnovejše razvojno okolje raziskovalne skupine GRASP (Graphical Representations of Algorithms, Structures and Processes – Grafične Reprezentacije Algoritmov, Struktur in Procesov) z univerze Auburn.

jGRASP omogoča generiranje treh pomembnih tipov vizualizacij: (1) Diagrame kontrolnih struktur (za jezike Java, C, C++, Objective-C, Ada in VHDL) za vizalizacijo izvorne kode, (2) Diagrame razredov UML (Java) za vizualizacijo arhitekture in (3) Dinamične diagrame (Java), s pomočjo katerih lahko vizualiziramo primitivne tipe in objekte, vključno s tradicionalnimi podatkovnimi strukturami, kot so linearni seznami in binarna drevesa. Poleg tega jGRASP nudi tudi urejevalnik objektov (Object Workbench), razhroščevalnik in vmesnik za neposredno interakcijo (Interactions Tab), ki so tesno povezani z vizualizacijami. Vsaka od njih je podrobneje opisana v nadaljevanju.

- **Diagram kontrolnih struktur** (CSD) je algoritmični diagram, ki se generira za programske jezike Java, C, C++, Objective-C, Ada in VHDL). Namenjen je grafičnemu označevanju kontrolnih struktur, kontrolnih poti in celostne strukture vsake programske enote, s čimer se izboljša razumljivost in preglednost izvorne kode. Diagram je prisoten na mestih, kjer je programska koda zamaknjena s tabulatorji. Integriran je v urejevalnik kode in se ga kadarkoli lahko ponovno regenerira. Je alternativa diagramom tokov in drugim reprezentacijam algoritmov. Uporablja se v kombinaciji z arhitekturnimi diagrami (npr. UML diagrami). Z njegovo pomočjo lahko "zložimo" dele kode, ki jih CSD diagram združi (metode, zanke, if stavke) in jih nato odpiramo za vsak nivo posebej. Primer diagrama je prikazan na slikah 2.1 in 2.2. [TODO - več funkcij in več zloženih funkcij]

- **Diagram razredov UML** se generira za kodo, napisano v programskem jeziku Java. Odvisnosti med razredi so označene s puščicami. Z izbiro razreda lahko prikažemo njegove komponente, z izbiro puščic med razredi pa prikažemo odvisnosti. Ta diagram pomaga pri razumevanju pomembnega aspekta objektno usmerjenega programiranja – odvisnosti med razredi.
- Različni **dinamični diagrami** prikazujejo objekte in primitivne tipe, medtem ko uporabnik izvaja program korak za korakom ali ko izvede metodo nad objektom iz urejevalnika objektov. Prezentacijski diagrami (Presentation views) so namenjeni prikazovanju instanc razredov, ki predstavljajo tradicionalne podatkovne strukture. Diagrami med izvajanjem programa poskušajo prepoznati linearne sezname, binarna drevesa, razpršene tabele, sklade, vrste itn. Če je struktura uspešno prepoznana, se odpre ustrezen prezentacijski diagram za objekt. Prezentacijski diagram za linearni seznam je prikazan na sliki 2.1.
- **Urejevalnik objektov** omogoča uporabniku kreiranje instanc razredov in klicanje njihovih metod. Ko se objekt nahaja v urejevalniku, lahko uporabnik odpre okno za spremljanje sprememb, ki jih povzročijo klici metod. Paradigma urejevalnika objektov se je izkazala za zelo uporabno pri učenju konceptov objektno usmerjenega programiranja, še posebej pri študentih začetnikih.
- **Integrirani razhroščevalnik** deluje v kombinaciji z diagramom kontrolnih struktur, diagramom razredov UML, urejevalnikom objektov in vmesnikom za neposredno interakcijo. Nudi način za spremljanje izvajanja programa korak za korakom. Uporabnik lahko spremlja niti, sklad klicev funkcij in lokalne spremenljivke ter njihovo spreminjanje po vsakem koraku. Razhroščevalnik se uporablja med predavanji kot interaktiven medij za razlaganje programov.
- **Vmesnik za neposredno interakcijo** omogoča uporabnikom vnašanje večine stavkov in izrazov jezika Java ter njihovo takojšnjo izvedbo oziroma evaluacijo. Ta vmesnik je uporaben pri učenju in preizkušanju novih elementov v Javi.



Slika 2.1: Prezentacijski diagram za linearni seznam (jGrasp)

2.2 Memview

Memview je pedagoško naravnan vizualni razhroščevalnik. Predvideva razdelitev računalniškega spomina na tri dele:

- sklad klicev funkcij,
- statični objekti, alocirani na kopici (statičen spomin) in
- dinamični objekti, alocirani na kopici (objektni spomin).

Program je implementiran kot dodatek integriranemu razvojnemu okolju DrJava ter nudi dinamičen in interaktiven prikaz računalniškega spomina na podlagi te razdelitve. Njegova preprosta tridelna grafična reprezentacija omogoča začetnikom neposreden pogled na življenjski cikel objektov in jim pomaga pri razumevanju treh ključnih konceptov:

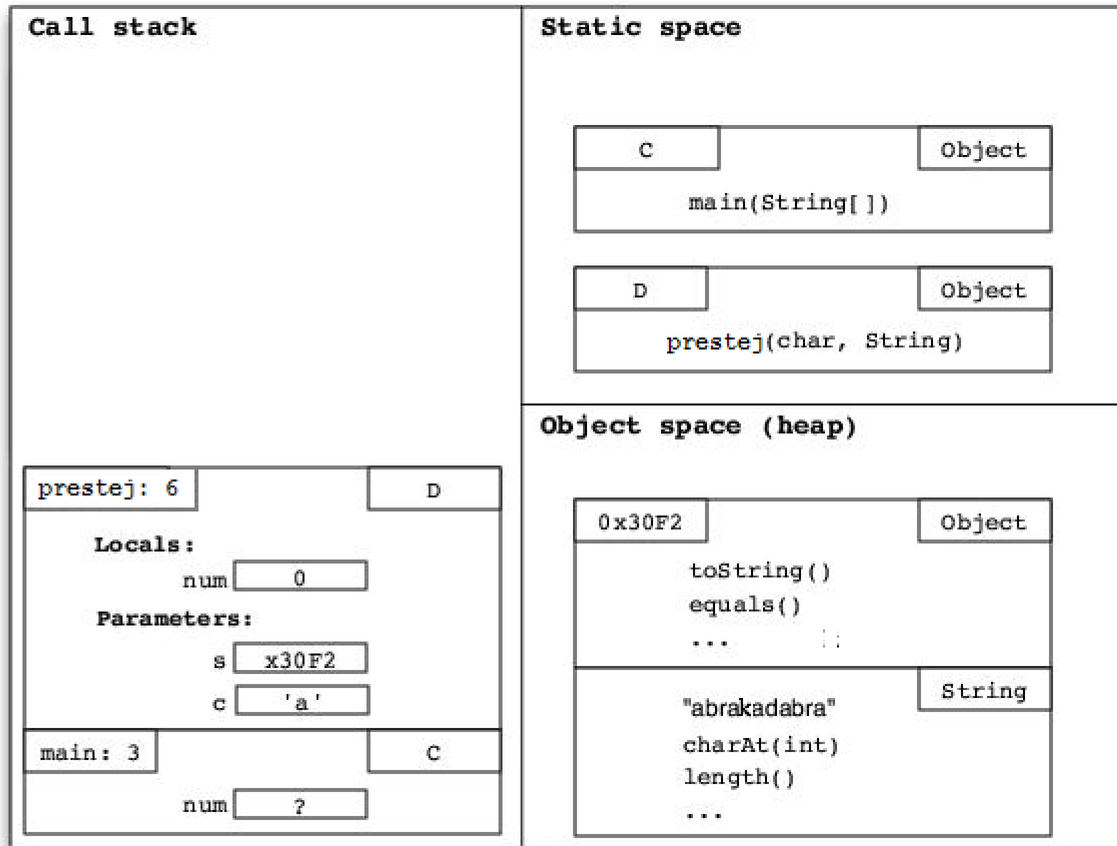
- pojem "naslova" v spominu,
- kako shranjevanje naslova ustvari referenco z enega na drug objekt in
- razlike med kopico, skladom in statičnim spominom.

Slika 2.2 prikazuje stanje spomina v 6. vrstici metode prestej, ki vrne število pojavitev črke 'a' v besedi "abrakadabra":

```
1    public class C {
2        public static void main(String[] args) {
3            int num = D.prestej('a', "abrakadabra");
4            System.out.println(num);
5        }
6    }
```

```
1    public class D {
2        //Prestej stevilo znakov v nizu.
3        public static int prestej(char c, String s) {
4            int num = 0;
5            for(int i = 0; i != s.length(); i++) {
6                if(s.charAt(i) == c) {
7                    num++;
8                }
9            }
10           return num;
11       }
12   }
```

Pravokotnik na levi prikazuje sklad klicev funkcij, pravokotnik desno zgoraj prikazuje statičen spomin in pravokotnik desno spodaj objektni spomin.



Slika 2.2: Stanje spomina v metodi D.prestej (6. vrstica)

2.2.1 Sklad klicev funkcij

Sklad klicev funkcij je predstavljen kot sklad pravokotnikov. Pravokotnik, ki predstavlja funkcijo main, je povsem spodaj, ker je bila ta funkcija poklicana prva. V 3. vrstici se izvede klic metode prestej, zato se na vrh sklada doda ustrezen pravokotnik. Iz slike 2.2 je torej razvidno, da je izvajanje programa ustavljeno v metodi prestej. Vsak pravokotnik je označen z imenom metode in številko vrstice, ki se trenutno izvaja v tej metodi. V notranjosti so predstavljeni parametri funkcije in lokalne spremenljivke.

2.2.2 Statičen spomin

Statične spremenljivke in statične metode so prikazane v statičnem spominu. Vsak razred je predstavljen s pravokotnikom, ki vsebuje te informacije. Na sliki 2.2 sta prikazana razreda C in D. Vsak od njiju ima samo eno metodo. Če imela razreda deklarirane statične spremenljivke, bi se pojavile v ustreznem pravokotniku. V zgornjem desnem kotu vsakega je navedeno tudi ime nadrazreda. V tem primeru C in D dedujeta od razreda Object.

2.2.3 Objektni spomin

V objektnem spominu je na sliki 2.2 prikazan samo en objekt. Ta je tipa String in predstavlja niz znakov "abrakadabra". x30F2 je unikaten, izmišljen pomnilniški naslov v šestnajstiški obliki, na katerem se ta objekt nahaja. Objekt je razdeljen na dva dela: metode, podedovane iz razreda Object in tiste, deklarirane v razredu String.

2.2.4 Ime in vidljivost

Vsak pravokotnik (sklad klicev funkcij, objektni spomin, statični spomin) izgleda približno enako. V zgornjem levem kotu je navedeno ime, v zgornjem desnem kotu pa vidljivost. S pomočjo zadnje ugotovimo, kje dobiti dodatne informacije.

2.3 DrJava

DrJava je lahko, vendar zmogljivo integrirano razvojno okolje, namenjeno uporabi programskega jezika Java, razvito na univerzi Rice. Njegov preprost vmesnik in urejevalnik kode s podporo barvanju na podlagi sintakse omogočata študentom, da se lahko posvetijo razvijanju programerskih sposobnosti, brez dolgotrajnega privajanja na okolje. Ima vgrajen razhroščevalnik in vmesnik za neposredno interakcijo, ki omogoča vnos javanskih stavkov in izrazov ter njihovo takojšnjo izvedbo oziroma evaluacijo. Na univerzi v Torontu se DrJava uporablja na dva načina: kot pomoč študentom, da se posvetijo konceptom in ne mehaniki ter kot pripomoček za predstavitev novih programskih konceptov med predavanji. Študenti ugotavljajo, da neposredna in takojšnja interakcija z novimi funkcijami izboljša razumevanje in pospeši učenje. DrJava ponuja transparenten vmesnik, katerega glavna lastnost je preprostost. Predstavljen je kot okno, razdeljeno na dva dela: konzolo za neposredno interakcijo, s katero lahko neposredno izvajamo ukaze in na konzolo za definicije, kamor vnašamo definicije razredov. Zadnja podpira barvanje kode na podlagi sintakse in avtomatično vstavljanje zamikov. Konzoli sta združeni z vgrajenim razhroščevalnikom, ki prevede razrede, definirane v konzoli za definicije, za uporabo v konzoli za neposredno interakcijo. Z uporabo DrJava se uporabnikom ni treba ukvarjati z vhodno / izhodnimi (VI) funkcijami za izpis besedila, ukazno vrstico, spremenljivkami okolja (npr. CLASSPATH) in kompleksnostmi vmesnikov, ki jih ponujajo komercialna razvojna orodja. Kljub temu, da je DrJava še vedno v fazi aktivnega razvoja, se že uporablja pri začetniških tečajih Jave na številnih srednjih šolah in fakultetah.

2.3.1 Vmesnik za neposredno interakcijo

Vmesnik za neposredno interakcijo deluje po principu zanke "preberi–evaluiraj–izpiši" ("read–eval–print loop" oz. REPL)[2], ki omogoča uporabnikom vnos in takojšnjo izvedbo ukazov in izrazov, vključno z deklaracijo novih spremenljivk. Prikazan je v spodnjem delu slike 2.3. V zadnjih desetletjih je večina funkcionalnih programskih jezikov, kot so Lisp, Scheme in ML, podpiralo REPL za omogočanje inkrementalnega razvoja programov. REPL omogoča konceptualno preprosto, vendar zmogljivo ogrodje za interakcijo s programskimi

komponentami med samim razvojem. Tak vmesnik omogoča programerju hiter dostop do različnih komponent programa brez ponovnega prevajanja ali prilagajanja programske kode. Še posebno uporaben je pri začetniških tečajih programiranja, ker omogoča študentom izvajanje preprostih poskusov, s čimer spoznajo interakcijo različnih programskih konstruktov in vidijo rezultate podizrazov v programu. Vmesnik REPL je uporaben tudi kot prilagodljivo orodje za razhroščevanje in testiranje programov ter eksperimentiranje z novimi knjižnicami. Java je prvi široko uporabljen jezik, ki omogoča učinkovito uporabo vmesnika REPL. Način prevajanja in izvajanja večine programskih jezikov interpreterju onemogoča dostop do elementov prevedene kode ali pa je ta zelo otežen. V tem primeru je potreben interpreter, ki je popolnoma ločen in neodvisen od prevajalnika. Vsi ukazi, vnešeni preko REPL, so izvedeni samo v ločenem interpreterju. Pri taki rešitvi je poleg dvakratne implementacije celotnega jezika – enkrat za prevajalnik in drugič za interpreter – največja težava konsistentnost obeh. Izkušnje pri uporabi pomožnih interpreterjev kažejo, da je ta težava nepremostljiva [3][4].

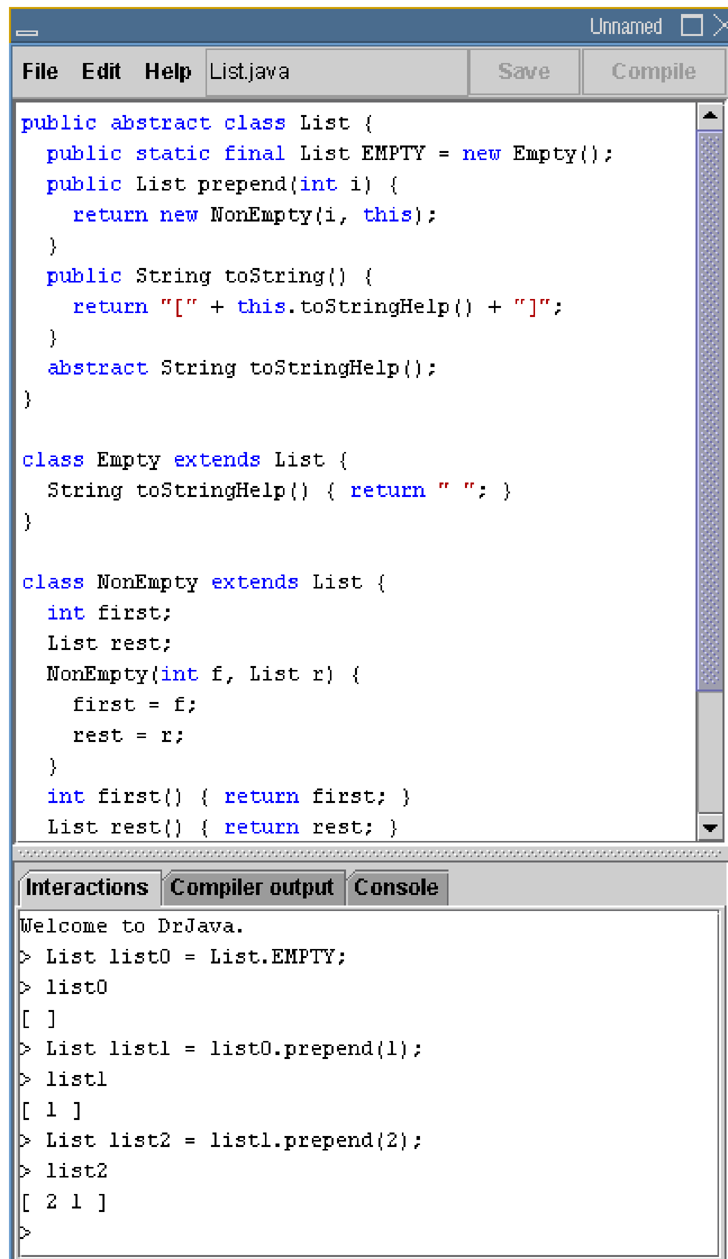
2.3.1.1 Testiranje in razhroščevanje

Vmesnik za neposredno interakcijo je uporaben kot preprosto orodje za testiranje in razhroščevanje programov. Individualne metode lahko testiramo tako, da metode za testiranje vključimo v program in jih nato izvedemo preko REPL kot alternativne vstopne točke v program. Tak pristop je bolj prilagodljiv kot vključevanje metode main v vsak razred. Ko se med testiranjem odkrije napaka, je REPL pogosto boljše orodje za odpravo kot klasičen razhroščevalnik. Čeprav ta omogoča dodajanje kontrolnih točk (breakpoints) in izvajanje programa korak za korakom, programer ne more interaktivno zagnati poljubne metode. Pri konvencionalnem načinu programiranja moramo v ta namen spremeniti main metodo programa, ponovno prevesti in izvesti program in nato povrniti main metodo v prejšnje stanje. Z uporabo REPL lahko brez ponovnega prevajanja izvajanje programa preusmerimo na katerokoli metodo.

Princip REPL je še posebno uporaben za začetnike kot orodje za nadomestilo razhroščevalnika. Pri slednjem se morajo namreč naučiti, kako nastavljati kontrolne točke, kako zbrisati sklad klicev funkcij in kako preverjati vrednost spremenljivk. Z uporabo REPL pa se za izvajanje, testiranje in razhroščevanje uporablja isti vmesnik.

2.3.1.2 Delo s knjižnicami

Vmesnik za neposredno interakcijo služi kot učinkovito sredstvo za raziskovanje programskih vmesnikov knjižnic (Application Programming Interface – API). Študenti se hitreje naučijo uporabljati nove knjižnice, če lahko na preprost način preizkusijo posamezne ukaze. Ta način učenja je še posebej uporaben za grafične knjižnice (npr. Swing). Uporabnik lahko interaktivno ustvarja objekte tipa JFrame ali JPanel, jih prikazuje in opazuje, kako se spreminja njihova vsebina pri dodajanju novih komponent. Takojšen odziv pomaga pri učenju ustvarjanja in razporejanja grafičnih komponent.



Slika 2.3: Okno programa DrJava. Prikazuje urejevalnik in vmesnik za neposredno interakcijo.

2.3.2 Urejevalnik

Ker začetniki pogosto delajo sintaktične napake ter imajo težave pri ugotavljanju le-teh, DrJava poskuša čimprej pravilno zaznati osnovne sintaktične napake. Poleg avtomatičnega zamikanja kode in barvnega označevanja ključnih besed omogoča tudi barvno označevanje parov ujemajočih se oklepajev ter barvanje komentarjev in nizov med navednicama.

2.3.3 Integrirani razhroščevalnik

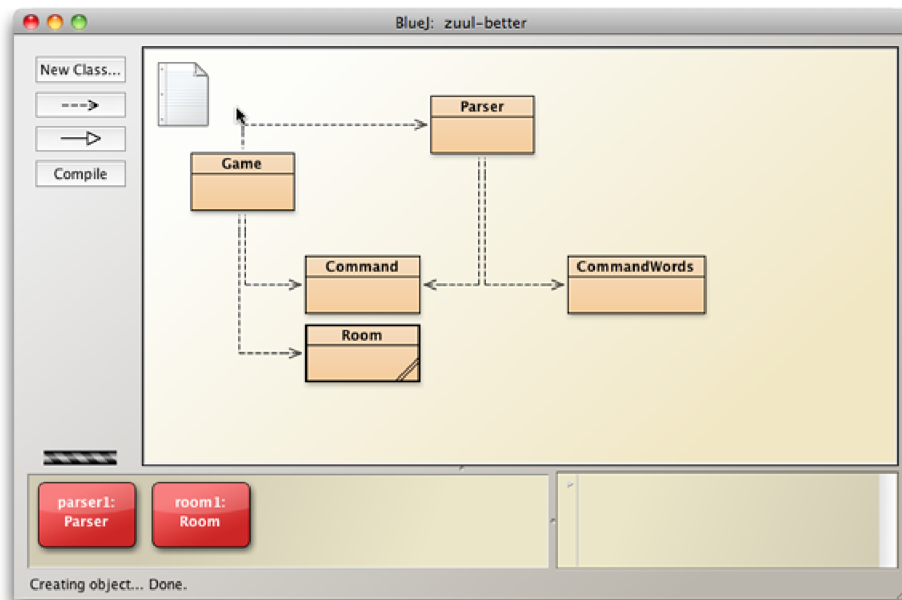
Zaradi enostavnosti je v urejevalnik vgrajen tudi razhroščevalnik, kar omogoča, da se uporabnik premakne na lokacijo sintaktične napake s preprostim klikom. To prihrani uporabnikom interakcijo z razhroščevalnikom kot ločeno entiteto in ročno iskanje lokacij sintaktičnih napak.

2.4 BlueJ

Namen okolja BlueJ je nudenje okolja za učenje objektno usmerjenega programiranja v jeziku Java. Razvijalci so poseben poudarek namenili različnim tehnikam vizualizacije in interakcije z namenom zagotovitve interaktivnega okolja, ki spodbuja eksperimentiranje in raziskovanje. Nekatere posebnosti programa so:

- grafična reprezentacija strukture razredov,
- možnost interaktivnega kreiranja objektov in klicanja njihovih metod,
- interaktivno testiranje.

Glavno okno prikazuje strukturo razredov (slika 2.4) aplikacije, ki jo razvijamo v obliki UML diagramov. Prek tega pogleda lahko uporabnik kreira in testira nove objekte. V njem so na pregleden način prikazani koncepti objektnega programiranja (razredi, objekti, komunikacija preko klicev metod).



Slika 2.4: Glavno okno programa BlueJ, ki prikazuje program v obliki UML diagrama.

2.5 CLIP

CLIP [5] (Command Line InterPreter) je interpreter za podmnožico programskega jezika C++. Poudarjeni so preprosta sporočila sintaktičnih napak, lokalizacija in prijaznost do

začetnikov. CLIP je namenjen učenju prvih korakov v jeziku C++. Uporablja se ga izključno preko ukazne vrstice. Podpira dva načina uporabe: interaktivni način in način za izvajanje posamezne datoteke z izvorno kodo.

2.5.1 Interaktivni način

V tem načinu se uporabnik vseskozi nahaja v globalnem prostoru (global scope). To pomeni, da lahko definira nove strukture in funkcije. Prav tako pa lahko vnaša tudi navadne ukaze in izraze, kot bi bil v notranjosti funkcije main. CLIP v nasprotju s prevajalnikom ne prevaja posameznik ukazov, ampak jih takoj izvede. Uporabniku ni treba vnašati direktiv include (include directives) ter specificirati imenskega prostora, ker CLIP to naredi namesto njega. Možno je tudi vnašanje ukazov, ki zavzamejo več vrstic. Ukazi se preverjajo tudi med izvajanjem, zato program lahko izpisuje različna sporočila, kot npr. deljenje z nič, pregloboka rekurzija, neskončna zanka, kazalec z vrednostjo 0 in uporaba neinicializirane spremenljivke.

2.6 VIP

VIP [6] (Visual InterPreter) je vizualni interpreter za učenje programskega jezika C++. Prikazuje notranje stanje programa in omogoča izvedbo ukazov korak za korakom. Podprt programski jezik je podmnožica jezika C++, ki vsebuje relevantne aspekte za spoznavanje s programiranjem (primitivni tipi, osnovni operatorji, strukture, razreda vector in string, ...). Program je osnovan na Clip interpreterju za C++. Glavno okno programa prikazuje izvorno kodo in notranje stanje programa med izvajanjem. Dodani so gumbi za nadzor izvajanja programa. Nekatere posebnosti:

- Sintaktični analizator[1] je implementiran s pomočjo programa SableCC.
- Pred začetkom izvajanja programa se izvede semantično preverjanje.
- Interpreter zgradi ASD (Abstraktno Sintaksno Drevo). Med izvajanjem se sprehaja po njem in evaluira izraze, ki jih predstavljajo posamezna vozlišča.

1 Sintaktični analizator je algoritem ali računalniški program za slovnično analizo programa.

Poglavje 3

3 Uporabljeni koncepti in orodja

V tem poglavju so opisani koncepti in orodja, ki sem jih uporabljal pri izdelavi diplomaske naloge.

3.1 Flex

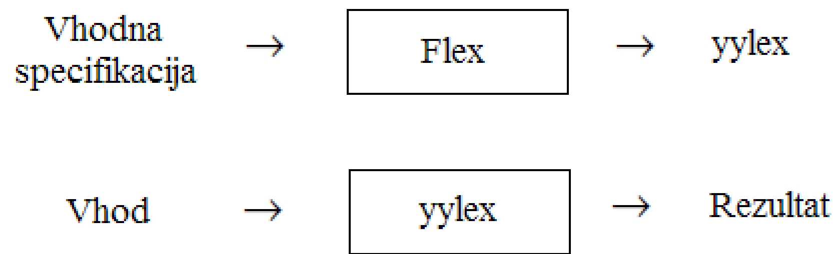
3.1.1 Splošno o programu

Flex je generator programov za leksikografsko procesiranje vhodnega niza znakov. Sprejme visokonivojsko, problemsko usmerjeno specifikacijo za prepoznavanje znakovnih nizov in kot rezultat vrne program v splošno namenskem programskem jeziku, ki prepoznava regularne izraze. Uporabnik specificira regularne izraze v vhodni specifikaciji. Koda, ki jo flex generira, prepozna te izraze v vhodnem nizu in ga razdeli na nize znakov, ki ustrezajo izrazom. Med prepoznavanjem izrazov se izvaja programska koda, kakor jo določi uporabnik. Ta vsakemu izrazu lahko določi svoj del kode. Ko s strani flex-a generiran program prepozna niz, ki ustreza določenemu regularnemu izrazu, se izvede koda, ki mu je pridružena.

Poleg prepoznave izrazov je ponavadi potrebno doseči še dodatne cilje, kar uporabnik doseže z dodatno programsko kodo, ki je lahko celo rezultat drugih generatorjev. Nato se generira program za prepoznavo izrazov v splošno namenskem programskem jeziku, ki mu uporabnik lahko doda svojo kodo. Zato se za navedbo znakovnih izrazov uporablja visokonivojski izrazni jezik, uporabnikova svoboda pri dodajanju svoje kode pa ni zmanjšana.

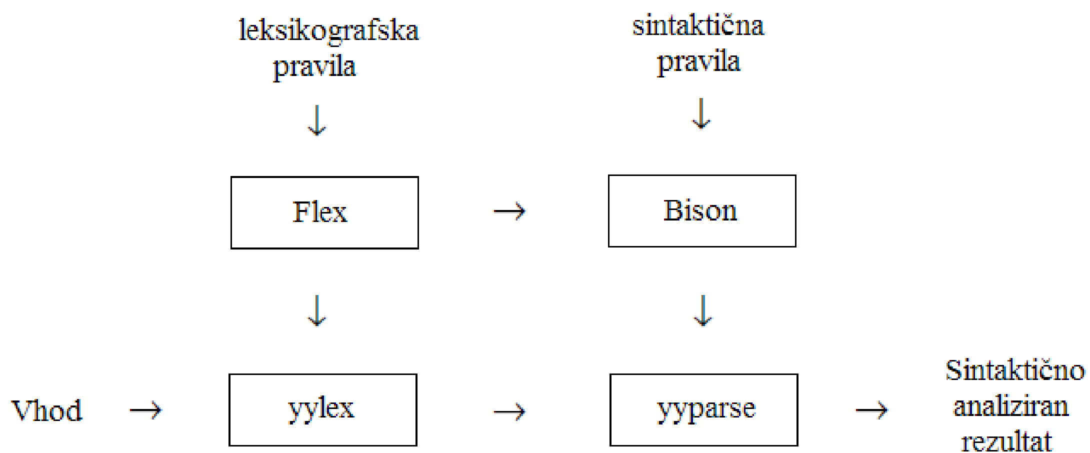
Flex ni popoln jezik, temveč generator, ki predstavlja opcijo za razširitev različnih programskih jezikov – gostiteljskih jezikov (host languages). Kakor lahko splošno namenski jeziki producirajo kodo za različne tipe računalnikov, lahko rezultat flex-a uporabimo z različnimi gostiteljskimi jeziki. Gostiteljski jezik se uporablja tako v kodi, ki jo flex generira, kot v programskih izsekih, ki jih doda uporabnik. Tako je flex primeren za različne uporabnike in okolja. Flex lahko teče na različnih operacijskih sistemih, generirana koda pa je uporabna povsod, kjer je prisoten ustrezen prevajalnik.

Flex pretvori uporabnikove izraze in akcije (vhodna specifikacija) v splošno namenski gostiteljski jezik. Generiran program se imenuje yylex. Ta prepoznava izraze v nizu (vhodu) in izvede ustrezno akcijo za vsak prepoznan izraz. Princip je prikazan na sliki 3.1.



Slika 3.1: Preprost prikaz principa delovanja programa Flex

Flex se lahko uporablja samostojno za preproste transformacije ali za analizo in zbiranje statističnih podatkov na leksikografskem nivoju. Prav tako se lahko uporablja v kombinaciji z generatorjem sintaktičnega analizatorja. Še posebej lahko ga je uporabljati skupaj s programom Bison. Flex programi prepoznavajo regularne izraze, Bison pa služi generiranju sintaktičnega analizatorja, ki temelji na velikem razredu kontekstno neodvisnih gramatik, vendar pa potrebuje nižjenivojski analizator, ki prepoznava posamezne nize. Zato se pogosto uporablja kombinacija programov Flex in Bison. Če Flex uporabljamo kot preprocesor za sintaktični analizator, služi particioniranju vhodnega toka podatkov, analizator pa dodeli strukturo prepoznanim delom. Ta princip je prikazan na sliki 3.2. V programe, ki jih generira Flex, se lahko vključuje tudi dodatne programe, generirane s strani drugih generatorjev ali napisane na roko.



Slika 3.2: Delovanje programov Flex in Bison

Flex generira deterministični končni avtomat na podlagi regularnih izrazov v vhodni specifikaciji. Avtomat se ne prevaja, temveč se zaradi varčevanja s prostorom interpretira. Rezultat je kljub temu hiter analizator. Čas, potreben za prepoznavo in particioniranje vhodnega toka podatkov, je proporcionalen velikosti vhoda. Število pravil in kompleksnost teh pravil ne vplivata na hitrost delovanja. Vplivata pa na velikost končnega avtomata in s tem na končno velikost programa.

Flex ni omejen na vhodni tok znakov, pri katerem je za razpoznavo nizov dovolj vpogled samo enega naslednjega znaka. Če npr. obstajata dve pravili, eno iščoč niz "ab" in

drugo niz "abcdefg" ter je vhodni tok znakov niz "abcdefh", bo Flex prepoznal "ab" in pustil kazalec pozicije na znaku 'c'. Tak način delovanja je zahtevnejši kot procesiranje bolj preprostih jezikov.

3.1.2 Vhodna specifikacija

Osnovna struktura vhodne specifikacije je naslednja:

```
{ definicije }
%%
{ pravila }
%%
{ uporabnikove dodatne funkcije }
```

Pogosto so definicije in uporabnikove dodatne funkcije kar izpuščene. Drugi niz "%%" je opcijski, prvi pa je nujen, ker označuje začetek sekcije s pravili. Minimalna specifikacija je torej

```
%%
```

(nič definicij, nič pravil), ki se pretvori v program, ki celoten vhodni tok podatkov nespremenjen skopira na izhod.

V zgoraj prikazani strukturi vhodne specifikacije sekcija s pravili predstavlja uporabnikove odločitve o poteku programa. To je tabela, katere levi stolpec vsebuje regularne izraze, desni stolpec pa izseke programske kode (akcije), ki se morajo izvršiti ob prepoznavi pripadajočega izraza. Tako bi npr. pravilo

```
integer        printf("Nasel sem besedo integer");
```

v vhodnem toku iskalo pojavitve niza znakov *integer* in ob vsaki pojavitvi izpisalo sporočilo "Nasel sem besedo integer". V tem primeru je gostiteljski jezik C, za izpis pa je uporabljena funkcija *printf* iz standardne knjižnice. Prvi beli znak označuje konec izraza. Če je akcija samo en izraz v programskem jeziku C, je lahko podan na desni strani vrstice. Če pa je sestavljen ali zavzame več kot eno vrstico, se mora nahajati med zavitima oklepajema.

3.1.3 Regularni izrazi

Z regularnimi izrazi uporabnik specificira nabor nizov, ki morajo biti prepoznani. Regularni izrazi vsebujejo navadne znake in znake, ki predstavljajo operatorje (za izbiro, ponavljanje, …). Črke abecede in številke so vedno obravnavani kot navadni znaki; zato regularni izraz

```
integer
```

išče pojavitve niza znakov "integer" in izraz

```
a57D
```

išče pojavitve niza znakov "a57D".

Znaki, ki predstavljajo operatorje, so: " \ [ ] ˆ – ? . * + | ( ) $ / { } % < >. Če jih želimo uporabiti kot navadne znake, moramo uporabiti ubežno sekvenco. Dvojni narekovaj pa se uporablja za citiranje nizov.

3.1.4 Akcije

Ko program v vhodnem nizu najde ujemanje z določenim regularnim izrazom, izvede ustrezno akcijo. Če ta ni podana, se izvede osnovna akcija – kopiranje celotnega vhoda na izhod. Osnovna akcija se izvede nad vsemi nizi, ki ostanejo neprepoznani s strani regularnih izrazov. Torej mora uporabnik, ki želi prebrati celoten vhodni niz in ne želi ničesar poslati na izhod, ustvariti izraze, ki prepoznajo vse. Pri uporabi Flex-a v kombinaciji z Yacc-om je to najpogostejša situacija. Lahko si predstavljamo, da se namesto kopiranja vhoda na izhod izvajajo akcije – torej lahko pravilo, ki samo kopira vhod na izhod, izpustimo. Prav tako je najverjetneje, da se bo kombinacija znakov iz vhoda, ki ni omenjena med izrazi, izpisala na zaslon in tako opozorila na pomankljivost med izrazi.

3.2 Bison

3.2.1 Splošno o programu

Navadno ima vhod programov neko strukturo. Pravzaprav lahko za vsak program, ki bere ukaze, rečemo, da definira svoj vhodni jezik. Vhodni jezik je lahko kompleksen, kot npr. programski jezik ali pa zelo preprost, kot npr. zaporedje številk. Osnovne metode za branje vhoda so večinoma omejene, zahtevne za uporabo in nenatančne pri preverjanju pravilnosti vhoda.

Bison je orodje za opisovanje vhoda, ki ga končni program sprejme. Uporabnik specificira strukture, ki sestavljajo vhod ter programsko kodo (akcije), ki se mora izvesti ob prepoznavi posamezne strukture. Bison spremeni to specifikacijo v proceduro (sintaktični analizator), ki procesira vhod programa; pogosto je priročno in primerno, da za nadzor nad tokom končnega programa v celoti poskrbi ta procedura. Med procesiranjem vhoda za vsak naslednji osnovni element (žeton oz. token) pokliče funkcijo leksikografskega analizatorja.

Bison je implementiran v programskem jeziku C na način, ki omogoča prenosljivost. Tudi akcije in generiran sintaktični analizator so v jeziku C. Specifikacija, ki jo sprejme, je kontekstno neodvisna gramatika s pravili za razreševanje dvoumnosti, rezultat pa je LALR(1) sintaktični analizator.

Bison in podobne programe (Yacc) se poleg gradnje prevajalnikov uporablja tudi za nekatere druge, manj konvencionalne jezike, kot npr. matematične jezike za kalkulatorje, razhroščevalnike itd.

3.2.2 Kontekstno neodvisna gramatika

Bison za procesiranje jezika potrebuje njegov opis v obliki kontekstno neodvisne gramatike. To je gramatika, ki predstavlja formalni jezik, v katerem so lahko izrazi gnezdeni poljubno globoko, sintaktične strukture pa se ne smejo prekrivati. Predstavlja preprost in natančen mehanizem za opisovanje metod, preko katerih se manjše osnovne enote združujejo v večje,

pri čemer se zajame tudi celotna struktura. Preprostost formalizma je vzrok za natančne matematične študije. Kontekstno neodvisne gramatike igrajo ključno vlogo pri opisovanju in načrtovanju programskih jezikov in prevajalnikov. Prav tako se uporabljajo pri analizi sintakse naravnih jezikov. Izražamo jih z BNF (Backus-Naur Form) notacijo. Ta sestoji iz produkcijskih pravil, ki imajo naslednjo obliko:

$V \rightarrow w$

, kjer je V nekončen simbol, w pa je poljubno zaporedje končnih in nekončnih simbolov (lahko je tudi prazen). Vsa pravila skupaj predstavljajo drevo, kjer so nekončni simboli predstavljeni kot vozlišča, končni simboli kot listi, vsako vozlišče pa ima poddrevo v skladu s produkcijskimi pravili. Drevo predstavlja gnezditveno strukturo izraza, torej se V lahko spreminja, w pa je nespremenljiv.

V kontekstno neodvisni gramatiki levo stran produkcijskega pravila vedno predstavlja en nekončen simbol. V splošni gramatiki bi lahko bil tudi niz končnih in nekončnih simbolov. Gramatika je *kontekstno neodvisna*, ker se nekončen simbol na levi strani pravila vedno lahko nadomesti s tistim, kar je na desni strani. Kontekst, v katerem se simbol pojavi, je torej nepomemben.

Kontekstno neodvisni jeziki so tisti, ki jih lahko razume končni avtomat z enim neskončnim skladom. Za ohranitev nadzora nad gnezdenimi enotami se ob začetku vsake enote na sklad shrani trenutno stanje, ob koncu enote pa se to stanje prebere s sklada.

Blokovna struktura pri programskih jezikih se je prvič pojavila pri projektu Algol, ki je vseboval tudi kontekstno neodvisno gramatiko za opis Algolove sintakse. Te so nato postale standard za programske jezike. Princip blokovne strukture, ki ga uveljavljajo kontekstno neodvisne gramatike, je tako značilen za gramatike, da se pojma sintaksa in gramatika pogosto povezujeta s pravili kontekstno neodvisnih gramatik, predvsem v računalništvu. Formalne omejitve, ki niso zajete v gramatiki, se imenujejo semantika jezika.

Kontekstno neodvisne gramatike so dovolj preproste, da omogočajo izdelavo učinkovitih algoritmov za procesiranje jezika, ki ga predstavljajo. Ti algoritmi za dani vhod ugotovijo zaporedje pravil, ki generira ta vhod. Primer je Earleyev algoritem, najbolj znana pa sta LL in LR algoritma, ki sta bolj učinkovita in delujeta s podmnožico kontekstno neodvisnih gramatik.

Obstaja več različnih podrazredov kontekstno neodvisnih gramatik. Čeprav Bison podpira skoraj vse vrste, je optimiziran za LALR(1) gramatike.

3.2.2.1 LALR(1) gramatika

LALR(1) gramatike so tiste, pri katerih je za procesiranje vhoda vedno dovolj vpogled samo enega žetona naprej. Sintaktični analizatorji za te gramatike so deterministični, kar pomeni, da je v vsakem trenutku naslednje pravilo določeno na podlagi že prebranega dela vhoda in še neprebranega, nespremenljivega dela vhoda (lookahead). Kontekstno neodvisna gramatika je lahko dvoumna, kar pomeni, da obstaja več kot eno zaporedje pravil, s katerimi pridemo do istega vhoda. Tudi nedvoumne gramatike so lahko nedeterministične, kar pomeni, da ne obstaja fiksno število vpogledanih žetonov, ki bi zadostovalo za določanje naslednjega pravila.

3.3 Ogrodje Qt

Qt je večplatformsko ogrodje za razvoj aplikacij, predvsem programov z grafičnimi uporabniškimi vmesniki. Prav tako pa se lahko uporablja za razvoj konzolnih programov in

strežnikov.

Qt uporablja standarden C++, vendar v veliki meri uporablja poseben predprocesor (Meta Object Compiler) za obogatitev jezika. Ta prebere vse izvorne datoteke programa in interpretira določene makro ukaze tako, da jih nadomesti z dodatno C++ programsko kodo z dodatnimi informacijami o razredih, uporabljenih v programu. Qt uporablja te informacije za nudenje funkcij in principov, ki niso na voljo v standardnem C++: sistem signalov in rež (signal/slot system), introspekcija in asinhrono klicanje funkcij. Qt se lahko uporablja tudi z drugimi programskimi jeziki s pomočjo posebnih vmesnikov (language bindings). Teče na vseh večjih platformah in ima močno podporo za internacionalizacijo. Med drugimi funkcijami vključuje tudi možnost interakcije z relacijskimi podatkovnimi bazami SQL, procesiranje dokumentov XML, upravljanje z nitmi, podporo različnim omrežnim standardom in enoten vmesnik za delo z datotečnim sistemom.

Poglavje 4

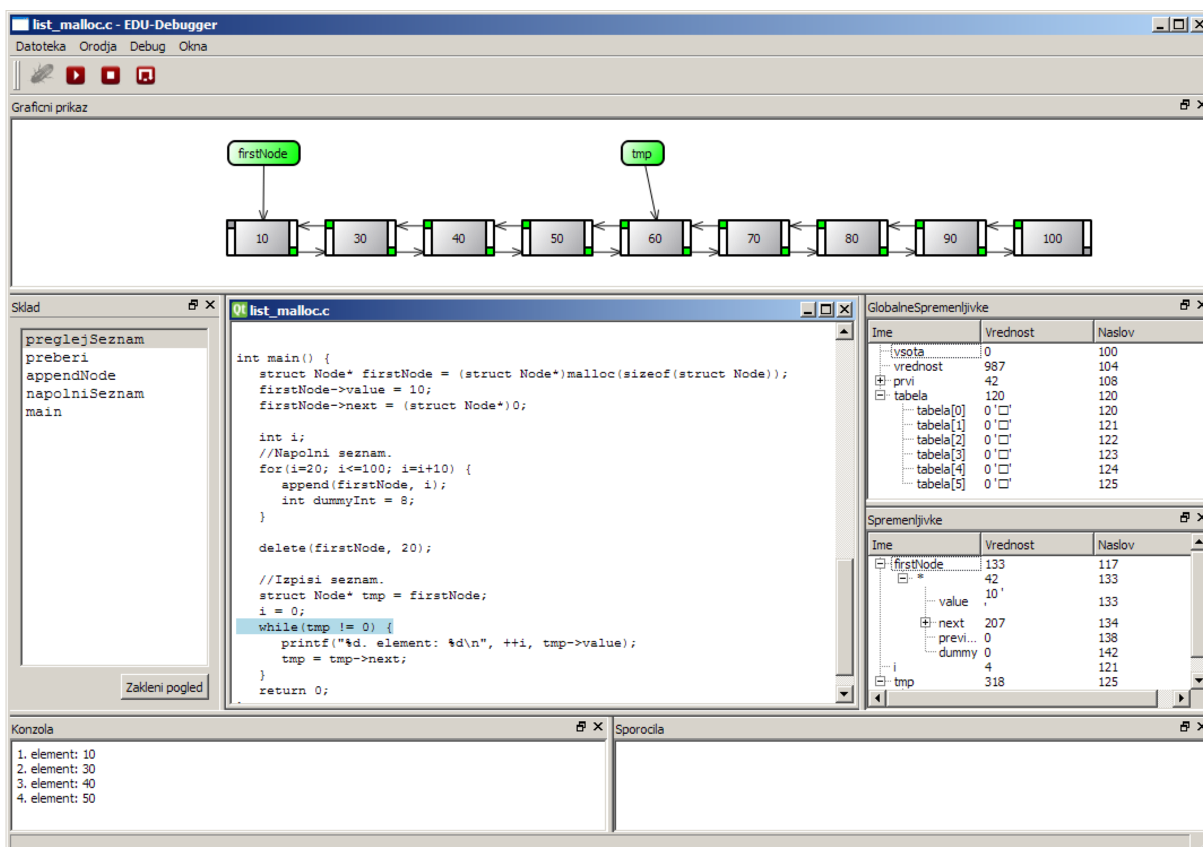
4 Implementacija

4.1 Splošen opis programa

Namen razhroščevalnika je nudenje boljše predstavitve delovanja programa, kot je je uporabnik deležen pri zagonu programa preko ukazne vrstice. Za ta namen sem dodal možnost prikaza več orodij, ki nudijo vpogled v stanje programa. Uporabnik lahko spremlja vrednosti spremenljivk in vsebino sklada klicev funkcij. Prav tako lahko upazuje podatke, ki jih program izpisuje na zaslon in jih vnaša, ko program to zahteva. Posebnost razhroščevalnika pa je grafičen prikaz podatkovnih struktur. Ta se izriše na preprost in pregleden način in pomaga uporabniku pri razumevanju le-teh. Uporabnik lahko enega za drugim izvaja posamezne ukaze in opazuje spremembe, ki jih ti ukazi povzročijo. Glavno okno z vsemi orodji je prikazano na sliki 4.1.

Aplikacija ni razhroščevalnik v pravem pomenu besede. Izvorna koda, ki jo uporabnik "požene", se ne prevede v izvedljiv program, temveč se zgolj sintaktično analizira. Med analizo se zgradi abstraktno sintaksno drevo, ki predstavlja ukaze programa v pravem zaporedju. Izvajanje posameznih ukazov pomeni sprehajanje po tem drevesu in beleženje trenutnega stanja programa v skladu z izvedenimi ukazi. Trenutno stanje programa je prikazano s pomočjo prej omenjenih orodij, ki so opisana v tem poglavju.

V nadaljevanju je podrobno opisana funkcionalnost programa ter način njegove uporabe. Opisani so tudi podprti programski konstrukti jezika C, ki jih uporabnik lahko uporabi v izvorni kodi. Na koncu poglavja so podrobneje opisani še nekateri implementacijski detajli programa.



Slika 4.1: Glavno okno razhroščevalnika

4.2 Funkcionalnost programa

Ob zagonu programa se pokaže glavno okno. Znotraj tega pa se nahajajo vsa orodja za uporabo programa. Uporabnik programa lahko z izbiro ustreznega ukaza v menuju odpre obstoječo tekstovno datoteko z izvorno kodo v programskem jeziku C (z omejitvami, ki so opisane pozneje) ali pa ustvari novo datoteko in jo shrani na trdem disku računalnika. Za tem program omogoči, da uporabnik preveri sintaktično pravilnost svojega programa. To stori z ukazom za začetek razhroščevanja. Vse vrstice, v katerih se nahajajo sintaktične napake, se obarvajo z rdečo barvo. Opis napak se skupaj s številko vrstice izpiše v orodju za izpis sporočil. Vsi ukazi, ki sprožijo opozorilo (npr. različna podatkovna tipa pri prirejanju), pa so podčrtani z oranžno vijugasto črto. Tudi opis opozoril se izpiše v orodju za izpis sporočil. Dokler uporabnik ne odpravi vseh sintaktičnih napak, program ne omogoči začetka razhroščevanja. Ko so odpravljene vse napake, se z modro barvo obarva prva vrstica programa in uporabnik dobi možnost, da začne s pritiskom na gumb ali izbiro ustrezne opcije v menuju izvajati ukaz za ukazom. Vseskozi je z modro barvo označena vrstica z ukazom, ki se bo izvedel naslednji. Vsa orodja za vpogled v stanje programa se avtomatsko posodabljajo z vsakim izvedenim ukazom. Med razhroščevanjem lahko uporabnik odpre poseben grafičnen prikaz podatkovne strukture (linearni seznam ali binarno drevo). To naredi preko dialoga, v katerem nastavi ustrezne parametre. Če se grafični elementi ne razporedijo na ustrezen način, ima uporabnik možnost, da jih razporedi ročno in s tem izboljša preglednost. Opcije

programa, dialogi in orodja za pregled nad notranjim stanjem programa so opisani v nadaljevanju poglavja.

4.2.1 Menujska vrstica

Na vrhu glavnega okna se nahaja menujska vrstica, ki vsebuje vse ukaze za nadzor nad programom. Prikazana je na sliki 4.2. Menuji, ki jih vsebuje, so naslednji:

Datoteka:

- **Nova** – opcija omogoča uporabniku, da ustvari novo tekstovno datoteko. Bližnjica za opcijo je kombinacija tipk *Control* in *n*. Odpre se prazno okno urejevalnika, s katerim lahko uporabnik napiše svoj program ali vanj prilepi kodo iz odložišča. Uporabnik lahko z večkratno izbiro te opcije hkrati ureja več novih datotek. Izbira opcije ni mogoča med razhroščevanjem.
- **Odpri...** – opcija omogoča uporabniku, da odpre že obstoječo datoteko, ki se nahaja na trdem disku. Bližnjica za opcijo je kombinacija tipk *Control* in *o*. Odpre se dialog za izbiro datoteke. Možno je izbrati samo datoteke s končnico .c. S pritiskom na gumb Odpri se vsebina datoteke pojavi v novem oknu urejevalnika. Ob odprtju datoteke si program zapomni direktorij, v katerem se datoteka nahaja in ob naslednji pojavitvi avtomatsko prikaže njegovo vsebino. Izbira opcije ni mogoča med razhroščevanjem.
- **Shrani** – opcija omogoča uporabniku, da na obstoječi datoteki shrani spremembe, ki jih je naredil preko urejevalnika. Bližnjica za opcijo je kombinacija tipk *Control* in *s*. Če ima uporabnik odprtih več oken urejevalnika, se ukaz izvede nad urejevalnikom, ki je v tistem trenutku aktiven (ima modro obarvano naslovno vrstico). Morebitna zvezdica v naslovni vrstici, ki označuje spremembo vsebine urejevalnika, po izvedbi tega ukaza izgine. Če datoteka še ne obstaja, se ob izbiri te opcije izvede ukaz Shrani kot. Izbira opcije ni mogoča med razhroščevanjem.
- **Shrani kot...** - opcija omogoča uporabniku, da na trdi disk shrani vsebino urejevalnika v obliki tekstovne datoteke s končnico .c. Če ima uporabnik odprtih več oken urejevalnika, se ukaz izvede nad urejevalnikom, ki je v tistem trenutku aktiven (ima modro obarvano naslovno vrstico). Morebitna zvezdica v naslovni vrstici, ki označuje spremembo vsebine urejevalnika, po izvedbi tega ukaza izgine. Izbira te opcije ni mogoča med razhroščevanjem.
- **Izhod** – z izbiro te opcije uporabnik zapre program. Bližnjica za opcijo je kombinacija tipk *Alt* in *F4*. Preden se program zapre, v register zapiše podatke o geometriji in dimenzijah glavnega okna ter vseh orodij za prikaz stanja programa. Prav tako se shranita direktorija zadnje odprte in shranjene datoteke. Te podake nato ob ponovnem zagonu program prebere in jih uporabi za vzpostavitev istega izgleda.

Orodja:

- **Grafični prikaz...** - ob izbiri te opcije se odpre dialog, ki omogoča nastavljanje parametrov za orodje, ki grafično prikaže podatkovne strukture. Dialog in orodje sta podrobneje opisana v nadaljevanju.

Debug:

- **Začni z razhroščevanjem** – ob izbiri te opcije se izvede sintaktično preverjanje programa v aktivnem urejevalniku. Bližnjica za opcijo je tipka *F4*. Morebitne napake in opozorila se ustrezno označijo v urejevalniku in orodju za sporočila. Če ni odkritih

napak, se obarva prva vrstica programa in uporabnik lahko začne z razhroščevanjem. Med razhroščevanjem urejanje izvorne kode programa v urejevalniku ni mogoče.

- **Nadaljuj z izvajanjem programa** – ta opcija se omogoči šele potem, ko uporabnik začne z razhroščevanjem. Ob izbiri program začne izvajati ukaze enega za drugim brez posredovanja uporabnika, dokler se program ne konča.
- **Izvedi naslednji ukaz** – z izbiro te opcije uporabnik izvede ukaz, ki je v tistem trenutku obarvan modro. Ukaz se izvede, osveži se stanje vseh orodij za vpogled v notranje stanje programa in obarva se naslednji ukaz. Bližnjica za opcijo je tipka *F5*.
- **Končaj razhroščevanje** – z izbiro te opcije uporabnik konča razhroščevanje. Prikaz orodij za vpogled v notranje stanje programa se ponastavi. Urejanje kode programa je zopet omogočeno.

Okna:

Menu vsebuje opcije za prikaz/skrivanje orodne vrstice in vseh orodij za vpogled v notranje stanje programa.

Slika 4.2: Menujska vrstica programa

4.2.2 Orodna vrstica

Orodna vrstica vsebuje gumbe, ki ustrezajo izbiram menuja *Debug*. Služi lažjemu dostopu do teh opcij med razhroščevanjem. Prikazana je na sliki 4.3.

Slika 4.3: Orodna vrstica programa

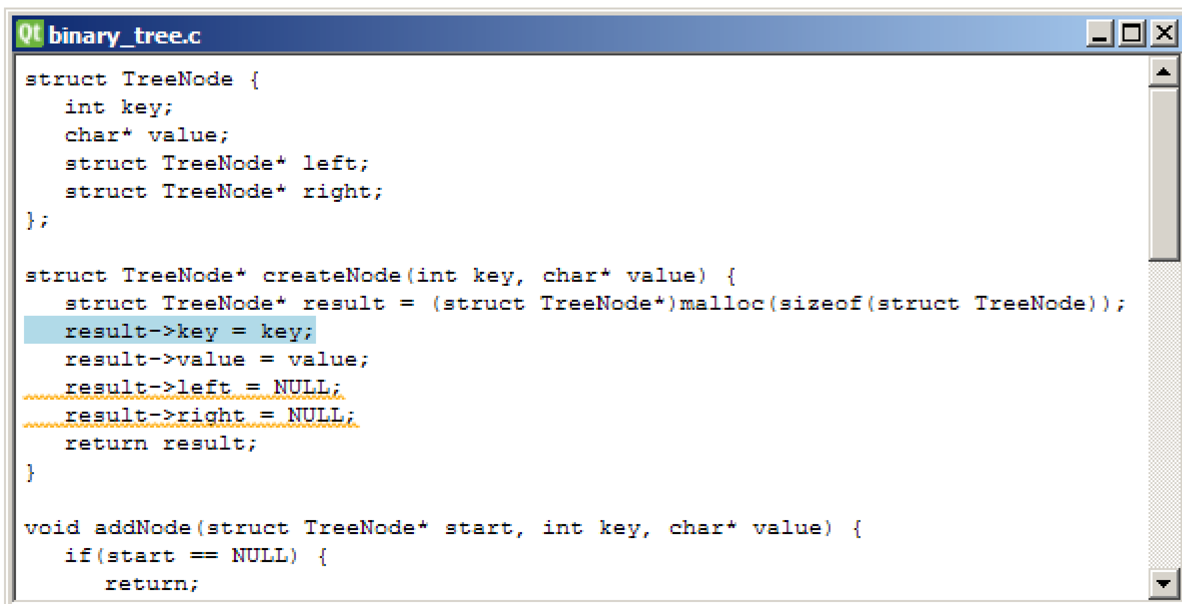
4.2.3 Urejevalnik izvorne kode

Urejevalnik izvorne kode je MDI (Multiple Document Interface) okno, ki vsebuje urejevalnik besedila. Ker se urejevalnik pojavi znotraj območja MDI, ima lahko uporabnik odprtih več urejevalnikov naenkrat. Nov urejevalnik se odpre vedno, kadar uporabnik ustvari novo datoteko ali odpre obstoječo. Uporabnik vsebino aktivnega urejevalnika (tistega, ki ima modro obarvano naslovno vrstico) shranjuje preko menujskih izbir *Shrani* in *Shrani kot*. Neshranjene spremembe v urejevalniku se odražajo z zvezdico, ki se nahaja v naslovni vrstici poleg imena datoteke. Urejevalnik ima tudi nekatere dodatne možnosti, dostopne preko pojavnega menuja (ta se pojavi ob pritisku na urejevalnik z desno miškino tipko) ali bližnjic:

- ***Razveljavi*** (Ctrl + z) – uporabnik lahko razveljavi zadnjih nekaj sprememb v urejevalniku.
- ***Ponovno uveljavi*** (Ctrl + y) – uporabnik lahko ponovno uveljavi razveljavljeno spremembo.

- ***Izreži*** (Ctrl + x) – uporabnik lahko s tem ukazom v odložišče kopira predhodno izbrano besedilo in ga obenem izbriše iz urejevalnika.
- ***Kopiraj*** (Ctrl + c) – uporabnik lahko s tem ukazom v odložišče kopira predhodno izbrano besedilo.
- ***Prilepi*** (Ctrl + v) – uporabnik lahko v urejevalnik na mesto, kjer se nahaja kurzor, prilepi vsebino odložišča.
- ***Izbriši*** – s tem ukazom uporabnik lahko izbriše predhodno izbrano besedilo.
- ***Izberi vse*** (Ctrl + a) – s tem ukazom uporabnik izbere celotno besedilo, ki se nahaja v urejevalniku.

V fazi sintaktičnega preverjanja izvorne kode so vrstice, ki vsebujejo napake, obarvane z rdečo barvo, vrstice, ki vsebujejo opozorila, pa so podčrtane z oranžno vijugasto črto. Med razhroščevanjem je urejanje besedila v urejevalniku onemogočeno, prav tako ga uporabnik ne more zapreti. Z modro barvo je obarvana vrstica z naslednjim ukazom. Ko uporabnik zaključi z razhroščevanjem, lahko zapre urejevalnik s pritiskom na gumb v desnem zgornjem kotu ali kombinacijo tipk *Ctrl* in *w*. Urejevalnik je prikazan na sliki 4.4.

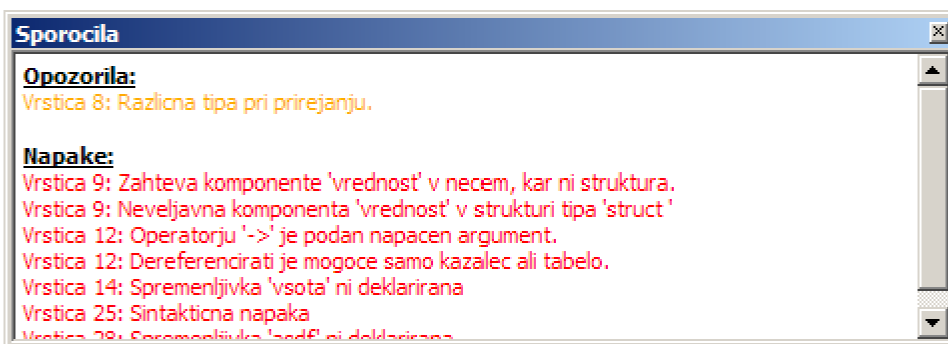


Slika 4.4: Urejevalnik med razhroščevanjem

4.2.4 Orodja za pregled notranjega stanja programa

4.2.4.1 Sporočila

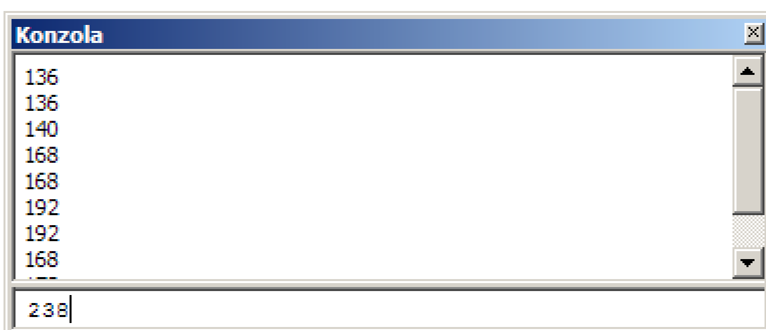
Orodje za prikaz sporočil služi izpisovanju opisov sintaktičnih napak in opozoril v fazi sintaktičnega preverjanja izvorne kode programa. Njegova vsebina se obnovi ob vsaki uporabnikovi zahtevi po začetku razhroščevanja. Orodje in primer izpisa sta prikazana na sliki 4.5.



Slika 4.5: Prikaz opozoril in sintaktičnih napak

4.2.4.2 Konzola

Konzola omogoča interakcijo programa z uporabnikom preko V/I funkcij. Ko program izvede funkcijo za izpis na zaslon (printf), se ustrezni znaki izpišejo v konzoli. Ko program izvede funkcijo za branje podatkov (scanf), lahko uporabnik preko konzole vnese želeni niz znakov. S pritiskom na tipko *Enter* uporabnik pošlje vnešen niz znakov funkciji, ki nadaljuje z izvajanjem. Ob koncu razhroščevanja vsebina konzole ostane nespremenjena, s čimer je omogočeno preverjanje pravilnosti ravnokar zaključenega programa. Ko pa uporabnik ponovno začne z razhroščevanjem, se vsebina konzole zbriše. Primer konzole je prikazan na sliki 4.6.

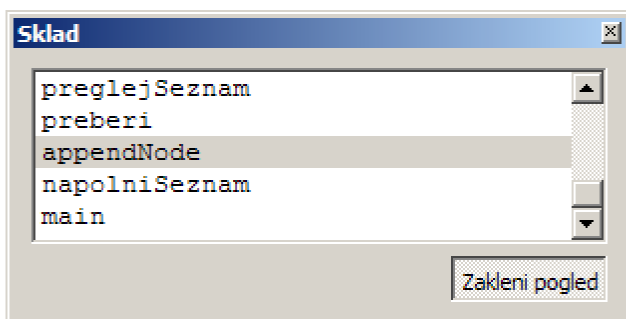


Slika 4.6: Prikaz vhodno/izhodne konzole

4.2.4.3 Sklad klicev funkcij

Orodje za prikaz sklada prikazuje navpičen seznam z imeni funkcij, ki se v danem trenutku nahajajo na skladu. Funkcija, ki je bila poklicana prva (funkcija main) se nahaja na dnu sklada, funkcija, ki je bila poklicana zatem, se nahaja eno stopnjo višje itn. Funkcija, ki je bila poklicana zadnja, se tako nahaja na vrhu seznama. Med razhroščevanjem je vedno aktiven vrhnji vnos seznama (zadnja poklicana funkcija). To pomeni, da orodje za prikaz spremenljivk prikazuje vrednosti vhodnih parametrov in lokalnih spremenljivk te funkcije. S klikom na drug vnos seznama (drugo funkcijo) se prikaz orodja za spremenljivke spremeni tako, da prikazuje vhodne parametre in lokalne spremenljivke izbrane funkcije. Z izvedbo naslednjega ukaza ponovno postane aktivna zadnja funkcija. To lahko uporabnik prepreči tako, da izbere želeno funkcijo in klikne na gumb *Zakleni pogled*. S tem si zagotovi, da bo, dokler bo obstajala na skladu, aktivna izbrana funkcija. Ko funkcije na skladu ni več, lahko uporabnik zaklene pogled na novo funkcijo. Pogled lahko predčasno odklene s ponovnim

pritiskom na gumb. Sklad je prikazan na sliki 4.7.

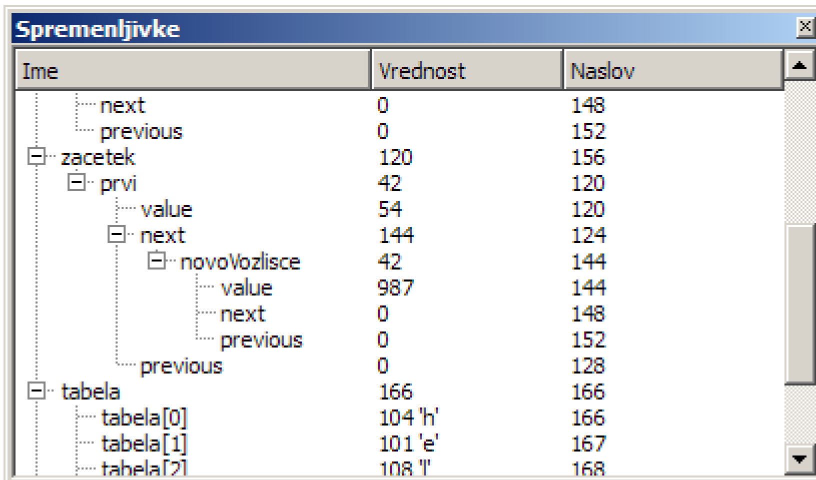


Slika 4.7: Sklad klicev funkcij

4.2.4.4 Spremenljivke

Orodje za prikaz spremenljivk prikazuje vhodne parametre in lokalne spremenljivke funkcije, ki je v določenem trenutku aktivna v orodju za prikaz sklada. Vsaka spremenljivka zaseda eno vrstico, ki je razdeljena na tri stolpce. V prvem stolpcu je prikazano ime spremenljivke, v drugem stolpcu je prikazana vrednost spremenljivke, v trejem stolpcu pa je prikazan izmišljen pomnilniški naslov, na katerem se nahaja spremenljivka. Orodje za prikaz spremenljivk je prikazano na sliki 4.8. Pri spremenljivkah določenega tipa pa so še nekatere posebnosti:

- **char** – Pri spremenljivkah, ki predstavljajo znake, se poleg številske vrednosti med enojnima narekovajema izpiše tudi znak, ki ustreza tej vrednosti po tabeli ASCII (American Standard Code for Information Interchange).
- **struct** – Vrstice, ki predstavljajo strukture, lahko uporabnik z dvoklikom razširi. Odpre se drevesna struktura, ki vsebuje imena vseh strukturinih komponent, njihove vrednosti in pomnilniške naslove.
- **tabela** – Vrstice, ki predstavljajo tabele, lahko uporabnik z dvoklikom razširi. Odpre se drevesna struktura, ki prikazuje vrednosti in pomnilniške naslove vseh elementov tabele.
- **kazalec** – Vrstice, ki predstavljajo kazalce z veljavnimi vrednostmi, lahko uporabnik z dvoklikom razširi. Odpre se drevesna struktura, ki prikazuje spremenljivko, na katero kaže kazalec. Če ima kazalec vrednost 0 ali drugo neveljavno vrednost, drevesne strukture ni mogoče prikazati.



Slika 4.8: Prikaz vhodnih parametrov in lokalnih spremenljivk funkcije

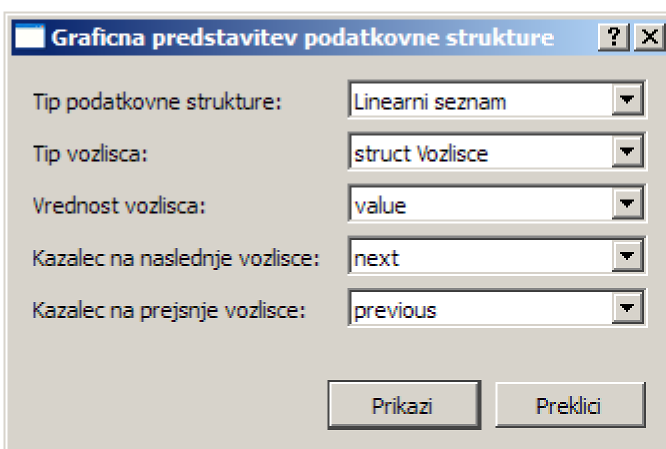
4.2.4.5 Globalne spremenljivke

Orodje za prikaz globalnih spremenljivk prikazuje imena, vrednosti in izmišljene pomnilniške naslove vseh globalnih spremenljivk programa. Ima enake značilnosti kot orodje za prikaz spremenljivk, nabor spremenljivk pa se seveda ne spreminja z dodajanjem in odvzemanjem funkcij s sklada.

4.2.4.6 Grafični prikaz podatkovne strukture

Uporabnik lahko med razhroščevanjem programa prikaže orodje za grafično predstavitev določene podatkovne strukture. Orodje lahko pomaga pri razumevanju nekaterih podatkovnih struktur. Trenutno predvideni podatkovni strukturi sta enosmerni / dvosmerni linearni seznam in binarno drevo. Prikažejo se lahko tudi druge podatkovne strukture, vendar grafično razporejanje kazalcev in struktur verjetno ne bo ustrezno. V takem primeru lahko uporabnik ročno premika prikazane grafične elemente.

Uporabniku se med razhroščevanjem z izbiro ustrezne opcije menuja (Orodja → Grafični prikaz) prikaže dialog, predstavljen na sliki 4.9.



Slika 4.9: Dialog za nastavitev parametrov

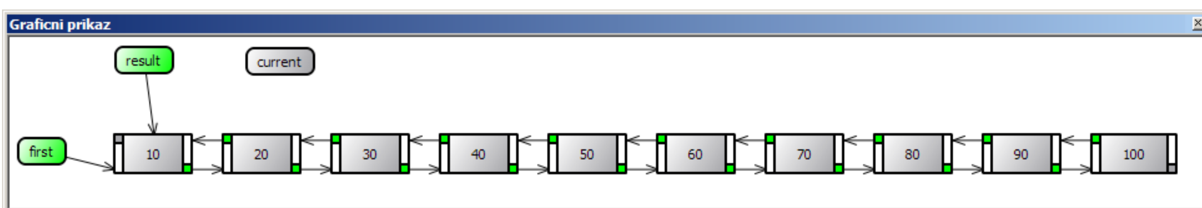
Uporabnik nato nastavi naslednje opcije:

- **Tip podatkovne strukture** – uporabnik v menuju izbere opcijo *Linearni seznam* ali *Binarno drevo* glede na tip podatkovne strukture, ki jo njegov program realizira. Opcija za linearni seznam strukture (vozlišča) razporeja vodoravno, opcija za binarno drevo pa jih razporeja v obliki drevesa.
- **Tip vozlišča** – uporabnik v menuju izbere tip strukture, ki predstavlja vozlišče v prejšnjem koraku izbrane podatkovne strukture. Na voljo ima vse tipe struktur, ki so definirani kjerkoli v programu.
- **Vrednost vozlišča** – uporabnik v menuju izbere komponento strukture, katere vrednost želi prikazano za vsako strukturo v grafičnem prikazu.
- **Kazalec na naslednje vozlišče / Kazalec na desnega otroka** – uporabnik v menuju izmed ponujenih možnosti izbere tisti kazalec, ki predstavlja povezavo na naslednje vozlišče pri linearnem seznamu oziroma tisti kazalec, ki predstavlja povezavo na desnega otroka pri binarnem drevesu.
- **Kazalec na prejšnje vozlišče / Kazalec na levega otroka** – uporabnik v menuju izmed ponujenih možnosti izbere tisti kazalec, ki predstavlja povezavo na prejšnje vozlišče pri dvosmernem linearnem seznamu oziroma tisti kazalec, ki predstavlja povezavo na levega otroka pri binarnem drevesu.

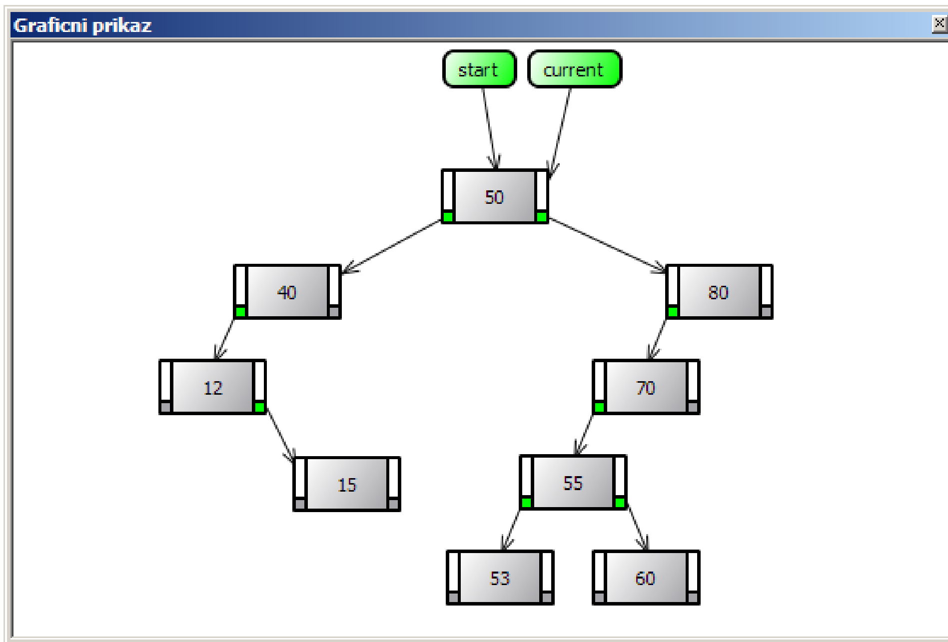
Nato uporabnik pritisne gumb *Prikaži*, ob čemer se odpre okno z grafičnim prikazom. Ta vsebuje vse kazalce, deklarirane znotraj funkcije, ki se v določenem trenutku izvaja ter vse kazalce, deklarirane na globalnem nivoju. Vsak kazalec je predstavljen s pravokotnikom, ki v sredini vsebuje ime kazalca. Če ima kazalec vrednost 0 ali drugo neveljavno vrednost (ne kaže nikamor), je pravokotnik obarvan sivo, če pa ima kazalec veljavno vrednost (kaže na neko spremenljivko), je pravokotnik obarvan zeleno.

Vozlišča so prav tako predstavljena s pravokotniki. V sredini pravokotnikov se nahaja vrednost, ki jo je uporabnik predhodno izbral v dialogu. Vsak pravokotnik vsebuje dva manjša kvadrata, ki predstavljata kazalca na naslednje/prejšnje vozlišče oziroma levega/desnega otroka. Tudi ta kazalca sta obarvana sivo, če imata neveljavno vrednost in obarvana zeleno, če imata veljavno vrednost.

Vsak kazalec z veljavno vrednostjo ima pridruženo usmerjeno puščico, ki kaže na ustrezno vozlišče. Grafični prikaz se posodobi po vsakem izvedenem ukazu. Uporabnik lahko kazalce in vozlišča z vlečenjem miške premika tudi ročno. Primera grafičnih prikazov za dvosmerni linearni seznam in binarno drevo sta prikazana na slikah 4.10 in 4.11.



Slika 4.10: Grafični prikaz dvosmernega linearnega seznama



Slika 4.11: Grafični prikaz (urejenega) binarnega drevesa

4.3 Izvorna koda

Program pred začetkom razhroščevanja prebere in sprocesira uporabnikovo izvorno kodo. Na njeni podlagi gradi abstraktno sintaksno drevo, vmes pa preverja sintaktično pravilnost. Ker program podpira samo podmnožico programskega jezika C, se mora zato uporabnik držati določenih omejitev.

4.3.1 Vgrajeni operatorji

Podprti so naslednji najpogosteje uporabljani operatorji, našteti po prioriteti:

- postfiksna: ++ in --
- klic funkcije s parametri: ( )
- dereferenciacija tabele oz. kazalca: [ ]
- dostop do strukturine komponente: .
- dostop do strukturine komponente preko kazalca: ->
- prefiksna: ++ in --
- unarni minus: -
- sprememba tipa (cast): ( )
- dereferenciacija tabele oz. kazalca: *
- naslov-od: &
- sizeof
- aritmetični: *, /, %
- aritmetični: +, -

- primerjanje: <, <=, >, >=
- primerjanje: ==, !=
- prirejanje: =

4.3.2 Podatkovni tipi

Uporabljati je mogoče naslednje podatkovne tipe:

- char
- short
- int
- long
- float
- double
- struct
- kazalec (*)
- tabela ([ ])
- void

4.3.3 Vgrajene funkcije

Vgrajenih je nekaj najosnovnejših funkcij za V/I operacije in upravljanje s pomnilnikom. Funkcije se uporablja na podoben način kot istoimenske funkcije standardne C knjižnice.

- ***printf*** – Funkcija se uporablja za izpis podatkov na zaslon. Podprti parametri za oblikovanje izpisa so: %d, %c, %f, %s in \n, \t
- ***scanf*** – Funkcija se uporablja za branje podatkov.
- ***malloc*** – Funkcija se uporablja za dinamično rezervacijo pomnilnika.
- ***free*** – Funkcija se uporablja za sproščanje spomina, alociranega s funkcijo malloc.
- ***sizeof*** – Funkcija vrne velikost podane spremenljivke oz. podatkovnega tipa v bajtih.

4.3.4 Kontrolne strukture

Naštete so kontrolne strukture, znotraj zank je mogoča uporaba *continue* in *break* stavkov.

- ***for*** zanka
- ***while*** zanka
- ***do–while*** zanka
- ***if-else*** stavek

4.3.5 Pisanje novih funkcij

Program uporabniku omogoča definiranje lastnih funkcij. Definiranje funkcij je enako

kot v programskem jeziku C. Napisati je potrebno deklaracijo podatkovnega tipa, ki ga funkcija vrača (lahko je tip *void*), ime funkcije, seznam vhodnih parametrov in telo funkcije znotraj zavitih oklepajev. V telesu funkcij lahko uporabnik piše svoje komentarje, ki jih program ignorira. Program pozna dve vrsti komentarjev:

- enovrstični – Začne se z nizom "//" in velja do konca trenutne vrstice.
- večvrstični – Začne se z nizom "/*" in velja vse do pojavitve niza "*/".

4.3.6 Omejitve pri pisanju izvorne kode

Pri pisanju programov v programskem jeziku C se pogosto uporabljajo ukazi, namenjeni preprocesorju. Moj program ne prepozna teh ukazov, zato njihova uporaba ni mogoča.

Program ne podpira določenih kontrolnih struktur:

- switch stavek
- goto stavek
- ternarni pogojni stavek (<pogoj> ? <pogoj_drzi> : <pogoj_ne_drzi>)

Naštete so še nekatere rezervirane besede, ki jih program ne podpira (in s tem tudi njihove funkcionalnosti):

union, typedef, signed, unsigned, const, static

4.4 Podrobnejši opis razhroščevalnika

4.4.1 Regularni izrazi za leksikografski analizator

Leksikografski analizator sem uporabil za prepoznavanje osnovnih gradnikov (žetonov) programskega jezika C. Informacija o teh gradnikih se nato posreduje sintaktičnemu analizatorju preko klica funkcije yylex.

Prepoznavanje rezerviranih besed je v osnovi zelo enostavno. Navedene so definicije za prepoznavo nekaj rezerviranih besed (brez akcij in dodatne programske kode):

```
    "char"          { return CHAR; }
    "short"         { return SHORT; }
    "int"           { return INT; }
…
    "while"         { return WHILE; }
    "for"           { return FOR; }
    "if"            { return IF; }
    "else"          { return ELSE; }
    "break"         { return BREAK; }
…
```

Na isti način sem definiral prepoznavo operatorjev:

```
    "<=" { return LE; }
    "==" { return EQ; }
    "!=" { return NE; }
...
    "++" { return INC; }
    "--" { return DEC; }
…
```

Pri prepoznavanju števil je potrebno ugotoviti vrednost prebranega števila in jo posredovati sintaktičnemu analizatorju:

```
    [1-9][0-9]*     {
         yylval.iValue = atoi(yytext);
         return INTEGER;
    }
```

Izraz je sestavljen iz regularnega izraza, ki prepozna kakršnokoli zaporedje števk, in programske kode znotraj zavitih oklepajev, ki se izvede ob vsaki prepoznavi. yylval je ime strukture, ki jo uporablja sintaktični analizator med procesiranjem vhodnega niza podatkov. V tem primeru nastavimo komponento iValue tako, da ustreza vrednosti prepoznane številke in nato sporočimo, da je naslednji prepoznan žeton številka.

Prepoznavanje znakov med enojnima narekovajema, nizov med dvojnima narekovajema in komentarjev je definirano s pomočjo ekskluzivnih pogojnih pravil. Ta so aktivna samo, kadar je analizator v ustreznem stanju. Tako npr. po branju dvojnega narekovaja analizator vklopi stanje, poimenovano STR. Dokler analizator ne prebere še enega dvojnega narekovaja, ki označuje konec niza, ostane v tem stanju. Ker je pravilo ekskluzivno, se v tem času vhodni niz znakov preverja samo s tem pravilom. Ko analizator prebere zaključujoči dvojni narekovaj, ponovno vklopi navadno stanje, nastavi ustrezno komponento strukture yylval in sporoči, da je naslednji žeton niz znakov. Na podoben način je definirano prepoznavanje posameznih znakov med enojnima narekovajema in večvrstičnih komentarjev.

Prepoznava znakovnih nizov:

```
    \"              { BEGIN STR; s = buf; }
    <STR>\\n        { *s++ = '\n'; }
    <STR>\\t        { *s++ = '\t'; }
    <STR>\\\"       { *s++ = '\"'; }
    <STR>\"         {
                         *s = 0;
                         BEGIN(INITIAL);
                         yylval.cPtr = strdup(buf);
                         return STRING;
                    }
    <STR>\n    {
                    reportParseProblem(
                    true,
                    tr("Niz znakov ni ustrezno zakljucen"),
                    yylineno);
               }
```

```
<STR>.    { *s++ = *yytext; }
```

4.4.2 BNF gramatika za sintaktični analizator

Analizator med delovanjem uporablja spremenljivko yylval za predstavitev semantičnih vrednosti žetonov in posameznih skupin. V osnovi je tip spremenljivke int, možno pa je definirati svoj tip. Definiral sem unijo, ki vključuje vse možne tipe vrednosti. Definirana je na naslednji način:

```
%union {
     int iValue;         /* cela stevila */
     double dValue;      /* decimalne stevilke */
     nodeType* nPtr;     /* node pointer */
     Variable* vPtr;     /* kazalec na spremenljivko */
     char* cPtr;         /* kazalec na niz znakov */
     /* tip spremenljivke, ki ga vrne flex */
     Variable::Type lexVarType;
     /* dokoncen tip spremenljivke */
     std::pair<Variable::Type, std::string>* varType;
     std::vector<int>* sPtr;      /* sklad */
     std::vector<Variable*>* varVector;
     std::vector<nodeType*>* nodeVector;
};
```

Bison generira sintaktični analizator na podlagi vhodne BNF gramatike. Na internetu so dosegljive vseobsegajoče gramatike za programski jezik C, vendar sem ocenil, da bi bilo dodajanje programske kode za gradnjo abstraktnega sintaksnega drevesa preveč zapleteno. Zato sem napisal svojo gramatiko, ki je sicer omejena, vendar omogoča preprosto dodajanje osnovnih funkcionalnosti.

Vsako vozlišče zgrajenega sintaksnega drevesa je predstavljeno s strukturo (nodeType), ki sem jo definiral na naslednji način:

```
typedef enum { typeTerm, typeOpr } nodeEnum;

typedef struct {
     Variable* var;
} termNodeType;

typedef struct {
     int oper;           /* operator */
     int nops;           /* stevilo operandov */
     struct nodeTypeTag* op[1];    /* operandi */
} oprNodeType;

typedef struct nodeTypeTag {
     nodeEnum type;      /* Tip vozlisca */
     /* typeTerm -> spremenljivka ali konstanta. */
```

```
        bool variable;
        int startLine;
        Variable* finalType;
        bool assignable;
        /* return vrednost, ki jo vrne funkcija */
        Variable* result;

        union {
             termNodeType term;  /* spr. ali konst. */
             oprNodeType opr;    /* operator */
        };
    } nodeType;
```

4.4.3 Postopek razhroščevanja

Ko uporabnik odpre želeno datoteko z izvorno kodo, se njena vsebina prebere s trdega diska in prikaže v urejevalniku. Ko uporabnik pritisne gumb za začetek razhroščevanja, se med drugim pokliče funkcija yyparse, s katero je realiziran sintaktični analizator. Ta funkcija za vsak naslednji žeton pokliče funkcijo yylex, s katero je realiziran leksikografski analizator. Programska koda, ki se izvaja ob vsaki ustrezni prepoznavi (akcije), skrbi, da se med analizo gradi abstraktno sintaksno drevo, ki ustreza prebranim ukazom. Hkrati se zbirajo tudi opozorila in sintaktične napake, ki se po koncu analize izpišejo v orodju za prikaz sporočil. Besedilo opozoril in sintaktičnih napak je preprosto in razumljivo. Razhroščevanje se lahko začne, če po koncu analize ni zabeležene nobene napake.

Poglavje 5

5 Zaključek

V diplomskem delu sem se ukvarjal z razvojem aplikacije, ki bi začetnikom olajšala razumevanje programskega jezika C. Študije potrjujejo, da različne vizualne predstavitve programov precej pripomorejo k temu, zato je glavni namen aplikacije nudenje le-teh. Preučil sem že obstoječe rešitve, večinoma namenjene delu z drugimi (višjenivojskimi) programskimi jeziki in poskušal nekaj podobnih rešitev implementirati v moji aplikaciji. V zadnjem poglavju so na kratko opisani rezultat in predlogi za nadaljne delo.

5.1 Rezultati

Razvil sem aplikacijo, ki nudi nekaj vizualnih predstavitev uporabnikovega programa med izvajanjem. Vidna je razlika med lokalnimi in globalnimi spremenljivkami programa, saj so prikazane ločeno. Uporabnik ima vpogled v vrednosti posameznih spremenljivk na kateremkoli koraku med izvajanjem programa. Vizualiziran je tudi sklad klicev funkcij, ki ponazori delovanje rekurzije in omogoča pregled njegove vsebine. Omogoča tudi preprosto in pregledno grafično ponazoritev podatkovnih struktur, kot sta linearni seznam in binarno drevo. Program ob zaznavi sintaktičnih napak izpiše preprosta, lahko razumljiva sporočila. Podpira tudi funkciji za branje in izpisovanje podatkov na zaslon, da lahko uporabnik preveri pravilnost svojega programa.

5.2 Možnosti za nadaljnje delo

Motivacija za nadaljnje delo je povezana predvsem z:

- odpravljanjem omejitev, ki jih aplikacija trenutno postavlja glede izvorne kode,
- nudenjem dodatnih funkcionalnosti in vizualizacij ter izboljšavo obstoječih,
- prilagajanjem aplikacije v skladu z mnenji in željami uporabnikov.

5.2.1 Odpravljanje omejitev izvorne kode

Aplikacija ni pravi razhroščevalnik, saj ne omogoča razhroščevanja izvedljivih programov. Je zgolj nekakšen vizualni interpreter (tolmač), ki sam implementira analizo izvorne kode uporabnikovega programa. Pri pisanju le-te program postavlja določene

omejitve, ki bi jih bilo mogoče odpraviti (dodajanje switch stavka in operatorjev, funkcionalnosti nekaterih rezerviranih besed itd.).

5.2.2 Dodatne funkcionalnosti

Orodje za vizualizacijo podatkovnih struktur trenutno podpira prikaz eno- ali dvosmerno povezanih linearnih seznamov in binarnih dreves. Razporejanje grafičnih elementov bi bilo mogoče dopolniti na načine, ki bi ponazorili tudi druge podatkovne strukture (sklad, razpršena tabela, ...).

Dodati bi bilo mogoče vmesnik, ki bi realiziral princip REPL, ki omogoča dinamično izvajanje ukazov med samim izvajanjem programa.

Urejevalnik izvorne kode bi bilo mogoče dopolniti tako, da bi avtomatično zamikal ustrezne dele kode in z različnimi barvami označeval ključne besede jezika C.

Veliko uporabno vrednost ima možnost uporabe kontrolnih točk (breakpoints). Program jih trenutno ne podpira, implementacija te funkcionalnosti pa je verjetno najbolj uporabna izboljšava, ki bi jo bilo mogoče narediti.

5.2.3 Prilagajanje aplikacije

Aplikacija je namenjena poučevanju programskega jezika C in lažjemu učenju tega jezika. Zato bi bilo smiselno upoštevati vse pripombe, mnenja in želje s strani tako učiteljev kot študentov in prilagajati aplikacijo v skladu z njimi.

5.3 Pomisleki

Orodja, namenjena nudenju dodatne pomoči pri učenju programiranja, so v večini primerov zelo uporabna. Uporabniku namreč omogočajo lažjo predstavo delovanja programov in algoritmov. V prizadevanju za čimpreprostejše in avtomatizirano delo gredo nekatera orodja tako daleč, da lahko zmedejo uporabnika, katerega namen je osvojitev osnov programiranja. Namesto uporabnika poskušajo namreč narediti preveč stvari. S tem skrijejo bistvene koncepte, povezane z izdelavo programov. Pretirano "razvajanje" uporabnika nima blagodejnega učinka na učenje in je pogosto lahko vzrok sklepanja napačnih predpostavk.

Primer so integrirana razvojna okolja, ki omogočajo prevajanje, povezovanje in zagon uporabnikovega programa samo s pritiskom na gumb. Uporabniki tako lahko npr. predpostavljajo, da brez programa Eclipse ni mogoče napisati in zagnati programa v programskem jeziku Java. Drug primer je uporaba principa REPL, ki omogoča interaktivno preučevanje funkcionalnosti programskega jezika. Omogoča klic metod s poljubnimi parametri in celo definiranje novih metod. Ta funkcionalnost ima veliko prednost za začetnike, saj jim v obdobju pisanja programa ni potrebno zapuščati okolja. Tako se lahko osredotočijo na zasnovo svojega programa brez ukvarjanja z različnimi orodji. Princip omogoča branje podatkov in njihov izpis brez uporabe V/I ali grafičnih funkcij. Kljub veliki priročnosti vsega tega pa bi lahko sledilo presenečenje, saj pri navadni uporabi svojega programa tega niso deležni.

Razhroščevalnik, ki sem ga v okviru te diplomske naloge razvil, je zato mišljen zgolj

kot dodaten pripomoček pri razumevanju in ne kot edino sredstvo za učenje programiranja.

Seznam slik

Slika 2.1: Prezentacijski diagram za linearni seznam (jGrasp)..6
Slika 2.2: Stanje spomina v metodi D.prestej (6. vrstica)..8
Slika 2.3: Okno programa DrJava. Prikazuje urejevalnik in vmesnik za neposredno interakcijo...11
Slika 2.4: Glavno okno programa BlueJ, ki prikazuje program v obliki UML diagrama.........12
Slika 3.1: Preprost prikaz principa delovanja programa Flex..15
Slika 3.2: Delovanje programov Flex in Bison...15
Slika 4.1: Glavno okno razhroščevalnika...21
Slika 4.2: Menujska vrstica programa..23
Slika 4.3: Orodna vrstica programa...23
Slika 4.4: Urejevalnik med razhroščevanjem..24
Slika 4.5: Prikaz opozoril in sintaktičnih napak...25
Slika 4.6: Prikaz vhodno/izhodne konzole..25
Slika 4.7: Sklad klicev funkcij...26
Slika 4.8: Prikaz vhodnih parametrov in lokalnih spremenljivk funkcije...............................27
Slika 4.9: Dialog za nastavitev parametrov...27
Slika 4.10: Grafični prikaz dvosmernega linearnega seznama...28
Slika 4.11: Grafični prikaz (urejenega) binarnega drevesa...29

Viri

[1] P. Gries, V. Mnih, J. Taylor, G. Wilson, L. Zamparo, "Memview: A Pedagogically-Motivated Visual Debugger", v zborniku *35th ASEE/IEEE Frontiers in Education Conference*, Indianapolis, Indiana, 2005

[2] E. Sandewall, "Programming in an interactive environment: the Lisp experience", *Computing Surveys*, št. 10, str. 35-71, 1997

[3] R. Conway, T. Wilcox, "Design and Implementation of a Diagnostic Compiler for PL/I", *Communications of ACM*, št. 16, str. 169-179, 1973

[4] P. Shantz, R. German, J.Mitchell, R. Shirley, C. Zarnke, "WATFOR-The University of Waterloo FORTRAN IV Compiler", *Communications of the ACM*, št. 10, str. 41-44, 1967

[5] (2007), CLIP - a Command Line InterPreter for a subset of C++. Dostopno na: http://www.cs.tut.fi/~vip/clip/clip_english.html

[6] (2008), VIP (Visual InterPreter). Dostopno na: http://www.cs.tut.fi/~vip/en/