

# **MAVN prevodilac**

Osnovi paralelnog programiranja i  
softverski alati

Ognjen Jarčević

Mentor: Asistent-master Milica Matić

## Labele (Label.h)

Fajl *Label.h* definiše klasu *Label* kao i propratne promenljive i metode koje se odnose na labele programa. Neki od značajnijih delova fajla *Label.h* su:

### Klasa Label

- odgovarajući konstruktori klase *Label*, *get* i *set* metode njenih polja
- *int position*, označava mesto varijable u globalnoj listi varijabli
- *string name*, koja označava ime varijable

### Lista labela

- `typedef list<Label*> Labels`, imenuje *Llabels* kao listu labela
- `Labels g_labels`, pravi globalnu listu labela koju ćemo koristiti tokom celog zadatka
- `Label* getLabel(string)`, dobavlja labelu iz `g_labels` sa prosleđenim imenom, u slučaju da ne uspe, javlja grešku
- `bool labelExists(name, Labels)`, proverava da li labela (prosleđeno njeno ime) postoji u prosleđenoj listi

```

main.cpp  LexicalAnalysis  (Global Scope)
10 void test()
11 {
12     Label* labela = new Label();
13     Variable* varijabla = new Variable();
14     getLabel("NepostojecaLabela");
15     getVariable("Nepostojeca varijabla");
16 }

Label.h  Variable.h  (Global Scope)
32 if (var->getName() == name)
33 {
34     return var;
35 }
36 }
37 throw std::invalid_argument("Error: wanted label not found in the list!");
38 //return nullptr;
39 }
40 }
41 bool labelExists(string name, Labels& labels)
42 {
43     for (Label* label : labels)
44     {
45         if (label->getName() == name)
46             return true;
47     }
48     return false;
49 }
  
```

Exception Unhandled

Unhandled exception at 0x7596F162 in LexicalAnalysis.exe:  
Microsoft C++ exception: std::invalid\_argument at memory

Copy Details | Start Live Share session...

Exception Settings

## Variable (Variable.h)

Fajl *Variable.h* definiše klasu *Variable* kao i propratne promenljive i metode koje se odnose na labele programa. Neki značajniji delovi fajla *Variable.h* su:

### Klasa Variable

- odgovarajući konstruktori klase *Variable*, *get* i *set* metode njenih polja
- ```
enum VariableType {
    MEM_VAR, //varijabla memorijskog tipa
    REG_VAR, //varijabla koja se skladišti u registar
    NO_TYPE //konstanta
};
```

*VariableType* imenuje sve moguće tipove varijable

- *VariableType m\_type*, označava tip varijable
- *string m\_name*, označava ime varijable
- *int m\_value*, označava vrednost koju varijabla nosi u slučaju da je *MEM\_VAR*, ili konstanta
- *Regs m\_assignment*, označava registar kome je varijabla dodeljena
- implementacije raznih *string toString()* metoda koje se kasnije koriste zarad ispisa varijable u različitim formatima tokom izvršavanja programa

### Lista varijabli

- ```
typedef list<Variable*> Variables
```

, imenuje *Variables* kao listu labela
- *Variables g\_variables*, pravi globalnu listu varijabli koju ćemo koristiti tokom celog zadatka
- *Label\* getVariable(string)*, dobavlja varijablu iz *g\_variables* sa prosleđenim imenom, u slučaju da ne uspe, javlja grešku
- *bool variableExists(name, Variables)*, proverava da li labela (prosleđeno njeno ime) postoji u prosleđenoj listi

```

main.cpp
LexicalAnalysis (Global Scope)
10 void test()
11 {
12     Label* labela = new Label();
13     Variable* varijabla = new Variable();
14     getLabel("NepostojecaLabela");
15     getVariable("Nepostojeca varijabla");
16 }

127 % No issues found

Label.h
Variable.h (Global Scope)
32 if (var->getName() == name)
33 {
34     return var;
35 }
36 }
37 throw std::invalid_argument("Error: wanted label not found in the list!");
38 //return nullptr;
39 }
40 }
41 }
42 {
43 }
44 }
45 }
46 }
47 }

Exception Unhandled
Unhandled exception at 0x7596F162 in LexicalAnalysis.exe:
Microsoft C++ exception: std::invalid_argument at memory
Copy Details | Start Live Share session...
Exception Settings

```

## Instruction (Instruction.h)

Fajl *Instruction.h* definiše klasu *Instruction* kao i propratne promenljive i metode koje se odnose na labele programa. Neki značajniji delovi fajla *Instruction.h*:

### Klasa *Instruction*

- odgovarajući konstruktori klase *Instruction*, *get* i *set* metode njenih polja
- *InstructionType m\_type*, označava tip instrukcije
- *string m\_name*, označava ime varijable
- *int m\_position*, označava mesto varijable u globalnoj listi varijabli
- *Variables m\_dst, m\_src*, liste unutar svake pojedinačne instrukcije koje popunjavamo tokom pravljenja instrukcije unutar sintaksne analize
- *Variables m\_use, m\_in, m\_out, Instructions m\_succ*, liste unutar svake pojedinačne instrukcije koje koristimo tokom analize životnog veka
- *Regs m\_assignment*, označava registar kome je varijabla dodeljena
- *Label\* m\_label*, koje se koristi ukoliko instrukcija koristi ili kreira labelu
- implementacije raznih *string toString()* metoda koje se kasnije koriste zarad ispisa intrukcije u različitim formatima tokom izvršavanja programa

## Lista instrukcija

- `typedef list<Instruction*> Instructions`, imenuje Instructions kao listu instrukcija
- `Instructions g_instructions`, pravi globalnu listu varijabli koju ćemo koristiti tokom celog zadatka
- `Instruction* getLabelPosition(Label*)`, vrati instrukciju iz globale liste instrukcija u kojoj je definisana prosledjena labela, u slučaju da ne uspe, javlja grešku
- `bool variableExists(name, Variables)`, proverava da li labela (prosleđeno njeno ime) postoji u prosleđenoj listi

The screenshot shows two files in a C++ IDE. The top file, `main.cpp`, contains a `test()` function that creates a `Label*` object, pushes it onto `g_labels`, and calls `getLabelPosition(labela)`. The bottom file, `Instruction.h`, shows the implementation of `getLabelPosition`. It attempts to find a label in a list, but when it fails, it throws a `std::invalid_argument` exception with the message "Error: label not found". An "Exception Unhandled" dialog box is displayed, indicating that the exception was not caught. The dialog box text reads: "Unhandled exception at 0x7596F162 in LexicalAnalysis.exe: Microsoft C++ exception: std::invalid\_argument at memory".

## Sintaksna analiza (SyntaxAnalysis.h)

Ako ulaz u vidu `.mavn` fajla leksički ispravan posle leksičke analize bi trebalo da posedujemo listu tokena kojom u ovom slučaju dalje procesuiramo u `SyntaxAnalysis.h`. U ovoj verziji on pored standardne sintaksne analize obavlja i prepoznavanje šablona liste tokena, ispis u mips32 formatu, kao i javljanje grešaka

Novoimplementirani tipovi instrukcija su `I_NOP`, `I_BLT`, `ADD_U`.

### Standardna sintaksna analiza i izbor instrukcija:

Redom prolazimo kroz listu tokena metodom `void eat(TokenType)`. Token koji trenutno analiziramo skladištimo u promenljivoj `Token currentToken`. Kako prolazimo kroz listu tokena i prepoznavamo određene šablone, tako pravimo nove varijable, labela i instrukcije i ubacujemo ih u odgovarajuće globalne liste: `Variables g_variables`, `Labels g_labels`, `Instructions g_instructions`.

### Ispis u fajl u mips32 formatu:

U `SyntaxAnalysis.h` se takođe nalaze i potrebne metode za upis rezultata u fajl u `mips32` formatu. One se obimno služe pozivima raznih metoda za ispis definisanih u `Label.h`, `Variables.h` i `Instruction.h`.

### Javljanje grešaka:

U slučaju da se pojavio neočekivan redosled tokena u listi tokena iliti da je `void eat(TokenType)` metodi prosleđen neočekivan `TokenType`, metoda javlja grešku.

U slučaju da je prepoznata i kreirana nova varijabla ili labela sa već postojećim imenom, program javlja grešku odmah nakon kreiranja te sporne varijable ili labela, a pre dodavanja spornog sadržaja u odgovarajuću listu.

U slučaju da instrukcija sadrži prethodno nedeklarisanu labelu ili varijablu javiće se greška.

```

SyntaxAnalysis.h
LexicalAnalysis
100
101 bool SyntaxAnalysis::Do()
102 {
103     currentToken = getNextToken();
104     eat(T_ERROR);
105     //Q();
106     return !errorFound;
107 }
108
109
110 Syntax error! Token: _mem unexpected
111 Syntax analysis failed!
112
113 D:\Faks\OPPIISA\Projekat\OgnjenJarcevicRA099\Debug\Le
114 xicalAnalysis.exe (process 6368) exited with code 0.
  
```

## Analiza životnog veka promenljive (LivenessAnalysis.h)

Vršenje analize životnog veka promenljive metodom *void livenessAnalysis()* jeste adekvatno dodavanje u varijabli u liste *Variables m\_in* i *Variables m\_out* svake instrukcije u listi *g\_instructions*.

- Metoda *fillSucc()* popunjava listu *m\_succ* svake pojedinačne instrukcije iz liste globalnih instrukcija
- Metoda *fillUse()* popunjava listu *m\_use* svake pojedinačne instrukcije iz liste globalnih instrukcija
- Metoda *printLiveness()* ispisuje instrukcije iz globalne liste instrukcija na konzolu, kao i njihova relevantna polja tokom analize životnog veka
- Metoda *livenessAnalysis()* vrši analizu životnog veka promenljive po već uspostavljenom algoritmu. Ona poziva *fillSucc()* i *fillUse()* pre izvršavanja algoritma, i *printLiveness()* posle svake iteracije

## Alokacija resursa (ResourceAnalysis.h)

Prolazeći kroz listu instrukcija, tj. gledajući *m\_dst* (varijable koje instrukcija definiše) i *m\_out* formiramo graf interferencije između registarskih varijabli. Pomoću formiranog grafa kreiramo i punimo stek varijabli. Broj boja *K* je jednak broju dostupnih registara, u našem slučaju 4.

Biramo čvor sa najvećim brojem interferencija, a da je taj broj manji od *K*. Izabrani čvor uklanjamo sa grafika, i prebacujemo u stek. Potrebno je izvršavati ovaj algoritam dok svi čvorovi grafa nisu obrađeni. U slučaju da u bilo kom momentu ne postoji čvor kome je broj interferencija manji od *K*, memoriju je nemoguće dodeliti (*spill*). U ovom ograničenom okviru zadatka program će javiti grešku, memorija neće biti alocirana, samim tim *output* fajl neće biti ispunjen.

Graf interferencije treba ponovo napuniti analogno prvom putu, i zatim na osnovu njega, steka, i broja registara, alocirati svakoj promenljivoj registar i ukloniti je sa steka. Detaljnije o ovom procesu u funkciji

```
bool doResourceAllocation(InterferenceGraph&, stack<Variable*>&)
```

Nakon uspešne alokacije memorije ispis će u *mips32* formatu biti upisan u fajl.

## Interference graph:

## List of nodes

r1 pos in table: 0  
 r2 pos in table: 1  
 r3 pos in table: 2  
 r4 pos in table: 3  
 r5 pos in table: 4  
 r6 pos in table: 5  
 r7 pos in table: 6  
 r8 pos in table: 7

## Nodes to be moved to stack

r2 pos in table: 1  
 r4 pos in table: 3  
 r5 pos in table: 4  
 r6 pos in table: 5  
 r7 pos in table: 6

## Table view

	r1	r2	r3	r4	r5	r6	r7	r8
r1	—	—	—	—	—	—	—	—
r2	—	—	—	*	*	*	*	—
r3	—	—	—	—	—	—	—	—
r4	—	*	—	—	*	*	*	—
r5	—	*	—	*	—	*	*	—
r6	—	*	—	*	*	—	*	—
r7	—	*	—	*	*	*	—	—
r8	—	—	—	—	—	—	—	—

SPILL!