Generisanje međukoda – rešeni ispitni zadaci

Zadatak 1. (januar 2016.) Definisati klasu za predstavljanje uslovne naredbe u apstraktnom sintaksnom stablu. Uslovna naredba je definisana sledećim skupom smena:

CondStatement → *Statement* **if** *Expression* | *Statement* **unless** *Expression*

Smisao strukture je da se naredba navedena na početku strukture izvršava ukoliko je uslov naveden na kraju zadovoljen kada ispred uslova stoji ključna reč **if**. Ukoliko ispred uslova stoji ključna reč **unless**, naredba treba da se izvrši ako uslov nije zadovoljen.
Definisati zapis ovako definisane uslovne naredbe u međukodu niskog nivoa, i u klasi za predstavljanje ove naredbe u apstraktnom sintaksnom stablu implementirati funkciju *translate* za generisanje takvog međukoda.

Rešenje

Klasa *CondStatement*

```
public class CondStatement extends Statement {
  private Statement s;
  private Expression exp;
  private boolean ifUnless; // true za if, false za unless

  public CondStatement(Statement s, Expression exp, boolean ifUnless) {
    this.s = s;
    this.exp = exp;
    this.ifUnless = ifUnless;
  }

  @Override
  public void translate(BufferedWriter out) throws IOException {
    ...
  }
}
```

Da bi se definisalo telo funkcije *translate* treba najpre definisati izgled generisanog međukoda niskog nivoa.
Mogući međukod:

IF slučaj:

IMC<exp>
Load_Mem R1, RESULT<exp>
JumpIfZero R1, kraj
IMC<s>
kraj:

UNLESS slučaj:

IMC<exp>
Load_Mem R1, RESULT<exp>
JumpIfNotZero R1, kraj
IMC<s>
kraj:

Metoda za generisanje ovakvog međukoda je sledeća:

```
@Override
public void translate(BufferedWriter out) throws IOException {
  String kraj = ASTNode.genLab();
  exp.translate(out);
  exp.genLoad("R1", out);
  if (ifUnless) {
    // za if naredbu ako se evaluacijom exp dobije vrednost false (0 u međukodu)
    // skače se na labelu za kraj i preskače se izvršenje naredbe s
    out.write("JumpIfZero R1, " + kraj + "\n");
  } else {
    // za else naredbu ako se evaluacijom exp
    // dobije vrednost true (1 u međukodu)
    // skače se na labelu za kraj i preskače se izvršenje naredbe s
    out.write("JumpIfNotZero R1, " + kraj + "\n");
  }
  s.translate(out);
  out.write(kraj + ":\n");
}
```

Zadatak 2. (jun 2008.) Definisati klasu za predstavljanje for petlje u apstraktnom sintaksnom stablu. For petlja je definisana sledećom smenom:

ForStatement → **FOR ID =** *Expression* **TO** *Expression* **DO** *Statement*

Navedeni identifikator je brojač for petlje, prvi navedeni izraz je inicijalna vrednost brojača, a drugi izraz je konačna vrednost brojača. Prilikom svakog prolaska kroz petlju vrednost brojača se uvećava za 1.
Definisati zapis ovako definisane for petlje u međukodu niskog nivoa, i u klasi za predstavljanje ove petlje u apstraktnom sintaksnom stablu implementirati funkciju *translate* za generisanje takvog međukoda.

Rešenje

Klasa *ForStatement*

```
public class ForStatement extends Statement {
  private Expression startExp, endExp;
  private Statement statement;
  private Variable id;

  public ForStatement(Expression startExp, Expression endExp, Statement statement,
Variable id) {
    this.startExp = startExp;
    this.endExp = endExp;
    this.statement = statement;
    this.id = id;
  }

  @Override
  public void translate(BufferedWriter out) throws IOException {
    ...
  }
}
```

Da bi se definisalo telo funkcije *translate* treba najpre definisati izgled generisanog međukoda niskog nivoa.
Mogući međukod:

```
  // Prevodimo krajnju i početnu vrednost za brojač petlje.
  IMC <endExp>
  IMC <startExp>
  // U promenljivu id koja predstavlja brojač petlje smeštamo početnu vrednost brojača.
  Load_Mem R1, RESULT<startExp>
  Store R1, id.name
petlja:
  // Učitavamo krajnju vrednost brojača u registar R2.
  Load_Mem R2, RESULT<endExp>
  // Trenutna vrednost brojača se već nalazi u R1.
  Compare_Less R2, R1
  // Ako je uslov poređenja ispunjen u R2 se nalazi vrednost true (!=0), a u suprotnom false (=0).
  JumpIfNotZero R2, kraj
  // Prevodimo naredbu unutar for petlje.
  IMC<statement>
  // Prevođenje naredbe je možda promenilo vrednost u R1 pa
  // ponovo učitavamo trenutnu vrednost brojača u R1.
  Load_Mem R1, id.name
```

```
  // Ne može da doda direktno konstanta 1 na vrednost u registru R1 već mora prvo da se učita u R2.
  Load_Const R2, 1
  Add R1, R2
  // Nova vrednost brojača za narednu iteraciju se smešta u promenljivu id.
  Store R1, id.name
  Jump petlja
kraj:
```

Metoda za generisanje ovakvog međukoda je sledeća:

```java
  @Override
  public void translate(BufferedWriter out) throws IOException {
    // Najpre generišemo labele za pocetak i kraj petlje
    String petlja = ASTNode.genLab();
    String kraj = ASTNode.genLab();
    // Prevodimo krajnju i početnu vrednost za brojač petlje.
    endExp.translate(out);
    startExp.translate(out);
    // U promenljivu id koja predstavlja brojač petlje smeštamo početnu vrednost
    // brojača.
    startExp.genLoad("R1", out);
    out.write("Store R1, " + id.name);
    out.newLine();
    out.write(petlja + ":");
    out.newLine();
    // Učitavamo krajnju vrednost brojača u registar R2.
    endExp.genLoad("R2", out);
    // Trenutna vrednost brojača se već nalazi u R1.
    out.write("Compare_Less R2, R1");
    out.newLine();
    // Ako je uslov poređenja ispunjen u R2 se nalazi
    // vrednost true (!=0), a u suprotnom false (=0).
    out.write("JumpIfNotZero R2, " + kraj);
    out.newLine();
    // Prevodimo naredbu unutar for petlje.
    statement.translate(out);
    // Prevođenje naredbe je možda promenilo vrednost u R1 pa
    // ponovo učitavamo trenutnu vrednost brojača u R1.
    out.write("Load_Mem R1, " + id.name);
    out.newLine();
    // Ne može da doda direktno konstanta 1 na vrednost u registru R1
    // već mora prvo da se učita u R2.
    out.write("Load_Const R2, 1");
    out.newLine();
    out.write("Add R1, R2");
    out.newLine();
    // Nova vrednost brojača za narednu iteraciju se smešta u promenljivu id.
    out.write("Store R1, " + id.name);
    out.newLine();
    out.write("Jump " + petlja);
    out.write(kraj + ":");
    out.newLine();
  }
```

Zadatak 3. (oktobar 2 2019.) Petlja je u jednom programskom jeziku definisana sledećom smenom:

LoopStatement → **loop** *StatementList* **end loop**

Unutar petlje mora da postoji bar jedna exit naredba koja je definisana smenom (smatrati da je provera odrađena u toku semantičke analize):

ExitStatement → **exit when** *Expression* **;**

Definisati klase za predstavljanje ovih naredbi u AST-u. Definisati međukodove niskog nivoa za izvršavanje ovako definisane petlje i implementirati metode za generisanje međukoda.

Rešenje

Klasa *LoopStatement*

```
public class LoopStatement extends Statement {
  private ArrayList<Statement> statements;

  public LoopStatement(ArrayList<Statement> statements) {
    this.statements = statements;
  }

  @Override
  public void translate(BufferedWriter out) throws IOException {
    ...
  }
}
```

Klasa *ExitStatement*

```
public class ExitStatement extends Statement {
  private Expression exitExp;
  private String labelaKraj;

  public ExitStatement(Expression exitExp, String labelaKraj) {
    this.exitExp = exitExp;
    this.labelaKraj = labelaKraj;
  }

  public void setLabelaKraj(String labelaKraj) {
    this.labelaKraj = labelaKraj;
  }

  @Override
  public void translate(BufferedWriter out) throws IOException {
    ...
  }
}
```

Da bi se definisalo telo funkcije *translate* treba najpre definisati izgled generisanog međukoda niskog nivoa.
Mogući međukod:
LoopStatement:

pocetak:
  IMC<statements[0]>
  IMC<statements[1]>
  ...
  IMC<statements[n-1]>
  Jump pocetak
 kraj:

ExitStatement:

  IMC<exitExp>
  Load_Mem R1, RESULT<exitExp>
  JumpIfNotZero R1, kraj
  // Labela kraj je generisana u LoopStatement klasi i njen naziv treba proslediti ExitStatement klasi

Metoda za generisanje ovakvog međukoda u *LoopStatement* klasi je sledeća:

```
@Override
public void translate(BufferedWriter out) throws IOException {
  String pocetak = ASTNode.genLab();
  String kraj = ASTNode.genLab();
  out.write(pocetak + ":");
  out.newLine();
  for (int i = 0; i < statements.size(); i++) {
    Statement current = statements.get(i);
    if (current instanceof ExitStatement) {
      ((ExitStatement)current).setLabelaKraj(kraj);;
    }
    current.translate(out);
  }
  out.write("Jump " + pocetak);
  out.newLine();
  out.write(kraj + ":");
  out.newLine();
}
```

Metoda za generisanje ovakvog međukoda u *ExitStatement* klasi je sledeća:

```
@Override
public void translate(BufferedWriter out) throws IOException {
  exitExp.translate(out);
  exitExp.genLoad("R1", out);
  out.write("JumpIfNotZero R1," + labelaKraj);
  out.newLine();
 }
```

Zadatak 4. (decembar 2019.) „Select izraz“ u jednom programskom jeziku je definisan na sledeći način:

SelectExpression → **select (** *Expression* **,** *ExpressionList* **)**

Definisati klasu za predstavljanje „select izraza” u apstraktnom sintaksnom stablu.

Definisati međukod niskog nivoa za izračunavanje vrednosti „select izraza“ i u klasi koja ovaj izraz predstavlja u apstraktnom sintaksnom stablu implementirati funkciju za generisanje takvog međukoda.

Značenje izraza je sledeće: izračunava se vrednost prvog izraza u zagradi i to predstavlja redni broj izraza iz liste čija će vrednost biti vraćena kao rezultat.

Rešenje

Klasa *SelectExpression*

```
public class SelectExpression extends Expression {
  private Expression uslov;
  private ArrayList<Expression> izrazi;

  public SelectExpression(Expression uslov, ArrayList<Expression> izrazi) {
    this.uslov = uslov;
    this.izrazi = izrazi;
  }

  @Override
  public void translate(BufferedWriter out) throws IOException {
    ...
  }
}
```

Da bi se definisalo telo funkcije *translate* treba najpre definisati izgled generisanog međukoda niskog nivoa.
Mogući međukod:

```
  IMC <uslov>
  Load_Mem R1, RESULT<uslov>
  // Sve dok se ne izvrši IMC za neki od izraza iz liste, sadržaj registra R1 ostaje
  // sigurno nepromenjen i sadrži vrednost izraza uslov.

  // U R2 upisujemo redni broj tekućeg izraza (pretpostavimo da redni brojevi izraza počinju od 1)
  Load_Const R2, 1
  Compare_Equal R2, R1
  // Ako su redni broj tekućeg izraza i vrednost izraza uslov različiti skače se na razmatranje
  // sledećeg izraza.
  JumpIfZero R2, kraj1
  IMC<izraz1>
  Load_Mem R3, RESULT<izraz1>
  Jump krajSelect
kraj1:

  // Ista logika se ponavlja za drugi izraz
  Load_Const R2, 2
  Compare_Equal R2, R1
  JumpIfZero R2, kraj2
  IMC<izraz2>
  Load_Mem R3, RESULT<izraz2>
  Jump krajSelect
kraj2:
  .
  .
  .
  // Ista logika se ponavlja za sve izraze zaključno sa poslednjim N-tim
  Load_Const R2, N
  Compare_Equal R2, R1
  JumpIfZero R2, krajSelect
  IMC<izrazN>
  Load_Mem R3, RESULT<izrazN>

krajSelect:
  // U R3 je svakako rezultat koji treba da se smesti u promenljivu rezulztat za objekat this.
  Store R3, RESULT<SelectExpression>
```

Metoda za generisanje ovakvog međukoda je sledeća:

```
  @Override
  public void translate(BufferedWriter out) throws IOException {
    // krajSelect je poslednja labela koja označava kraj celog select izraza
    String krajSelect = ASTNode.genLab();
    uslov.translate(out);
    uslov.genLoad("R1", out);

    for (int i = 0; i < izrazi.size() - 1; i++) {
      // Generišemo labelu za kraj tekućeg izraza
      String krajTekuceg = ASTNode.genLab();
      // Pretpostavljamo da izraze brojimo kao 1-based
      // Ako želimo da ih brojimo kao 0-based umesto i + 1 ostavićemo samo i
      out.write("Load_Const R2, " + (i + 1));
      out.newLine();
      out.write("Compare_Equal R2, R1");
      out.newLine();
      out.write("JumpIfZero R2, " + krajTekuceg);
      out.newLine();
      Expression tekuciIzraz = izrazi.get(i);
      tekuciIzraz.translate(out);
      tekuciIzraz.genLoad("R3", out);
      out.write("Jump " + krajSelect);
      out.newLine();
      out.write(krajTekuceg + ":");
      out.newLine();
    }
    // Prethodna petlja se završava sa pretposlednjim izrazom iz liste jer
    // kod poslednjeg izraza ne postoji posebna labela za njegov kraj.
    // Generisanje koda za poslednji izraz:
    out.write("Load_Const R2, " + izrazi.size());
    out.newLine();
    out.write("Compare_Equal R2, R1");
    out.newLine();
    out.write("JumpIfZero R2, " + krajSelect);
    out.newLine();
    Expression poslednjiIzraz = izrazi.get(izrazi.size() - 1);
    poslednjiIzraz.translate(out);
    poslednjiIzraz.genLoad("R3", out);

    // Zajednički kraj za ceo select izraz
    out.write(krajSelect + ":");
    out.newLine();
    // U R3 je svakako rezultat koji treba da se smesti u promenljivu
    // rezulztat za objekat this. Naziv promenljive za objekat this
    // treba najpre da generišemo.
    super.result = ASTNode.genLab();
    out.write("Store R3, " + super.result);
    out.newLine();
  }
}
```