Treść zadania:

Zad O2

Napisać program, który optymalizuje kod w C (podzbiór C, bez pointerów etc…). Optymalizacja polega na usunięciu niekoniecznych obliczeń z wnętrz pętli. Zdefiniować składnię kodu wyrażeń które podlegają optymalizacji. Można przyjąć ograniczenia dotyczące użycia w kodzie wskazań na funkcje itp.
Możliwe zagnieżdżenie pętli

Opis zakładanej funkcjonalności:

- Odczyt, parsowanie i analiza kodu z plików tekstowych w języku C
- Sprawdzanie poprawności kodu i zgłaszanie błędów
- Poprawne wykonywanie podanych instrukcji
- Możliwość użycia oraz optymalizacja pętli for przez usunięcie z pętli niekoniecznych obliczeń
- Możliwość użycia instrukcji warunkowych i wyrażeń logicznych ||, &&, ==, !=, <, >, <=, >=, !
- Możliwość użycia operatorów matematycznych +, -, %, /, *
- Możliwość użycia typów int, float, double, long, short

Przykłady:

```
int main(){

int i, b[100] = …, c[100] = … ;

for (i=1; i<100; i++)

{ b[i] = c[i] *a * 135.8; }

return 0;}
```

Kod po przekształceniu

```
int main(){

int i, b[100] = …, c[100] = … ;

float __gen1 = a*135.8;

for (i=1; i<100; i++)

 { b[i] = c[i] *___gen1; }

return 0;}
```

```
int main(){

for (i=1; i<100; i++)

{ if(a%2 == 0) j++; }

return 0;}
```

Kod po przekształceniu

```
int main(){
bool_gen1 = a%2;
for (i=1; i<100; i++)
{ if(bool_gen1) j++; }
return 0;}
```

```
int main(){
for (i=1; i<100; i++) {
Int a = 5;
for (j=1; j<100;j++)
{b[i] = c[j] * a *4;}}
return 0;}
```

Kod po przekształceniu

```
int main(){
for (i=1; i<100; i++) {
Int a = 5 * 4;
for (j=1; j<100;j++)
{b[i] = b[i] + c[j] * a;}}
return 0;}
```

Gramatyka:

Program = int main() MainBlock

MainBlock = ”{“ { IfStatement | InitStatement | AssignStatement | ForStatement | StatementBlock } “return” “0” “}”

StatementBlock = ”{“ { IfStatement | InitStatement | AssignStatement | ForStatement | StatementBlock } “}” ;

ForBlock = ”{“ { IfStatement | InitStatement | AssignStatement | ForStatement | ForBlock | “break” “;” | “continue” “;” } “}” ;

ForStatement = “for” “(“ [variable AssignOp FiniteNumber]”; ”[Condition] ”; ”[AssignStatement] ”) ” ”{“ ForBlock “}”;

IfStatement = “if” “(“ Condition “)” StatementBlock [“else” StatementBlock] ;

InitStatement = Type (Variable | ArrayVariable) [AssignOp Assignable] “;” ;

AssignStatement = (Variable | ArrayVariable) AssignOp Assignable “;” ;

Condition = BaseCondition { LogicalOp BaseCondition};

BaseCondition = [NegationOp] Assigable RelationOp Assigable;

Assignable = Value {(AdditiveOp| MultiplicativeOp) Value}

Type = “int” | “float” | “double” | “long” | “short”

Variable = Letter { Letter, Digit } ;

Index = ”[” Number ”] ” ;

ArrayVariable = Variable Index ;

Value = Variable | FiniteNumber | ArrayVariable ;

RelationOp = ”> ” | ”< ” | ”>= ” | ”<= ” | ”== ” | ”!= ” ;

LogicalOp = ”|| ” | ”&& ” ;

NegationOp = “!” ;

AssignOp = “=” ;

AdditiveOp = ”+ ” | ”- ” ;

MultiplicativeOp = ”* ” | ”/ ” | ”% ” ;

Letter = ”a”…”z” | ”A”…”Z” ;

Digit = ”0”… ”9” ;

NoZero = ”1”… ”9” ;

Number = ”0” | (NoZero , {Digit});

FiniteNumber = [“-“] Number [“.” Digit {Number}]

Wymagania funkcjonalne:

- Odczytanie, parsowanie i analiza kodu zapisanego w pliku tekstowym
- Kontrola poprawności danych oraz zgłaszanie błędów
- Optymalizacja wykonania pętli przez usunięcie niekoniecznych obliczeń
- Poprawne wykonanie odczytanego kodu

Wymagania niefunkcjonalne:

- Program w sposób jasny wskazuje znalezione błędy
- Zmiany w kodzie powstałe w skutek optymalizacji są w wyraźny sposób pokazane użytkownikowi

Sposób uruchomienia:

Program dostaje na wejście z plik z kodem. Program komunikuje wyniki kolejnych etapów analizy z wskazaniem błędów, jeśli takie występują. Zoptymalizowany kod jest wypisywany do nowoutworzonego pliku tekstowego.

Sposób realizacji:

- Program pisany w języku Java
- Główne moduły:
    - Moduł lexera – analiza leksykalna kodu, odpowiedzialna za podzielenie pliku wejściowego na tokeny. Odczyt będzie się odbywał znak po znaku do momentu do momentu odczytania rozpoznawalnego tokenu języka. Zwracane będą one do parsera
    - Moduł parsera – analiza składniowa kodu, pobierając tokeny z lexera sprawdza ich poprawność gramatyczną. Tworzy dzrzewo składniowe
    - Moduł analizatora semantycznego – sprawdza poprawność utworzonego przez parser drzewa.
    - Moduł optymalizacyjny – dokonuje zmian w drzewie w celu optymalizacji w pętlach for
    - Moduł wykonawczy – ma za zadanie sekwencyjne wykonanie instrukcji zawartych w drzewie.

Sposób testowania:

Testowanie będzie przeprowadzone w postaci testów jednostkowych sprawdzających poprawność wykrywania błędów w kodzie oraz poprawność przeprowadzanych operacji. Poprawność optymalizacji będzie sprawdzana przez porównanie wyników przed i po wprowadzeniu zmian do kodu.