1 Wstęp

Zadanie polega na stworzeniu analizatora kodu i kalkulatora dla wyrażeń na łańcuchach (stringach).

Język składa się z wyrażeń na łańcuchach znaków (stringach) oraz zawiera elementy arytmetyki całkowitoliczbowej i wyrażeń logicznych na liczbach całkowitych i łańcuchach. Posiada konstrukcję warunkową `if` (dwie wersje) i pętlę `while` (też dwie wersje). Obsługuje również zmienne dwóch typów: całkowitoliczbowe i łańcuchowe.

2 Gramatyka języka źródłowego

Oznaczenia:
NUM - liczby całkowite ze znakiem, zakres zgodny z typem `signed long int` (32 bity)
STRING – łańcuch alfanumeryczny umieszczony między znakami cudzysłowu (”)
IDENT - identyfikator zmiennej (string, nazewnictwo identyfikatorów standardowe)
Pozostałe słowa kluczowe: **`and`**, **`or`**, **`not`**, **`if`**, **`then`**, **`else`**, **`while`**, **`do`**, **`print`**, **`readint`**, **`readstr`**, **`begin`**, **`end`**, **`exit`**, **`substring`**, **`length`**, **`position`**, **`concatenate`**.

Gramatyka języka:

*** operatory arytmetyczne
```
num_op = "+" | "-" | "*" | "/" | "%"
```
*** wyrażenie numeryczne, którego wartością jest liczba
```
num_expr = NUM | IDENT
     | "readint"
     | "-" num_expr
     | num_expr num_op num_expr
     | "(" num_expr ")"
     | "length(" str_expr ")"
     | "position(" str_expr "," str_expr ")"
```

*** wyrażenie, którego wartością jest łańcuch
```
str_expr = STRING | IDENT
     | "readstr"
     | "concatenate(" str_expr "," str_expr ")"
     | "substring(" str_expr "," num_expr "," num_expr ")"
```

*** operatory logiczne
```
bool_op = "and" | "or"
```
*** relacje logiczne
```
num_rel = "=" | "<" | "<=" | ">" | ">=" | "<>"
str_rel = "==" | "!="
bool_expr = "true" | "false"
     | "(" bool_expr ")"
     | "not" bool_expr
     | bool_expr bool_op bool_expr
     | num_expr num_rel num_expr
     | str_expr str_rel str_expr
```

*** podstawowe konstrukcje
```
simple_instr = assign_stat
```

```
     | if_stat
     | while_stat
     | "begin" instr "end"
     | output_stat
     | "exit"
```

*** ciąg instrukcji
```
instr = instr ";" simple_instr | simple_instr
```

*** przypisanie
```
assign_stat = IDENT ":=" num_expr
     | IDENT ":=" str_expr
```

*** konstrukcja warunkowa
```
if_stat = "if" bool_expr "then" simple_instr
     | "if" bool_expr "then" simple_instr "else" simple_instr
```

*** petla "while"
```
while_stat = "while" bool_expr "do" simple_instr
     | "do" simple_instr "while" bool_expr
```

*** wypisanie informacji na ekran
```
output_stat = "print(" num_expr ")"
     | "print(" str_expr ")"
```

*** program jako taki
```
program = instr
```

Dodatkowe objaśnienia:

- Są dwa typy zmiennych – liczby całkowite ze znakiem, z zakresem odpowiadającym typowi `signed int`, oraz łańcuchy znakowe (odpowiadające typowi `char*`). Zmiennych nie trzeba deklarować, ich pierwsze użycie decyduje o ich typie (niezainicjowana zmienna liczbowa przyjmuje wartość 0, a zmienna łańcuchowa zawiera w takiej sytuacji pusty łańcuch). Sprawdzenie, jakiego typu jest dana zmienna, powinno się odbywać podczas jej użycia.
- Funkcja `length(string)` – zwraca liczbę całkowitą równą długości łańcucha. Długością łańcucha pustego jest wartość 0.
- Funkcja `concatenate(string1,string2)` – zwraca łańcuch będący złączeniem łańcuchów podanych jako parametry.
- Funkcja `substring(string1,pos,length)` – zwraca substring łańcucha podanego jako parametr, rozpoczynający się na pozycji `pos` (wartość liczbowa) i o długości co najwyżej `length` (wartość liczbowa). Pozycja liczona jest od wartości 1. Jeśli wartość pos nie jest poprawna (jest mniejsza od 1 lub większa od długości stringa), funkcja zwraca pusty string. Jeśli wartość `length` jest zbyt duża (`pos+length-1>length(string)`), zwracany jest krótszy łańcuch kończący się na końcu łańcucha wejściowego.
- Funkcja `position(string1,string2)` – zwraca liczbę całkowitą określającą pierwszą pozycję łańcucha `string2` w łańcuchu `string1`, o ile `string2` jest zawarty w `string1`. Jeśli relacja zawierania nie jest spełniona, wartością funkcji jest 0.
- Funkcja `print(expr)` powoduje wypisanie na wyjście (ekran) liczby lub łańcucha będących wynikiem wyrażenia.
- Funkcja `readint` służy o wczytywania z wejścia (klawiatury) liczby całkowitej. Jej wynikiem jest wczytana liczba (można ja przypisać na zmienną lub użyć bezpośrednio w

kodzie, zgodnie z gramatyką).
- Funkcja `readstr` służy o wczytywania z wejścia (klawiatury) łańcucha znaków. Jej wynikiem jest wczytany łańcuch (można go przypisać na zmienną lub użyć bezpośrednio w kodzie, zgodnie z gramatyką).
- Funkcja `exit` powoduje bezwarunkowe zakończenie działania programu.
- Konstrukcje `if...then`, `if...then...else`, `while...do` oraz `do...while` działają w sposób standardowy.
- Konstrukcja `begin...end` służy do grupowania instrukcji prostych.

W ramach realizacji zadania można powyższą gramatykę przekształcić na formę jej równoważną (np. w celu uzyskania jednoznaczności wyprowadzenia operacji arytmetycznych/logicznych).

3 Zadanie

Za pomocą flexa, bisona dla C/C++ (lub odpowiadających im narzędzi dla innych języków programowania) należy napisać program, który będzie:

1. Dokonywał analizy programu zapisanego w powyższym języku w celu określenia jego poprawności i zgodności z powyższą specyfikacją w zakresie:
   a) zgodności z gramatyką,
   b) poprawności typów używanych zmiennych w zależności od miejsca ich występowania (zmienna może wystąpić w wielu miejscach w programie i w każdym z nich musi odpowiadać temu samemu typowi).
2. Wykonywał program zapisany w języku źródłowym (na zasadzie interpretacji, nie kompilacji do innego języka i wykonania tak skompilowanego programu).

Kryteria oceny:

- Za poprawną realizację zadania opisanego w punkcie 1a) można otrzymać ocenę **dostateczną** (wymaga ona jedynie zapisania podanej gramatyki w formie zgodnej z parą Flex/Bison).
- Za poprawną realizację zadania opisanego w punktach 1a)+1b) można otrzymać ocenę **dobrą**. Oczekuję, że w przypadku stwierdzenia użycia jednej zmiennej w różnych rolach, zostaną wypisane wszystkie instrukcje, gdzie została użyta.
- Za poprawną realizację całości zadania opisanego w punktach 1a)+1b)+2 można otrzymać ocenę **bardzo dobrą**.

Faktyczna ocena może się różnić zależnie od jakości rozwiązania oraz zaprezentowanego warsztatu programistycznego.