Metody Translacji

Zadanie zaliczeniowe - rok 2019/20

Przy pomocy narzędzi Gardens Point napisać kompilator opisanego dalej prostego języka programowania Mini. Efektem kompilacji ma być kod źródłowy w języku CIL, który może być za pomocą programu ilasm przekształcony w plik wykonywalny (*.exe) systemu Windows.

Kompilator powinien (musi) pobierać nazwę pliku źródłowego do przetwarzania jako parametr podawany w linii komend. Wynik (kod CIL) musi być generowany do pliku o takiej samej nazwie jak plik zawierający źródłowy uzupełnionej rozszerzeniem il (np. dla pliku źródłowego test.mt wynik musi być w pliku test.mt.il).

Rozwiązania należy nadsyłać mailem w podanych dalej terminach.
Przesłać należy archiwum zip nazwane tak jak Państwa konto w poczcie wydziałowej (np. Jan Malinowski mający konto malinowskij@student.mini.pw.edu.pl przesyła archiwum malinowskij.zip) zawierające dokładnie 4 pliki:
- kompilator.lex (plik dla generatora gplex)
- kompilator.y (plik dla generatora gppg)
- Main.cs (niezbędny kod w C#)
- oswiadczenie.pdf (załączone oświadczenie o samodzielności wykonania zadania)

Zadania będą testowane automatycznie, więc pliki muszą się nazywać dokładnie tak jak to jest opisane powyżej (w przeciwnym razie automatyczna kompilacja się nie powiedzie).
Rozwiązania będą testowane w środowisku Windows 10 z wykorzystaniem C# 7 i Visual Studio 2017 oraz gplex w wersji 1.2.2 i gppg w wersji 1.5.2 (to dokładnie te wersje, które można pobrać ze strony przedmiotu pod linkiem "narzędzia Gardens Point współpracujące z .NET 4.0).
Nie wolno korzystać z elementów wprowadzonych w C# 8, ani Visual Studio 2019.
Generator gppg należy wywoływać z opcją /gplex. Inne opcje nie powinny być potrzebne, gdyby ktoś ich użył, należy mnie o tym poinformować.

Oświadczenie o samodzielności wykonania zadania można pobrać ze strony przedmiotu. Należy je wydrukować, podpisać, zeskanować i dołączyć do rozwiązania zadania. Imię i nazwisko oraz numer albumu można również uzupełnić w Wordzie, ale odręczny podpis jest niezbędny. Prace bez oświadczenia albo z niepodpisanym oświadczeniem nie będą sprawdzane.
Oczywiście prace będą również sprawdzane programem antyplagiatowym (wiecie Państwo z ASD2, że mam do tego narzędzia).

Terminy nadsyłania rozwiązań:
- 10.06.2020, godz. 23.59 - termin podstawowy
- 14.06.2020, godz. 23.59 - kara 0.5 stopnia za spóźnienie
- 19.06.2020, godz. 23.59 - kara cały stopień za spóźnienie

Po 19.06.2020 rozwiązania nie będą przyjmowane (w szczególności nie ma mowy o zaliczeniu we wrześniu).

Opis języka Mini

Poniżej opisana jest podstawowa wersja języka Mini, której bezbłędna implementacja wymagana jest na ocenę 4. Implementacja z niewielkimi usterkami oznacza ocenę 3.5 lub 3.
Na ocenę 4.5 i 5 wymagana jest implementacja opisanej w kolejnym dokumencie rozszerzonej wersji języka (nazwijmy go Mini++).

1) Elementy języka.

Elementami podstawowej wersji języka Mini są następujące terminale:
- słowa kluczowe: **program if else while read write return int double bool true false**
- operatory i symbole specjalne: **= || && | & == != > >= < <= + - * / ! ~ ( ) { } ;**
- identyfikatory i liczby

Identyfikator to ciąg znaków składający się liter i cyfr rozpoczynający się od litery, dozwolone są duże i małe litery i są one rozróżnialne (słowa kluczowe muszą być zapisane małymi literami).

Liczba zmiennopozycyjna składa się z ciągu cyfr oznaczających część całkowitą liczby, kropki dziesiętnej oraz ciągu cyfr oznaczających część ułamkową liczby. Wszystkie te elementy są obowiązkowe. Zero wiodące dozwolone jest jedynie gdy jest jedyną cyfrą części całkowitej.

Liczba całkowita składa się z ciągu cyfr. Zero wiodące dozwolone jest jedynie gdy jest jedyną cyfrą w zapisie liczby.

Ponadto w kodzie źródłowym mogą pojawiać się również **spacje, tabulacje i znaki przejścia do nowej linii**. Są one ignorowane (ale rozdzielają sąsiadujące z nimi elementy).

Język dopuszcza również **komentarze** rozpoczynające się znakami // i kończące znakiem przejścia do nowej linii oraz **napisy** czyli ciągi znaków ujęte w cudzysłowy (napisy nie mogą zawierać znaku przejścia do nowej linii).
Komentarze są ignorowane (ale rozdzielają sąsiadujące z nimi elementy). Napisy mogą wystąpić jedynie w instrukcji write.

Inne znaki są w kodzie źródłowym (poza komentarzami i napisani) niedozwolone i powinny powodować błąd kompilacji (w komentarzach i napisach dozwolone są dowolne znaki, oprócz przejścia do nowej linii).

2) Program

Program rozpoczyna się od słowa kluczowego **program**, po którym następują ujęte w nawiasy klamrowe ciąg deklaracji i ciąg instrukcji.

Przykład:

```
program
    {
    int a;
    a = 3;
    write a+1;
    }
```

Uwaga: zarówno ciąg deklaracji jak i instrukcji mogą być puste, czyli poprawny jest program

```
program {}
```

3) Deklaracje

Pojedyncza deklaracja składa się z typu, identyfikatora i średnika.
Dozwolone typy to **int** (32-bitowa liczba całkowita ze znakiem), **double** (64-bitowa liczba zmiennopozycyjna) i **bool** (wartość logiczna - **true** lub **false**).
Wartości logiczne reprezentowane są jako liczby całkowite 1 (true) i 0 (false).
Deklaracje nie zawierają inicjalizatorów, ale zmienne inicjowane są niejawnie wartościami zerowymi odpowiedniego typu.

Przykłady:

```
int n;
double wynik;
bool isOK;
```

Uwaga 1: Wszystkie zmienne muszą być zadeklarowane, próba użycia niezadeklarowanej zmiennej powinna powodować błąd kompilacji z komunikatem "undeclared variable"

Uwaga 2: Nazwy zmiennych (identyfikatory) muszą być unikalne, próba ponownej deklaracji zmiennej o takiej samej nazwie jak wcześniej zadeklarowana powinna powodować błąd kompilacji z komunikatem "variable already declared "

4) Instrukcje

Język zawiera 7 instrukcji

A) Instrukcja blokowa to ciąg instrukcji ujętych w nawiasy klamrowe, jest użyteczna tam gdzie składnia języka wymaga pojedynczej instrukcji (np. jako instrukcje wewnętrzne w if lub while), a należy wykonać cały ciąg operacji.

Przykład:

{ read n; i = i+n; }

Uwaga: dozwolona jest instrukcja blokowa z pustym ciągiem instrukcji postaci { }.
B) Instrukcja wyrażeniowa to dowolne wyrażenie, po którym następuje średnik (analogicznie jak w języku C/C++).

Przykłady:

```
n = 1;
x+y;
( a>=0 && a<10 ) || b>3 ;
```

Uwaga: wyrażenia opisane będą w dalszej części dokumentu.

C) Instrukcja warunkowa ma dwie postaci.
Postać "z elsem" rozpoczyna się od słowa kluczowego **if**, po którym następuje ujęte z nawiasy wyrażenie typu bool, po nim instrukcja, a następne słowo kluczowe **else** i kolejna instrukcja.
Postać "bez elsa" jest taka sama z tym, że nie ma słowa **else** i następującej po nim instrukcji.
Działanie instrukcji warunkowej jest standardowe (jeśli warunek jest prawdziwy to wykonywana jest instrukcja następująca bezpośrednio po nim, a jeśli warunek jest fałszywy to wykonywana jest instrukcja po słowie kluczowym **else**, a dla wersji "bez else" nic nie jest wykonywane).

Przykłady:

```
if ( n > k )
    k = k*2;
else
    k - k/2;
if ( x==0 ) { res = 1; return; }
```

D) Instrukcja pętli rozpoczyna się od słowa kluczowego **while**, po którym następuje ujęte z nawiasy wyrażenie typu bool, a po nim instrukcja.
Działanie instrukcji pętli jest standardowe (jeśli warunek jest prawdziwy to wykonywana jest następująca po nim instrukcja, a następnie warunek sprawdzany jest ponownie i tak dalej).

Przykład:

```
n = 0;
while ( n<10 )
    {
    n = n + 1;
    write n;
    }
```

E) Instrukcja wejściowa rozpoczyna się od słowa kluczowego **read**, po którym następuje identyfikator, a po nim średnik.
W wyniku wykonania instrukcji wejściowej zmienna określona występującym i niej identyfikatorem otrzymuje wartość wczytaną ze strumienia wejściowego.

Uwaga 1: Strumień wejściowy standardowo oznacza klawiaturę, ale **MUSI** być prawidłowo obsługiwane przekierowanie strumienia wejściowego na wejście z pliku (jest to niezbędne dla prawidłowego przebiegu testów programu).

Uwaga 2: W zależności od typu zmiennej, do której wczytywane są dane instrukcja wejściowa musi prawidłowo obsługiwać ciągi odpowiadające liczbom całkowitym, liczbom zmiennopozycyjnym lub stałym **true** i **false**. Język nie definiuje zachowania programu dla niewłaściwego ciągu wejściowego.
Uwaga 2a: Elementem liczb zmiennopozycyjnych zawsze jest **kropka** (nigdy przecinek), niezależnie od ustawień systemu Windows.

F) Instrukcja wyjściowa ma dwie postaci.
Pierwsza postać rozpoczyna się od słowa kluczowego **write**, po którym następuje wyrażenie, a po nim średnik. W wyniku wykonania instrukcji wyjściowej do strumienia wyjściowego wypisywana jest wartość wyrażenia.
Druga postać rozpoczyna się od słowa kluczowego **write**, po którym następuje napis, a po nim średnik. W wyniku wykonania instrukcji wyjściowej do strumienia wyjściowego wypisywany jest podany napis. Napis to dowolny ciąg znaków ujęty w cudzysłowy.

Uwaga 1: Strumień wyjściowy standardowo oznacza ekran monitora, ale **MUSI** być prawidłowo obsługiwane przekierowanie strumienia wyjściowego do pliku (jest to **niezbędne** dla prawidłowego przebiegu testów programu).

Uwaga 2: Postać wypisywanych wyników MUSI być dokładnie taka jak w poniższym przykładzie (jest to **niezbędne** dla prawidłowego przebiegu testów programu).

Przykład:
Następujący program

```
program
    {
    int i;
    double d;
    bool b;
    i = 5;
    d = 123.456;
    b = true;
    write i;
    write "\n";
    write d;
    write "\n";
    write b;
    write "\n";
    }
```

Wypisuje na strumień wyjściowy

5
123.456000
True

Czyli dokładnie to samo co następujący fragment kodu w języku C#

```
static void Main()
    {
    int i = 5;
    double d = 123.456;
    bool b = true;
    Console.Write(i);
    Console.Write("\n");
    Console.Write(string.Format(
                    System.Globalization.CultureInfo.InvariantCulture,
                    "{0:0.000000}",d));
    Console.Write("\n");
    Console.Write(b);
    Console.Write("\n");
    }
```

G) Instrukcja powrotu składa się ze słowa kluczowego return i średnika.
Wykonanie instrukcji powrotu powoduje zakończenie programu.

Uwaga: Dotarcie sterowania do nawiasu klamrowego kończącego blok programu automatycznie kończy wykonanie programu, w takim przypadku instrukcja powrotu nie jest konieczna.

5) Wyrażenia

Poniższa tabela opisuje wszystkie operatory występujące w języku i ich podstawowe własności.

Lp.	Grupa - priorytet	Nazwa	Symbol	Łączność
1	unarne	minus unarny	-	prawostronna
		negacja bitowa	~	
		negacja logiczna	!	
		konwersja na int	(int)	
		konwersja na double	(double)	
2	bitowe	suma bitowa	|	lewostronna
		iloczyn bitowy	&	
3	multiplikatywne	mnożenie		lewostronna
		dzielenie		
4	addytywne	dodawanie		lewostronna
		odejmowanie		
5	relacje	równe	==	lewostronna
		różne	!=	
		większe niż	>	
		większe niż lub równe	>=	
		mniejsze niż	<	
		mniejsze niż lub równe	<=	
6	logiczne	suma logiczna	||	lewostronna
		iloczyn logiczna	&&	
7	przypisanie	przypisanie	=	prawostronna

Operatory z jednej grupy mają ten sam priorytet, w tabeli operatory umieszczone są w kolejności od najwyższego priorytetu (operatory unarne) do najniższego (przypisanie).
Uwaga: priorytety są inne niż w powszechnie używanych językach programowania (jest to świadoma decyzja).

1) Operatory unarne
A) Argument minusa unarnego może być typu int lub double, wynik jest takiego samego typu jak argument.
B) Argument negacji bitowej musi być typu int, wynik również jest typu int.
C) Argument negacji logicznej musi być typu bool, wynik również jest typu bool.
D) Argument konwersji na int może być dowolnego typu, wynik jest typu int.
uwaga 1: wynikiem konwersji wartości false jest 0, a true 1.
uwaga 2: konwersja z double na int polega na odrzuceniu części ułamkowej
E) Argument konwersji na double może być dowolnego typu, wynik jest typu double
uwaga 1: wynikiem konwersji wartości false jest 0.0, a true 1.0.
uwaga 2: dozwolona jest również niejawna konwersja z int na double.

Wszystkie operatory unarne umieszcza się przed ich argumentem.
Wszystkie operatory unarne można wielokrotnie powtarzać.

2) Operatory bitowe
Oba argumenty operatorów bitowych muszą być typu int, wynik również jest typu int.

3,4) Operatory multiplikatywne i addytywne
Każdy z argumentów operatorów multiplikatywnych i addytywnych może być typu int lub double. Jeśli oba argumenty są typu int to wynik również jest typu int, w przeciwnym przypadku wynik jest typu double.

5) Relacje
Każdy z argumentów operatorów relacyjnych może być typu int lub double, wynik jest typu bool. Dla operatorów == i != dozwolone są również oba argumenty typu bool (wynik oczywiście też jest typu bool).

6) Operatory logiczne
Każdy z argumentów operatorów logicznych musi być typu bool, wynik jest typu bool.
Operatory logiczne MUSZĄ zostać zrealizowane jako obliczenia skrócone.

7) Przypisanie
Lewym argumentem przypisania musi być zmienna (identyfikator).
Do zmiennej typu double można przypisać wartość wyrażenia typu double lub int.
Do zminennych typu int i bool można przypisać jedynie wartość wyrażenia takiego samego typu jak zmienna.

Ponadto elementem wyrażeń mogą być również nawiasy **( )**.
Ich znaczenie jest standardowe.

Jeśli opisane powyżej w punktach 1-7 zasady dotyczące typów nie są zachowane kompilator powinien sygnalizować błąd.

6) Obsługa błędów

W przypadku bezbłędnej kompilacji program musi kończyć się z kodem wyjścia 0, w przypadku błędów kod wyjścia musi być większy od 0 (jest to niezbędne do testowania).

Błędy w kodzie źródłowym nie mogą powodować zapętlenie ani zawieszenia się kompilatora (coś takiego uniemożliwia automatyczne testowanie i jest poważnym błędem). Niedopuszczalne jest również zakończenie pracy kompilatora zgłoszeniem wyjątku.

Nie wszystkie błędy muszą być wykryte, ale błędny program nie może być uznany za poprawny i nie może być dla niego próby generowania kodu.

Błędy powinny być możliwie dokładnie lokalizowane (powinny być podawane sensowne numery linii). Natomiast opisy błędów składniowych nie muszą być precyzyjne (dozwolone jest nawet proste "syntax error" z numerem linii). Błędy semantyczne jako łatwiejsze do zdiagnozowania powinny być opisane precyzyjniej.

Uwagi i wskazówki

1) Zestaw narzędzi Gardens Point można pobrać ze strony przedmiotu.

2) Wygenerowany za pomocą kompilatora program (wersja exe) oprócz poprawnego działania (co jest oczywiste) musi również przechodzić poprawnie weryfikację za pomocą programu peverify. Aby to się udało w kodzie CIL operacje binarne (multiplikatywne, addytywne i relacyjne) oraz przypisanie muszą mieć oba argumenty tego samego typu, zatem wszędzie tam gdzie ma miejsce niejawna konwersja typów z int na double generator musi wygenerować jawną operację konwersji.

3) Przypominam, że separatorem dziesiętnym w liczbach zawsze jest kropka (nigdy przecinek) niezależnie od ustawień systemu Windows. W osiągnięciu tego efektu może pomóc skorzystanie z System.Globalization.CultureInfo.InvariantCulture w odpowiednim miejscu generowanego kodu (tak, korzystanie z niektórych standardowych elementów frameworka .NET jest nie tylko dozwolone, ale konieczne). Niestety przy korzystaniu z InvariantCulture przecinek uznawany jest za separator tysięcy i w konsekwencji zapis 1,000 przy wczytywaniu z wejścia potraktowany zostanie jako 1000 (tak naprawdę jest jeszcze gorzej: zapis 1,,,2,,3,4.567 potraktowany zostanie jako 1234.567). Nie

należy przejmować się tymi dziwactwami w obsłudze przecinka, natomiast kropka musi być obsługiwana prawidłowo.

4) W przypadku gdy nie wiecie Państwo w jaki sposób skorzystać z elementów .NET pomocne może być napisanie prostego programu w C# korzystającego z tych elementów, a następnie analiza wygenerowanego pliku exe za pomocą programu ildasm.

5) W celu obsługi błędów składniowych należy w odpowiednich miejscach gramatyki dodać produkcje ze sztucznym tokenem error (jest on generowany automatycznie przez parser po wystąpieniu błędu składniowego). Należy również pamiętać o błędach związanych z nieoczekiwanym osiągnięciem końca pliku (powinna być wtedy wykonana akcja specjalna YYABORT)

6) W gramatyce może pojawić się jedynie konflikt shift-reduce związany z instrukcją if (z elsem lub bez) oraz konflikty shift-reduce związane z produkcjami dotyczącymi obsługi błędów (ze sztucznym tokenem error), inne konflikty są niedozwolone.

7) Wymagane priorytety i łączność operatorów muszą wynikać wprost z produkcji gramatyki, a nie z jakichś sztuczek dostępnych często w generatorach parserów (nigdy z takich sztuczek nie korzystam, więc nawet nie wiem czy i jakie są dostępne w Gardens Point).

8) O tym co jest usterką niewielką (powodującą obniżenie oceny do 3.5 lub 3), a co usterką poważną (powodującą niezaliczenie przedmiotu) będę każdorazowo decydował indywidualnie.
Ale dla uniknięcia rozczarowań zwracam uwagę na błędy, które być może uważacie Państwo za drobiazgi, a będą traktowane jako bardzo poważne usterki powodujące automatyczne niezaliczenie przedmiotu

a) Brak możliwości przekierowywania standardowego wejścia i wyjścia z klawiatury/ekranu monitora na wejście/wyjście z/do pliku uniemożliwi przeprowadzenie automatycznych testów i oznacza niezaliczenie przedmiotu (na szczęście trzeba się specjalnie postarać aby popełnić ten błąd, standardowo wszystko działa dobrze, ale w latach poprzednich zdarzył się student, który się postarał). Komunikaty o błędach należy wypisywać na standardowe wyjście (nie na wyjście błędów) i one również muszą dać się przekierować.

b) Jakiekolwiek błędy w wypisywaniu wyników - programy będę testowane automatycznie, generowane wyniki będą porównywane z oczekiwanymi na zasadzie badania identyczności plików, dlatego jakiekolwiek odstępstwo w działaniu instrukcji write w stosunku do działania opisanego w punkcie 4F (nawet np. nadmiarowa spacja) spowoduje, że wygenerowanie pliki nie będą jednakowe i program zostanie odrzucony. Powyższa zasada dotyczy wyników wypisywanych przez wygenerowany program, komunikaty o błędach kompilacji nie są zestandaryzowane i oczywiście mogą się różnić.

c) Operatory logiczne muszą być zaimplementowane jako obliczenia skrócone, zaimplementowanie ich jako obliczenia pełne oznacza niezaliczenie przedmiotu.

d) W przypadku poprawnej kompilacji kod wyjścia z programu (wartość zwrócona z metody Main) musi wynosić 0, a w przypadku błędów być większy od 0 (jest to niezbędne do automatycznego testowania).

e) Niedozwolone jest zatrzymywanie wykonania programu (np. ReadKey, ReadLine i inne takie). Program czekający na naciśniecie klawisza zostanie przez automatyczne testy potraktowany jako program, który się zawiesił!

Oczywiście to nie jest pełna lista poważnych usterek powodujących niezaliczenie przedmiotu. Przypominam również, że pozostawienie w programie wydruków kontrolnych, a w kodzie resztek nieudanych eksperymentów w postaci zakomentowanych fragmentów śmieciowego kodu jest objawem wyjątkowego niechlujstwa.