Podstawy języka C++, część druga

Na poprzednich zajęciach przypomniane zostały zagadnienia: podstawowych operacji wejścia-wyjścia, deklarowania zmiennych w programie, podstawowych typów danych oraz instrukcja warunkowa *if*. Dzisiaj zapoznamy się z pętlami oraz zagadnieniami, dotyczącymi funkcji.

Przeanalizujmy poniższy program (plik *parametry.cpp*)

```
#include<iostream>
using namespace std;

int main(int argc, char *argv[])
{
   cout<<"Program wypisuje wszystkie parametry wywołania siebie:"<<endl;
   cout<<"System poinformował, że liczba parametrów wywołania wynosi"<<argc<<endl;

   cout<<endl<<"Wypiszemy teraz w pętlach te parametry..."<<endl<<endl;

   cout << "Pętla \"for\" :" <<endl;
   int i;
   for(i=1; i<=argc; i=i+1)
   {
                cout << "Argument nr " << i << " : ";
                cout << argv[i-1] <<endl;
   }
   cout<<endl;

   cout<<"Pętla \"while\" :"<<endl;
   i=1;
   while(i<=argc)
   {
                cout << "Argument nr " << i << " : ";
                cout << argv[i-1] <<endl;
                i++;
   }
   cout<<endl;

   cout<<"Pętla \"do ... while\" :"<<endl;
   i=1;
   do
   {
                cout << "Argument nr " << i << " : ";
                cout << argv[i-1] <<endl;
                i++;
   }
   while(i<=argc);
   cout<<endl;
}
```

W programie zostały użyte trzy pętle, każda z nich realizuje wyświetlenie wszystkich argumentów wywołania programu (czyli tekstów, wpisanych w linii poleceń, po nazwie pliku, zawierającego uruchamiany program). Omówmy je pokrótce:

- pętla *for* – wewnątrz nawiasów zawiera trzy wyrażenia, opisujące jej działanie:

1. pierwsze wyrażenie wykonuje zadania poprzedzające kroki pętli (w programie następuje przypisanie zmiennej *i* wartości 1).
2. po średniku umieszcza się wyrażenie, warunkujące wykonanie kolejnego kroku pętli, czyli instrukcji (w szczególności blokowej) znajdującej się za nawiasem ")". W naszym przypadku mamy do czynienia z instrukcją blokową (instrukcje ujęte w nawiasy klamrowe *{ }*). W bloku mogą znajdować się również definicje zmiennych, mają one zasięg tego bloku – zmienne istnieją

tylko w obrębie bloku).

3. po drugim ze średników znajduje się instrukcja, dokonująca modyfikacji zmiennych sterujących pętlą (w naszym przypadku zwiększenie zmiennej *i* o jeden).

Kilka uwag ogólnych. Pętla *for* stosowana jest najczęściej w przypadkach, gdy zdeterminowana jest z góry liczba kroków pętli, jakie należy wykonać. W naszym przypadku w momencie uruchomienia programu wiadomo, ile parametrów otrzymał on od użytkownika z linii komend – ilość tych parametrów zawarta jest w zmiennej *argc*. W deklaracji pętli *for* można opuścić każde z trzech wyrażeń, określających jej działanie – brak pierwszego i trzeciego oznacza brak jakichkolwiek działań, brak drugiego oznacza wyrażenie zawsze prawdziwe. Innymi słowy, pętla bez wyrażenia warunkowego wykonuje się nieskończoną ilość razy (wewnątrz pętli można umieścić instrukcję „*break*", która spowoduje jej opuszczenie).

- pętle *while()* i *do ... while()* zasadniczo wykonują to samo – jeśli spełniony jest warunek, umieszczony w nawiasach, wykonują kolejny krok pętli.

Petle te różni moment sprawdzania warunku – pętla *while* sprawdza warunek *przed* wykonaniem instrukcji, zaś pętla *do ... while() po* jej wykonaniu. Rezultat jest taki, że pętla *do ... while()* zawsze wykona co najmniej jeden krok, podczas gdy pętla *while()* może nie wykonać ani jednego kroku. Obie te pętle używane są zasadniczo w sytuacjach, gdy podczas kroków pętli zmieniają się wartości wyrażeń, warunkujących ich kontynuację, czyli gdy nie jest znana z góry liczba kroków pętli. Proszę zauważyć, że w programie w treści pętli następuje zmiana wartości zmiennej *i* (sposób zostanie omówiony później).

Wewnątrz pętli w programie pojawia się instrukcja

```
cout << argv[i-1] <<endl;
```

powodująca wypisanie argumentu wywołania programu, *i*-tego w kolejności. Parametr *argv* jest tzw. tablicą wskaźników do tekstu (wynika to z deklaracji funkcji *main*, szczegóły nie są nam na razie potrzebne). Ważne jest to, że w językach *C* i *C++* elementy tablic numeruje się zawsze od zera, stąd odejmowanie liczby 1 od indeksu pozycji odczytywanej z tablicy. Aby odczytać wartość z komórki tablicy stosuje się wyrażenie, w którym na początku stoi nazwa tablicy, zaś po niej umieszczony jest indeks ("numer pozycji"), zamknięty w nawias klamrowy.

W programie pojawia się instrukcja ***i++;*** oznaczająca (w uproszczeniu) tyle samo, co **i=i+1;** w efekcie której zmienna *i* zostanie zwiększona o jeden. Istnieją pewne osobliwości, związane z operatorem ++, ale nimi zajmiemy się, omawiając dokładniej operatory języka C++.

Zadanie

Pobrać z Internetu plik *parametry.cpp*, następnie skompilować go i uruchomić, podając komendy:

```
./parametry
./parametry to
./parametry to jest test
```

Zaobserwować zmiany w działaniu programu. Następnie napisać program, który wypisze w linii 15 gwiazdek, za pomocą każdej z trzech pętli.

Zagadnieniem funkcji zajmiemy się przy okazji omawiania kolejnego programu (plik *funkcje.cpp*)

```
#include<iostream>
using namespace std;

// Prototypy funkcji
```

```
void powitanie(void);
double pobierz(void);
double wielomian(double);

int main()
{
double argument;

   powitanie();
   argument=pobierz();
   cout<< "Wartość wielomianu w punkcie " << argument << " wynosi ";
   cout<< wielomian(argument) << endl;
}

void powitanie(void)
{
   cout << "Program wylicza wartość wielomianu y=x^2+2x+1"<<endl;
   cout << "Za chwilę zostaniesz zapytany o wartość argumentu" <<endl<<endl;
}

double pobierz(void)
{
double wartosc;

   do
   {
                cout<<"Podaj wartość x (ale nie podawaj 0, bo to nudne...):";
                cin>>wartosc;
   } while (wartosc == 0);
   cout<<endl;
   return wartosc;
}

double wielomian(double x)
{
double wynik;

   wynik = x*x;
   wynik += 2*x;
   return wynik + 1;
}
```

Po wstawieniu pliku nagłówkowego pojawiają się linie, zbliżone do nagłówków definicji funkcji. Są to tzw. *prototypy funkcji* (będące deklaracjami funkcji), informujące kompilator, jaki jest typ wartości zwracany przez funkcję, oraz jakie argumenty przyjmuje funkcja. Jakkolwiek użycia prototypów można w pewnych sytuacjach uniknąć, ich stosowanie jest pożyteczne i zalecane. W prototypach funkcji można pominąć nazwy argumentów (dla kompilatora istotne są jedynie typy poszczególnych argumentów i ich kolejność, ale warto pozostawić nazwy zmiennych, celem zilustrowania, jakich informacji potrzebuje funkcja). W tekście programu pojawia się słowo kluczowe *void*, oznaczające, że funkcja nie zwraca żadnej wartości (jeśli oznacza typ wartości zwracanej przez funkcję), lub że funkcja nie przyjmuje żadnych argumentów (gdy umieszczone jest wewnątrz nawiasów deklaracji funkcji).

W funkcji *main* wykorzystujemy zdefiniowane przez nas funkcje na trzy sposoby:

- wywołujemy funkcję *powitanie*, która nie zwraca żadnej wartości, pisząc instrukcję: **powitanie();**
- funkcja *pobierz* wywołana jest w celu przypisania wartości zwróconej do zmiennej *argument* za pomocą instrukcji: **argument=pobierz();** (funkcja *pobierz* wczytuje od użytkownika wartość argumentu z klawiatury).

- funkcja może zostać użyta w wyrażeniu algebraicznym, albo innym miejscu, gdzie potrzebna jest wartość, zwracanego przez nią typu, wtedy wywołujemy ją zapisem bez średnika kończącego instrukcję, czyli np. **pobierz()**, średnik kończy dopiero całe wyrażenie, w którym występuje wywołanie funkcji.

Zajmijmy się teraz sposobem definiowania funkcji. Definicja rozpoczyna się zadeklarowaniem typu wartości zwracanej przez funkcję, następnie zawiera jej nazwę oraz opcjonalną listę deklaracji argumentów funkcji, rozdzielanych przecinkami, umieszczoną w obowiązkowych nawiasach okrągłych. Następnie należy umieścić instrukcję blokową, zawierającą treść funkcji, na którą składają się definicje zmiennych lokalnych oraz wszystkie instrukcje funkcji. Istotny jest sposób zwracania wartości przez funkcję. Jeśli funkcja zwraca jakąś wartość (typem wartości zwracanej nie jest *void*), czyni to za pomocą instrukcji **return wyrażenie;** przy czym wyrażenie powinno dać wartość typu zwracanego przez funkcję (lub typu pozwalającego na niejawną konwersję do typu zwracanego przez funkcję). Jeśli chcemy zakończyć działanie funkcji nie zwracającej żadnej wartości (typu *void*), należy użyć instrukcji **return;** przy czym po słowie kluczowym *return* nie może pojawić się w takiej sytuacji wyrażenie określające wartość zwracaną. Funkcja zakończy się również po osiągnięciu nawiasu klamrowego zamykającego jej definicję, jest to jednak błędem, jeśli jest zadeklarowana jako funkcja zwracająca wartość. Jedynym wyjątkiem jest funkcja *main*, w której pominięcie jawnego zwrócenia wartości jest dopuszczalne i równoważne użyciu instrukcji *return 0;* oznaczającej poprawne zakończenie programu.

W ostatniej z funkcji pojawia się operator +=. Jego działanie wyjaśnię przez analogię:

```
a=a+b;
a+=b;
```

Obie powyższe instrukcje mają ten sam skutek - wartość zmiennej *a* zostaje powiększona o wartość zmiennej *b*.

Zadanie

Pobrać i uruchomić program powyższy program. Następnie samodzielnie napisać program, który zsumuje zadaną liczbę początkowych liczb naturalnych. Sumowanie proszę zrealizować jako funkcję, długość sumowanego ciągu użytkownik podaje z klawiatury.

Na następne zajęcia proszę zapoznać się z rozdziałem piątym z książki „C++ dla każdego”, autorstwa Jesse Liberty.