Podstawy języka C++, powtórzenie

Powtórzmy pokrótce zagadnienia, którymi zajmowaliśmy się do tej pory na zajęciach. Poznawanie języka C++ rozpoczęliśmy od analizy programu *hello.cpp*. Pojawiły się tam następujące elementy:

- Dwa typy komentarzy: /* */ oraz // . Pierwszy typ umożliwia wstawienie komentarza dowolnej długości, zawartego wewnątrz znaczników, drugi powoduje potraktowanie jako komentarza tekstu znajdującego się za // do końca bieżącej linii. Zostało powiedziane, że komentarz ma znaczenie wyłącznie dla osoby, czytającej tekst programu.
- dyrektywa *#include* , służąca do włączenia do tekstu zawartości zadanego pliku. Stosujemy ją zwykle do włączenia tzw. plików bibliotecznych np. *iostream* do tekstu programu za pomocą dyrektywy *#include <iostream>*
- główna funkcja programu o nazwie *main*, z listą argumentów, pozwalającą na pobranie parametrów wywołania, udostępnianych przez system operacyjny. Parametry te były jednak ignorowane.
- użycie strumienia wyjściowego za pośrednictwem obiektu *cout* do wyprowadzenie stałego tekstu. Przy tej okazji poznaliśmy sposób zapisu stałej tekstowej w języku C++ - ciąg znaków, ujęty w znaki cudzysłowu - „ „

Na przykładzie programu zapisanego w pliku *hello.cpp* prześledziliśmy sposób przygotowania i uruchomienia programu dla systemu operacyjnego Linux. Tok postępowania przedstawia się nastepująco:

- Przygotowanie tekstu programu w pliku (lub plikach – może być ich więcej) o odpowiednich nazwach – w naszym przypadku po nazwie pojawia się kropka i „cpp”
- Kompilacja programu, w naszym przypadku **g++ -o plik_wynikowy plik.cpp**
- W przypadku prawidłowej kompilacji można uruchomić program, wydając polecenie **./ plik_wynikowy**, w przeciwnym wypadku należy poprawić błędy w tekście programu i powtórnie dokonać próby kompilacji.

Przy omawianiu kolejnego programu zwróciliśmy uwagę na następujące zagadnienia:

- Konieczność definiowania zmiennych, używanych w programie, poprzez użycie instrukcji *nazwa_typu nazwa_zmiennej;* Ponadto poznaliśmy cztery najbardziej podstawowe typy danych języka C++: *int*, *char*, *float* i *double*. Definicja zmiennej jest potrzebna kompilatorowi, gdyż określa ona, jak należy postępować ze zmienną (np. jak realizowane są operatory) oraz informuje o ilości pamięci, jaką trzeba przeznaczyć na zmienną.
- Poznaliśmy sposób pobierania danych ze strumienia wejściowego, poprzez obiekt *cin*, oraz możliwość „kaskadowych” operacji na strumieniach za pomocą instrukcji postaci *cout << ”A” << endl;*
- Omówiona została instrukcja warunkowa *if(warunek) instrukcja1; else instrukcja2;* o działaniu, polegającym na sprawdzeniu prawdziwości *warunku*, i w przypadku jego spełnienia wykonaniu *instrukcji1*, a w przeciwnym wypadku wykonaniu *instrukcji2* (jeśli istnieje część ze słowem *else* i instrukcja do wykonania). Ponadto omówiono przykład tworzenia warunku złożonego za pomocą operatora *&&* (logiczne *i* - *AND*)

Drugie zajęcia dotyczyły zagadnień pętli oraz tworzenia i użytkowania funkcji. Oba te zagadnienia są kluczowe w programowaniu w językach strukturalnych.

W języku C++ istnieją trzy rodzaje pętli. Są to pętle: *for()*, *while()* oraz *do ... while()*. Pętle

stosuje się, aby programować cykliczne operacje. Dla przykładu, zadanie polegające na piętnastokrotnym wypisaniu znaku * w wierszu można zaprogramować następująco:

```
int i;      // potrzebujemy zmiennej indeksującej kroki pętli
for(i=1; i<=15; i++) cout<<"*";
cout<<endl;
```

Przeanalizujmy ten fragment programu. Zadanie „wypisz piętnaście gwiazdek w linii" sprowadza się do wypisania piętnaście razy „jednej gwiazdki", realizowanego za pomocą instrukcji *cout<<"*";* a następnie przeniesienia kursora do nowego wiersza. Zadaniem pętli jest zorganizowanie piętnastokrotnego wywołania tej instrukcji, po czym należy przenieść kursor do nowego wiersza, co kończy wykonywanie zadania. Pętla *for* zawiera trzy wyrażenia, definiujące sposób jej działania. Pierwsze wyrażenie w nawiasie jest wykonywane jednokrotnie – przed pierwszym krokiem pętli – zazwyczaj wykorzystuje się je do inicjalizacji wartości zmiennej sterującej pętli, w naszym przypadku zmiennej *i*. Chcemy, aby przed wykonaniem pierwszego kroku pętli jej zmienna sterująca *i* miała wartość 1 – będzie ona określała, którą gwiazdkę właśnie mamy wypisać. Środkowe wyrażenie w pętli *for* to warunek na wykonanie kroku pętli, sprawdzany przed próbą wykonania każdego kroku, włącznie z pierwszym. Ponieważ chcemy wypisać 15 gwiazdek, musimy sprawdzać, czy *i* jest mniejsze od 16 (lub równoważnie, czy jest mniejsze lub równe 15). Po pomyślnym sprawdzeniu warunku wykonywany jest krok pętli (w naszym przypadku instrukcja wypisująca "gwiazdkę"), zaś po każdym kroku wykonywana jest instrukcja, zawarta po drugim średniku w pętli *for* – tam następuje zwykle uaktualnienie zmiennej sterującej pętli. Doliczamy tu wypisaną gwiazdkę (operatorem inkrementacji ++).

Jakie czynności realizowane są w omawianym fragmencie programu?

- Przypisujemy zmiennej *i* wartość 1 (*i=1*)
- Sprawdzany jest warunek: *i<=15* (*1<=15*) – warunek jest spełniony, więc wypisywana jest gwiazdka (*cout<<"*";*), zatem do tej pory wypisaliśmy jedną gwiazdkę, a zmienna *i* ma wartość 1.
- operator inkrementacji dokonuje zwiększenia o 1 wartości zmiennej *i* (obecnie jest ona równa 2).
- następuje ponowne sprawdzenie warunku: *i<=15;* warunek jest prawdziwy, więc wypisywana jest kolejna gwiazdka itd.
- Załóżmy, że właśnie została wypisana piętnasta gwiazdka (po wykonaniu wszystkich potrzebnych do tego celu kroków pętli) i wartość zmiennej *i* wynosi 15.
- Następuje inkrementacja zmiennej *i* do wartości 16.
- Sprawdzany jest warunek na wykonanie kroku pętli: *i<=15;* warunek jest fałszywy (bo *16>15*), więc pętla *for* kończy swoje działanie.

Pętle *while()* i *do ... while()* różnią się od pętli *for* tym, że musimy zaprogramować definicję oraz inicjalizację wartości zmiennej sterującej przed rozpoczęciem pętli, oraz uaktualnianie tej zmiennej w kroku pętli. Obie te pętle realizują wyłącznie sprawdzanie warunku na wykonanie kolejnego kroku – pętla *for* sprawdza go swoim drugim wyrażeniem (wyrażeniem po pierwszym średniku). "Piętnaście gwiazdek" za pomocą pętli *while()* :

```
// zakładam, że zmienna i jest zadeklarowana "gdzieś wcześniej"
i=1;      // wpisujemy na początek 1 – w zmiennej obecnie może być coś innego !!!
while(i<=15)      // warunek na wypisanie kolejnej gwiazdki
{
        cout<<"*";      // * na ekran!
        i++;            // zwiększ licznik gwiazdek o jeden
}
```

```
cout<<endl;       // na koniec... koniec wiersza
```

Rozważania są praktycznie te same, jak przy realizacji zadania za pomocą pętli *for*, z tą różnicą, że inkrementacja zmiennej *i* nie jest wydzielona od czynności wypisywania gwiazdki – stanowią jeden blok zadań do wykonania w kroku pętli, zaś początkowe przypisanie *i=1;* nie stanowi części pętli, lecz jest realizowane przed jej rozpoczęciem. Analogicznie przy pętli *do ... while()* (jedyna różnica dotyczy sprawdzania warunku na wykonanie kolejnego kroku *po* wykonaniu zaprogramowanych czynności):

```
i=1;
do
{
        cout<<"*";
        i++;
}
while(i<=15);
cout<<endl;
```

Ostatnim z zagadnień jest definiowanie i używanie funkcji. Na przykładzie funkcji *main* wiedzą Państwo, że schemat definicji jest następujący: typ wartości zwracanej, nazwa oraz opcjonalna lista deklaracji argumentów, rozdzielonych przecinkami, ujęta w obowiązkowe nawiasy okrągłe *( )*, po których następuje treść funkcji (w książkach można się spotkać z określeniem *ciało funkcji*) w bloku ujętym w nawiasy klamrowe *{ }*. Dla funkcji, innych niż *main,* zazwyczaj pisze się *prototypy funkcji*, będące deklaracjami funkcji. Na prototyp funkcji składa się deklaracja typu zwracanego, nazwy funkcji i listy typów argumentów (z możliwością wpisania lub pominięcia ich nazw), ujętej w nawiasy, prototyp kończy się średnikiem. Omówione zostały trzy typowe sposoby użycia funkcji:

- wywołanie jako instrukcji – bez wykorzystania zwróconej wartości.
- wykorzystanie w wyrażeniu algebraicznym – wartość funkcji wykorzystana jest do wyznaczenia wartości wyrażenia algebraicznego np. *a=pobierz()+2;*
- trywialny przypadek poprzedniego podpunktu – wartość funkcji zostaje przypisana zmiennej np. *i=sin(0);*

Jako ostatnie zostało omówione zagadnienie zwracania wartości przez funkcję przy pomocy instrukcji *return*, z wyrażeniem na zwracaną wartość (*return 0;*) lub bez niego (*return;*).

Zadania

- Napisać program, który przelicza jednostki długości z układu metrycznego (milimetry) na calowy i odwrotnie. Program po uruchomieniu pyta, w jakich jednostkach podana zostanie długość (wczytać liczbę całkowitą, 1 - układ metryczny, 2 - układ calowy, jeśli podano inną, powtórzyć wczytywanie), następnie pyta o tę długość, wyznacza wynik i wyświetla go na ekranie. Cal ma 25,4mm.
- Napisać program, który podaje (naturalną) potęgę zadanej liczby rzeczywistej. Wyliczanie potęgi zrealizować jako funkcję. Przy pobieraniu argumentów sprawdzić należy, czy podana przez użytkownika potęga jest dodatnia. Przy pisaniu funkcji skorzystać należy z mnożenia w pętli.