Pobieranie i wyprowadzanie danych

Napisz program, który zaczeka na wprowadzenie jakiejś liczby i potem wypisze ją na ekranie.

```
#include <iostream>
using namespace std;
int main()
{
int a;
cin >> a;     // pobranie liczby i wpisanie jej do zmiennej a
cout << a;    // wyprowadzenie pobranej liczby
char c;       // oczekiwanie na naciśnięcie jakiegoś klawisza
cin >> c;
return 0;
}
```

W pierwszej linii zapoznajemy kompilator ze strumieniami standardowego wejścia cin i wyjścia cout.
W drugiej linii udostępniamy programowi algorytmy *biblioteki standardowej* współczesnego języka C++.
Widoczna dalej funkcja main() jest koniecznym elementem programu konsolowego, napisanego w języku C++. W funkcji tej deklarujemy zmienną roboczą o nazwie a, wczytujemy do niej numeryczną wartość z klawiatury i wypisujemy ją na ekranie.
Końcowe linie mają znaczenie techniczne — realizują oczekiwanie na wciśnięcie klawisza, zapobiegając tym samym natychmiastowemu zamknięciu okienka konsoli przez system operacyjny, gdy tylko program dobiegnie końca

Wyprowadzanie napisów na ekran

Napisz program, który uprzejmie zapyta o jakąś liczbę i potem uprzejmie wypisze ją na ekranie.

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;
int main()
{
int a;
cout << "Podaj jakas liczbe: ";
cin >> a;
cout << "Podales " << a;
system("PAUSE");
return 0;
}
```

Przed zapytaniem o liczbę wyprowadzamy na ekran tekst zachęty, tak aby użytkownik wiedział, czego program od niego oczekuje.
Potem następuje znana już fraza pobierania liczby z klawiatury, a następnie wyprowadzanie na ekran nie tylko liczby, ale i stosownego komentarza.
Zauważmy tam bardzo przydatny szczegół — tak zwaną *kaskadowość* strumienia wyjściowego, polegającą na kolejnym kierowaniu do strumienia cout kilku wartości.

Pobieranie i wyprowadzanie większej ilości danych

Napisz program, który uprzejmie zapyta o trzy liczby i uprzejmie wypisze je na ekranie.

```
#include <iostream>
using namespace std;
int main()
{
int a1, a2, a3;
cout << "Podaj trzy liczby: ";
cin >> a1 >> a2 >> a3;
cout << "Podales " << a1 << ", " << a2 << " i " << a3;
system("PAUSE");
return 0;
}
```

Przykład powyższy ilustruje ważną i przydatną cechę strumieni — możliwość kaskadowego pobierania danych z klawiatury i równie kaskadowego wyprowadzania informacji na ekran

1. Dane w sekwencji (kaskadzie) wejściowej muszą być oddzielone tzw. białymi znakami — *spacją*, *tabulatorem*, *enterem*. Ilość takich rozdzielników, kończących wprowadzanie jednej zmiennej i zaczynających wczytywanie następnej, jest nieistotna. Zasada ta oznacza, że po wprowadzeniu pierwszej jedynki w poprzednim przykładzie należy nacisnąć dowolną ilość razy jakiś klawisz białego znaku, po czym wprowadzić kolejną wartość.
2. Całą sekwencję wprowadzanych danych kończymy klawiszem *Enter*. Program opuszcza instrukcję wczytywania i przechodzi do następnej instrukcji.
3. Typy zmiennych, które oczekują na wartości napływające ze strumienia cin, określają, jakich klawiszy nie wolno naciskać, ponieważ wtedy wprowadzanie danych się nie powiedzie. Jeśli w poprzednim przykładzie zmienne a1, a2, a3 były typu int (czyli całkowitego), należało wciskać klawisze numeryczne.

Przejście do nowej linii

Tak zmodyfikuj ostatni program, aby trzy pobrane z klawiatury wartości zostały wyprowadzone w trzech oddzielnych liniach:

```
#include <iostream>
using namespace std;
int main()
{
int a1, a2, a3;
cout << "Podaj trzy liczby: ";
cin >> a1 >> a2 >> a3;
cout << "Pierwsza liczba: " << a1 << endl;
cout << "Druga liczba: " << a2 << endl;
cout << "Trzecia liczba: " << a3;
char c;
cin >> c;
return 0;
```

```
}
```

W powyższym programie w linii kaskadowego wypisywania danych należy zauważyć wtrącenia stałej endl.
Wartość tej stałej jest uzgodniona ze strumieniem i oznacza *zmień linię!* (endl — *end of line*). Zapamiętaj to!

Ustalanie szerokości pola danej

Tak zmodyfikuj ostatni program, aby wyprowadzane teksty i następujące po nich wartości układały się w czytelne kolumny:

```
#include <iostream>
using namespace std;
int main()
{
int a1, a2, a3;
cout << "Podaj trzy liczby: ";
cin >> a1 >> a2 >> a3;
cout.width( 20);
cout << "Pierwsza liczba: ";
cout.width( 10);
cout << a1 << endl;
cout.width( 20);
cout << "Druga liczba: ";
cout.width( 10);
cout << a2 << endl;
cout.width( 20);
cout << "Trzecia liczba: ";
cout.width( 10);
cout << a3;
char c;
cin >> c;
```

```
return 0;
}
```

Ustalanie szerokości pola przeznaczonego na zmienną wymaga wywołania należącej do strumienia funkcji width(), czyli „szerokości”. Argumentem tej funkcji jest właśnie szerokość pola.
Funkcja należy do obiektu strumienia, dlatego sięgamy po nią za pomocą charakterystycznej dla języków obiektowych frazy cout.width() dającej się wypowiedzieć jako „uruchom algorytm width() należący do obiektu cout!”.
Zauważmy, że funkcję width() wywołujemy przed każdym wyprowadzaniem z ustalaną szerokością pola.

Nadmiar pola o szerokości zdefiniowanej funkcją width() nie musi być wypełniany spacjami. Możemy zdecydować, że będzie to jakiś inny znak.
Częsty błąd: *zapomnienie, że wywoływanie funkcji ustalającej szerokość pola należy powtarzać przed każdym wypisaniem danej do strumienia wyjściowego.*

Wypełnianie pola o nadmiernej szerokości

Tak zmodyfikuj ostatni program, aby nadmiar szerokości pól był wypełniany kropkami:

```
#include <iostream>
using namespace std;
int main()
{
int a1, a2, a3;
cout << "Podaj trzy liczby: ";
cin >> a1 >> a2 >> a3;
cout.fill( '.');
cout.width( 20);
cout << "Pierwsza liczba: ";
... // dalej bez zmian
```

Funkcja fill(), czyli „wypełnij”, także należąca do obiektu strumienia wyjściowego, jest wywoływana z argumentem znakowym — jest to ten znak, którym zostanie wypełniony nadmiar pola określonego funkcją width().
W przeciwieństwie do funkcji width(), którą należało wykonywać przed każdym wyprowadzaniem zmiennej, strumień zapamiętuje ustalenie dokonane za pomocą funkcji fill().
Wypełnianie nadmiarowej szerokości określonym znakiem obowiązuje aż do ponownego uruchomienia funkcji fill() z jakimś innym znakiem.

Trzecim po width() i fill() elementem formatowania wyprowadzanej informacji jest funkcja precision(), czyli „ustal liczbę miejsc po przecinku”.
Funkcja ta — podobnie jak fill(), ale nie jak width() — wprowadza regułę obowiązującą do końca życia programu albo do kolejnego jej wywołania, zmieniającego precyzję.
Funkcja ta także należy do obiektu reprezentującego wyjście na ekran konsoli cout.

Wyprowadzanie danych numerycznych

Niech kolejny program pobierze ze standardowego wejścia liczbę rzeczywistą i wypisze jej odwrotność, ograniczając liczbę miejsc po przecinku do dwóch:

```
#include <iostream>
using namespace std;
int main()
{
double a;
cout << "Podaj liczbe rzeczywista (z kropka) ";
cin >> a; // pobranie liczby i wpisanie jej do zmiennej a
cout.precision( 2);
cout << "1 / " << a << " = " << 1/a << endl;
char c; // oczekiwanie na naciśnięcie jakiegoś klawisza
cin >> c;
return 0;
}
```

Częsty błąd: *wywoływanie* precision() *lub* fill()*, mimo że ciągle obowiązują ustalenia dokonane za pomocą wcześniejszych wywołań tych funkcji. Nie wywołuj tych funkcji niepotrzebnie!*