Obiektowe wejście/wyjście w C++

Funkcje z rodziny printf(…) nie są zbyt bezpieczne w użyciu. Drobna i trudna do wykrycia literówka może powodować błędne wyświetlanie wyników (zwłaszcza wtedy, gdy pomyłka polega na użyciu %d zamiast %g, %f lub na odwrót). Poważniejsze w skutkach jest użycie zbyt dużej liczby elementów % w ciągu formatujących, co przy niedoborze pozostałych argumentów funkcji printf spowoduje najczęściej szybkie uśmiercenie aplikacji przez system operacyjny w wyniku błędu ochrony pamięci.

Jeśli jest to możliwe, warto zamiast printf() nauczyć się używać nowszego i bezpieczniejszego sposobu użycia stdio, a są nim obiekty cout oraz cin. Niekoniecznie jednak należy za wszelką cenę rezygnować z printf(…) – każde z tych rozwiązań mają swoje wady i zalety, poznawszy oba, będziesz mógł świadomie wybrać optymalne z nich.

Nie polecam używania w jednym programie obu tych mechanizmów naprzemiennie, może to doprowadzić do zaskakujących rezultatów wynikających z wewnętrznego buforowania danych przeznaczonych do wyświetlenia w konsoli.

```
#include <iostream>
void main()
{
	std::cout << "Hello, world!" << std::endl;
}
```

Powyższy kod to najprostszy przykład użycia obiektu cout, który jest obiektowym odpowiednikiem stdout. Dla wygody i uproszenia kodu warto domyślnie włączyć przestrzeń nazw (namespace) o identyfikatorze std. Pozwoli to pomijać przedrostek std::, a kod uprości się do następującej postaci:

```
#include <iostream>
using namespace std;
void main()
{
	cout << "Hello, world!" << endl;
}
```

Symbol endl oznacza znak końca wiersza, do tej pory używaliśmy znaku ‘\n’, rzadziej dwuznakowej kombinacji \r\n. Użycie endl nie jest konieczne, ale stanowi dobry zwyczaj polepszający przenośność kodu między różnymi systemami operacyjnymi, które w różny sposób kodują znaki końca wierszy w plikach tekstowych.

Moglibyśmy zatem posłużyć się także zapisem: cout << "Hello, world!\n";

Operator << w języku C oznaczał przesunięcie bitowe pewnej liczby w lewo o zadaną liczbę bitów, np. `printf("%d %d %d %d %d %d\n\n", 1<<0, 1<<1, 1<<2, 1<<3, 1<<4, 1<<5);`

Wyświetlone zostaną wyniki: 1 2 4 8 16 32, czyli kolejne potęgi liczby 2.

Dlaczego? Zastanów się, jak wygląda dwójkowa reprezentacja liczby całkowitej 1, oraz co się stanie, gdy ta jedynka zacznie „wędrować” w lewo.

Zadanie

1. Sprawdź, jak działają operatory >> oraz << dla zmiennych oraz liczb różnych typów. Użyj zmiennych typu int, char, unsigned int, unsigned char, double. Posłuż się wartościami początkowymi -1, +1, 2, 4, 255, MAX_INT. W tym zadaniu NIE chodzi o to, aby posługiwać się `cout << x;` czy `cin >> y;`. Celem zadania jest użycie operatorów << oraz >> w odniesieniu do operacji bitowych na liczbach dwójkowych. Wyniki tych działań możesz wyświetlić w dowolny sposób, na przykład tak jak to zostało zademonstrowane kilka akapitów wcześniej.

W języku C++ dodano możliwość zmiany sposobu działania niektórych operatorów, w szczególności dotyczy to właśnie operatora <<. Graficznie przypomina on symbol podwójnej strzałki, więc `wizualnie cout << "napis1" << zmienna << "napis2" << endl;` można odczytać jako rozkaz przesłania (skierowania) do obiektu cout kolejno napisu1, zmiennej, napisu2 i znaku końca wiersza.

Na marginesie, w C++ możliwe jest też przedefiniowanie operatora + w taki sposób, że działa jak odejmowanie, oraz zdefiniowanie własnej wersji operatora – tak, aby wykonywane było dodawanie. Niewtajemniczeni czytelnicy takiego kodu źródłowego mogą długo zastanawiać się, dlaczego 2+2 = 0…

Jak widać w powyższym przykładzie, można łączyć ‘szeregowo’ kolejne operatory <<, co znacząco ułatwia korzystanie z obiektu cout.

To, co zyskujemy, to brak „procentów” znanych z printf(…) – nie potrzeba zastanawiać się, który typ zmiennej wymaga %d, %s, %x, %g itd. Niestety nic za darmo – w przypadku bardziej skomplikowanych napisów i użycia bardziej wyszukanych sposobów ich wyświetlania (np. liczba z dokładnością do 4 miejsc po przecinku, poprzedzona spacjami, ze znakiem, wyrównana do kolumy 37), printf może okazać się bardziej zwięzłym i czytelniejszym rozwiązaniem.

Typowa aplikacja studencka najczęściej będzie dobrze współpracować z cout/cin.

Podobnie jak obiektowym odpowiednikiem standardowego wyjścia jest cout, tak standardowe wejście reprezentuje obiekt o nazwie cin.

Odpowiednikiem `scanf("%d", &zmienna)` jest przykładowy kod (nie zawiera obsługi błędów):

```
int x = 0;
cin >> x;
```

Gdybyśmy jednak próbowali do skutku wczytać poprawną liczbę całkowitą i obsłużyć błędy formatu, to pojawią się pewne komplikacje w kodzie spowodowane brakiem odpowiednika funkcji fflush(stdin). Można sobie wtedy poradzić trochę inaczej:

```
int x = 0;
cout << "Podaj liczbę int: ";
while(true)
{
     cin >> x;
     bool error = cin.fail();
     cin.clear();
     cin.ignore(INT_MAX, '\n');
     if(error)
          cout << "Spróbuj ponownie: ";
     else
          break;
}
cout << x << endl;
```

Porównaj powyższy kod z przykładową obsługą błędów, jaką podawałem dla scanf(…). Powyższe zabezpieczenie wymaga większej liczby wierszy kodu, ale potencjalnie daje nam też większe możliwości.

Zadania

1. Jak można wczytywać pojedynczy znak przy użyciu cin? Może to być przydatne np. w menu wyboru (1. wczytaj dane\n2. wyświetl dane\n0. koniec);
2. Spróbuj naprzemiennie używać w programie cout oraz printf. Kiedy może wystąpić niespodziewana zmiana kolejności wyświetlania wyników w konsoli?
3. Sprawdź, w jaki sposób można formatować dane wyjściowe w cout (podobne jak to robiliśmy w przypadku printf). Wyświetl liczbę zmiennoprzecinkową z dokładnością do 3 cyfr znaczących, zmienną typach char jako liczbę dziesiętną, zmienną typu char jako znak, napis wyrównany do prawej krawędzi 30-kolumnowego pola.
4. Jak można wczytywać pełne wiersze tekstu (także te, które zawierają spacje)? Wczytaj dane do tablicy znakowej. Pamiętaj o zabezpieczeniu (nie dopuść do przepełnienia tablicy)

```
char title[50] = { 0 };
cout << "Podaj tytul ksiazki: ";
//uzupełnij ten fragment kodu
cout << "Tytul: '" << title << "'" << endl;
```