Operatory wejścia i wyjścia w języku **C** Z zastosowaniem w **C++**

Damian Kurpiewski

Obsługa wejścia/wyjścia w C++

- W języku C++ zazwyczaj używamy strumieni: `cin` i `cout`
- Są proste w obsłudze
- Nie wymagają (zazwyczaj) dodatkowej wiedzy
- Do formatowania wyjścia używamy odpowiednich poleceń
- Same „wiedzą” jak obsłużyć dany typ (podstawowy)

Obsługa wejścia/wyjścia w języku C

- Realizowana za pomocą dwóch poleceń: `printf` i `scanf`
- Wymagają podania konkretnego formatu
- Łatwiejsze (krótsze) formatowanie wyjścia i obsługa specyficznego wejścia
- Każdy typ obsługujemy ręcznie
- Trzeba znać składnię/formaty
- **Funkcje `printf` i `scanf` działają jednak znacznie szybciej od operatorów `cin` i `cout` - przy niektórych zadaniach pozwalają uniknąć TLE**

Printf - przykład

```
int a = 10;
double  b = 2.6;
char c = ‘j’;
printf("a = %d ", a); // a = 10
printf("b = %f ", b); // b = 2.600000
printf("c = %c ", c); // c = j
printf("a=%d, b=%f, c=%c", a, b, c); // a=10, b=2.600000, c=j
```

Scanf - przykład

```
int a;
double  b;
char c;
scanf("%d", &a);
scanf("%f", &b);
scanf("%c", &c);
scanf("%d %f %c", &a, &b, &c);
```

Znaki specjalne

- Zaczynają się od backslasha - \
- Pozwalają na wypisanie znaków, których nie możemy ująć w standardowy sposób
- Backslash służy także do escapowania znaków – usuwania ich specjalnego znaczenia
- Dzięki temu możemy wypisać znak ignorując jego znaczenie w kodzie programu – np. cudzysłów

Znaki specjalne

- `\n` – znak nowej linii
- `\t` – tabulacja pozioma
- `\v` – tabulacja pionowa
- `\b` – backspace
- `\a` – alarm
- `\\` – backslash
- `\?` – znak zapytania
- `\’` – apostrof
- `\"` – cudzysłów

Printf

- Funkcja `printf` przyjmuje dwa argumenty:
  1. Format wypisywanego komunikatu
  2. Lista wartości (oddzielonych przecinkami) do podstawienia
- Pierwszy argument, format, to zwyczajny tekst, który może zawierać znaki specjalne
- Każdy standardowy symbol (litera, cyfra) zostanie wypisana jak zwykły tekst

```
printf("Hello World!");
```

Printf

- Każdy symbol specjalny składający się ze znaku % i litery oznacza, że w to miejsce należy wstawić odpowiednią wartość

- Np. %d w formacie zostanie zastąpiony wartością typu `int` podaną jako kolejny argument funkcji `printf`

```
printf("%d", 68);
printf("Wynik to %d", 24);
```

Printf

- Każdy standardowy typ zmiennych z języka C ma odpowiadający mu format w funkcji `printf`
- Jeżeli chcemy wypisać kilka wartości (zmiennych) podajemy je po przecinku
- Zostaną one wstawione w kolejne miejsca oznaczone w formacie

```
printf("%d %d", 12, 67);
printf("Jest godzina %d:%d", 12, 24);
```

Formaty typów

- %d – int
- %u – unsigned int
- %lld – long long int
- %llu - unsigned long long int
- %f – double
- %c – char
- %s – char*

Printf i string

- Typ `string` pochodzi z języka C++
- W związku z tym nie ma swojego formatu w funkcji `printf`
- Jego odpowiednik w C to tablica znaków `char`
- Możemy prostym sposobem przekonwertować string to tablicy znaków
- Służy do tego funkcja `c_str()`

```
string txt = "Wypisz ten tekst";
printf("%s", txt.c_str());
```

`Printf` – formatowanie szerokości

- Możemy wypisać wartość w polu o zadanej minimalnej szerokości
- Domyślnie liczby będą wyrównane do prawej
- W tym celu po znaku % a przed literą formatu podajemy liczbę określającą szerokość pola

```
printf("%3d", 0);
printf("%3d", 123456789);
printf("%8d", -10);
```

`Printf` – formatowanie szerokości

- Aby wyrównać wartość do lewej należy podać szerokość pola jako liczbę ujemną

```
printf("%-3d", 0);
printf("%-3d", 123456789);
printf("%-8d", -10);
```

`Printf` – uzupełniane do szerokości

- Wypisywaną wartość liczbową możemy także uzupełnić zerami z lewej strony
- W tym celu piszemy `%0`, zadaną szerokość pola i literę formatu

```
printf("%03d", 0); // 000
printf("%08d", 24); // 00000024
printf("%02d:%02d", 14, 5); // 14:05
```

`Printf` – liczby rzeczywiste

- Domyślnie przy użyciu formatu `%f` dostaniemy liczbę z dokładnością do 6 miejsc po przecinku
- Aby uzyskać inną dokładność, podajemy ją po kropce przed literą formatu
- Liczba jest zaokrąglana do zadanej liczby miejsc po przecinku

```
printf("%f", 12.85); // 12.850000
printf("%.2f", 12.85); // 12.85
printf("%.1f", 12.85); // 12.9
```

`Printf` – liczby rzeczywiste

- Możemy połączyć określenie szerokości pola z dokładnością do zadanej liczby miejsc po przecinku
- Należy pamiętać, że szerokość uwzględnia całą liczbę (łącznie ze znakiem `.`)

```
printf("%6.2f", 12.85);  //  12.85
printf("%06.2f", 12.85); // 012.85
printf("%-6.2f", 12.85);
```

`Printf` – notacja naukowa

Aby wypisać liczbę rzeczywistą w formacie naukowym używamy formatu %e

```
printf("%e", 5.65); // 5.650000e+00
printf("%e", 5.653745343438343); // 5.653745e+00
printf("%e", 4342342343245.0); // 4.342342e+12
```

`Printf` – znak liczby

Aby zawsze wypisywać znak liczby (nie tylko przy wartościach ujemnych) możemy podać znak + przed literą formatu

```
printf("%+d", 5); // +5
printf("%+d", -5); // -5
```

`Printf` – inne systemy

- Za pomocą `printf` możemy w łatwy sposób wypisać liczbę całkowitą w systemie oktalnym lub heksadycemalnym
- Służą do tego dwa formaty:
  - %o – liczba oktalna (system ósemkowy)
  - %x – liczba heksadycemalna (system szesnastkowy)

```
printf("%o", 127); // 177
printf("%x", 127); // 7f
printf("%X", 127); // 7F
```

Scanf

- Funkcja `scanf` pozwala nam wczytać dane od użytkownika
- Jej użycie wygląda podobnie: podajemy format i zmienne, do których chcemy wczytać wartości
- W formacie zazwyczaj uwzględniamy tylko formaty typów
- Zmienne natomiast powinny być podane jako adres w pamięci
- Dlatego poprzedzamy je symbolem &

```
int a;
scanf("%d", &a);
```

Scanf

Używamy takich samych oznaczeń formatów jak w przypadku funkcji `printf`

```
double d;
char c;
scanf("%f %c", &d, &c);
```

Scanf

- %% - wczytuje znak % i go ignoruje

```
int a;
scanf("%d%%", &a);   // 23%
```

Scanf

- %o – wczytuje liczbę w formacie oktalnym
- %x – wczytuje liczbę w formacie heksadecymalnym

```
int a, b;
scanf("%o", &a); // 012
scanf("%x", &b); // Af
printf("a=%d, b=%d", a, b); // a=10, b=175
```

`Scanf` - tekst

- Podobnie jak w przypadku funkcji `printf`, `scanf` nie obsługuje typu string
- Możemy jedynie wczytać tekst to tablicy typu `char`
- W przypadku tablic nie podajemy znaku &, ponieważ nazwa zmiennej tablicowej zwraca adres w pamięci, gdzie znajduje się początek tablicy

```
char txt[100];
scanf("%s", txt);
```

Scanf - ignorowanie

- Aby wczytać i zignorować wejście, czyli nie przypisywać jego wartości do zmiennej, możemy po znaku % dać *

```
int a;
scanf("%*s %d", &a);    // Age: 29
printf("a = %d", a);    // a = 29
```

`Scanf` – maksymalna długość

- Możemy określić ile maksymalnie znaków ma zostać wczytanych
- Reszta danych zostanie zignorowana

```
int a;
scanf("%2d", &a);        // 1234
printf("a = %d", a);     // a = 12
```

`Scanf` – maksymalna długość

- Możemy też nie ignorować reszty wejścia i przypisać je do kolejnej zmiennej

```
int a, b;
scanf("%2d%d", &a, &b);                 // 1234
printf("a = %d, b = %d", a, b);    // a = 12, b = 34
```

`Scanf` – koniec wejścia

- W momencie napotkania końca wejścia, funkcja `scanf` zwróci stałą EOF
- Aby więc wczytywać wejście aż do jego końca, możemy np.:

```
int a;
while(scanf("%d", &a) != EOF) {
 …
}
```