

# Podstawy programowania

## Materiały dydaktyczne do laboratorium

mgr inż. Robert Ostrowski<sup>‡</sup>

8 października 2022

---

## Zadania domowe

---

### 1 Więcej o typach danych

Do dalszych ćwiczeń wykorzystamy typ `unsigned char`, który zwykle zajmuje w pamięci tylko 1 bajt (8 bitów) i pozwala na przechowanie wartości z zakresu od 0 do 255 ( $2^8 = 256$  różnych wartości).

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

int main() {
    unsigned char x;
    x = 48;
    cout << x << endl;    // 0

    x = 65;
    cout << x << endl;    // A
    return 0;
}
```

Wydruk tego programu pokazuje, że wartości zapisane w typie `unsigned char` są domyślnie interpretowane przez strumień `cout` jako znaki.

Kolejne liczby z zakresu od 0 do 127 odpowiadają znakom wyświetlanym na ekranie zgodnie z kodami **ASCII**. Pozostałe kody od 128 do 255 to tak zwane rozszerzone kody **ASCII**, które mogą być różnie interpretowane w zależności od strony kodowej. Wszystkie dane w pamięci komputera zapisane są jako liczby, a typ danej zmiennej stanowi o tym, w jaki sposób dane te są interpretowane. Możemy zażądać zmiany sposobu interpretacji danych *konwertując* je na inny typ. Konwersje typu mogą być *jawne*, lub *niejawne*. Przypatrzmy się najpierw konwersji jawnej:

---

<sup>‡</sup>Instrukcja oryginalnie została opracowana przez dr. inż. Łukasza Kusznera

**Przykład 1. Rzutowanie typów**

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

int main() {
    unsigned char x;
    x = 'A';
    cout << (int)x << endl;    // 65

    x = 65;
    cout << (int)x << endl;    // 65
    return 0;
}
```

Powyższy przykład pokazuje również w jaki sposób użyć kodu danej litery stosując apostrofy.

**Zadanie 1.** Napisz program, który wyświetla kody ASCII małych samogłosek w alfabecie łacińskim, t.j. liter: **aeiouy**.

**Zadanie 2.** Napisz program, który wyświetla znaki o kodach ASCII: 59, 93, 35.

**Zadanie 3\*.** Napisz program, który wyświetla znaki o kodach ASCII: 7, 9, 10, 13, 32. Dowiedz się, w jaki sposób zapisywany jest znak końca wiersza w zależności od systemu operacyjnego (Unix, Windows, Mac OS).

**Zadanie 4\*\*.** Spróbuj znaleźć sposób wyświetlenia reprezentacji w pamięci dowolnej zmiennej.

## 2 Operatory i niejawna konwersja typów

**Przykład 2. Przekroczenie zakresu i konwersja niejawna**

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

int main() {
    unsigned char x;
    x = 16;
    x = x * x;
    cout << (int)x << endl;    // 0

    unsigned char y;
    y = 16;
    cout << (int)(y * y) << endl;    // 256
    return 0;
}
```

**Zadanie 5.** Przeanalizuj powyższy przykład, skąd wzięło się 0 w pierwszym wypisanym wierszu? Wynik 256 w kolejnym wierszu wynika z tego, że przed wykonaniem obliczeń zmienne typu **char** konwertowane są niejawnie (bez żądania programisty) do typu **int**;

podobnie jest dla typu `unsigned char`, który również jest konwertowany do `int` (z wyjątkiem sytuacji, gdy oba typy zajmują tyle samo bajtów pamięci).

**Zadanie 6.** Porównaj dwa poniższe zapisy:

```
unsigned char x;  
x = 16 * 16;  
x = x / 4;
```

oraz

**Przykład 3.**

```
unsigned char x;  
x = 16 / 4;  
x = x * 16;
```

Przeanalizuj (najpierw bez użycia komputera) jaka będzie wartość `x` w obu przypadkach? Sprawdź swoje przypuszczenia.

**Zadanie 7\*.** Porównaj kody poniższych programów, skąd biorą się różnice w wynikach?

**Przykład 4.**

```
#include<iostream>  
using namespace std;  
  
int main() {  
    unsigned char x = 2;  
    unsigned char y = 3;  
    cout << x - y << endl;    // -1  
    return 0;  
}
```

**Przykład 5.**

```
#include<iostream>  
using namespace std;  
  
int main() {  
    unsigned int x = 2;  
    unsigned int y = 3;  
    cout << x - y << endl;    // duża liczba  
    return 0;  
}
```

**Zadanie 8.** Przeanalizuj poniższy program, zwróć uwagę na rzutowanie i typ literalów (2 jest typu `int`, 2.0 jest typu `double`).

```
#include<iostream>  
using namespace std;  
  
int main() {  
    unsigned char x = 7;
```

```
cout << x / 2 << endl;           // 3
cout << x / 2.0 << endl;         // 3.5
cout << (double)x / 2 << endl;   // 3.5
return 0;
}
```

### 3 Obliczenia zmiennoprzecinkowe

Do typów zmiennoprzecinkowych zaliczamy między innymi `float` i `double`. Charakteryzują się one możliwością przechowywania liczb niecałkowitych w formie *mantysa* \*  $2^{\text{cecha}}$ , w którym mantysa i cecha przechowywane są na osobnych bitach. Zmieniając wartość cechy, przy tej samej mantysie otrzymujemy liczby z przesuniętym przecinkiem (w sensie reprezentacji dwójkowej), stąd nazwa *typy zmiennoprzecinkowe*. Obliczenia wykonywane na typach zmiennoprzecinkowych nie mają wszystkich własności znanych z matematyki, na przykład dodawanie i mnożenie nie musi być łączne (kolejność wykonywania działań może mieć wpływ na wynik<sup>1</sup>).

**Zadanie 9.** Przeanalizuj poniższy przykład, zastanów się jaki powinien być wynik działania programu, a następnie uruchom go i zweryfikuj swoje przypuszczenia.

**Przykład 6.**

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

int main() {
    double x = 0.3;
    x = x - 0.1;
    x = x - 0.1;
    x = x - 0.1;
    if (x == 0)
        cout << "zero" << endl;
    else
        cout << "nie_zero" << endl;
    cout << x << endl;
    return 0;
}
```

Zapamiętaj, że obliczenia z wykorzystaniem typów zmiennoprzecinkowych, takich jak `float` i `double`, mogą powodować błędy zaokrągleń. W szczególności warto zapamiętać, że skończone ułamki dziesiętne nie muszą mieć skończonego rozwinięcia dwójkowego (np. 0.1 ma nieskończone, okresowe rozwinięcie dwójkowe).

**Zadanie 10.** Dane są liczby  $a$  i  $b$  - współczynniki równania liniowego  $ax + b = 0$ . Napisz program, który znajduje rozwiązanie tego równania.

**Zadanie 11.** Napisz program, który oblicza pole trójkąta o zadanych bokach.

**Zadanie 12\*.** Napisz program, który oblicza pole koła o danym promieniu.

---

<sup>1</sup>Więcej informacji na ten temat można znaleźć na przykład tutaj: <http://hal.archives-ouvertes.fr/docs/00/28/14/29/PDF/floating-point-article.pdf>.

**Zadanie 13\*\*.** Zapoznaj się dokładniej z normą IEEE 754, która opisuje standard reprezentacji binarnej i operacji na liczbach zmiennoprzecinkowych.

## 4 Typ wskaźnikowy

Zmienna może przechowywać informacje o tym, gdzie znajduje się inna zmienna, a więc jej adres w pamięci komputera.

**Zadanie 14.** Przeanalizuj poniższy przykład, zastanów się jaki powinien być wynik działania programu, a następnie uruchom go i zweryfikuj swoje przypuszczenia.

### Przykład 7.

```
#include<iostream>
using namespace std;
int main() {
    int x = 5;
    int *pi = &x;
    int y = x;
    *pi = 6;
    cout << x << " " << y << endl;
    return 0;
}
```

A jak będzie w tym przykładzie?

### Przykład 8.

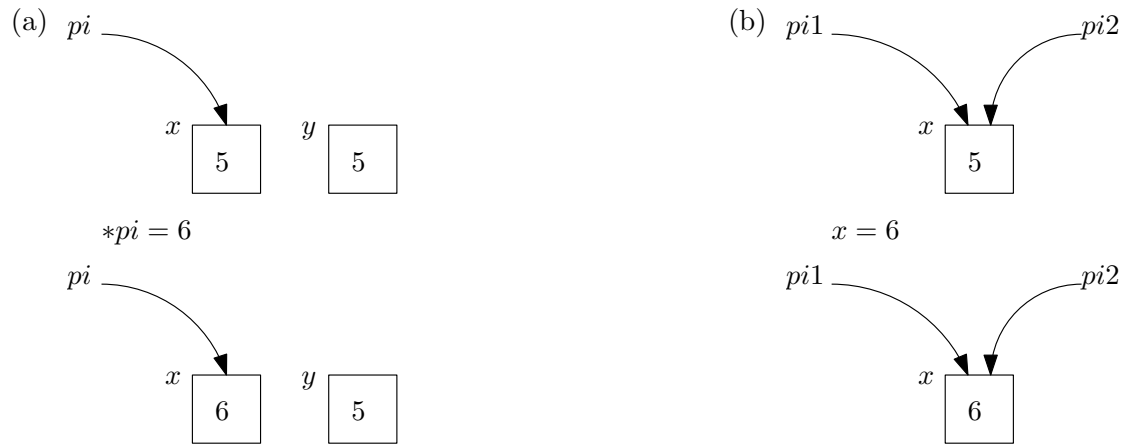
```
#include<iostream>
using namespace std;
int main() {
    int x = 5;
    int *pi1 = &x;
    int *pi2 = &x;
    cout << *pi1 << " " << *pi2 << endl;
    x = 6;
    cout << *pi1 << " " << *pi2 << endl;
    return 0;
}
```

Żeby uzmysłowić sobie jak działają operacje na wskaźnikach można reprezentować adres zmiennej poprzez strzałkę, która na niego pokazuje. Rysunek 1 obrazuje różnicę pomiędzy powyższymi przykładami.

Nic nie stoi na przeszkodzie, żeby zmienna wskaźnikowa przechowywała adres innej zmiennej wskaźnikowej. Proszę przeanalizować poniższy przykład i zrobić rysunek obrazujący tę sytuację:

### Przykład 9.

```
#include<iostream>
using namespace std;
int main() {
    int x = 5;
```



Rysunek 1: Ilustracja sytuacji z przykładów 7(a) i 8(b).

```

int *pi = &x;
int **ppi = &pi;
int y = x;
*ppi = &y;
*pi = 6;
**ppi = 7;
cout << x << " " << y << endl;
return 0;
}

```

## 5 Więcej o instrukcjach iteracyjnych

**Zadanie 15.** Proszę zapoznać się ze składnią pętli `while` oraz `do-while`.

**Zadanie 16.** Przeanalizuj poniższy program, który kolejne liczby podane przez użytkownika przepisuje na standardowe wyjście. Jeśli podane dane są liczbami, to program kończy działanie, gdy zostanie podana liczba 0 lub ujemna.

**Przykład 10.**

```

#include<iostream>
using namespace std;

int main() {
    int i;
    cin >> i;
    while (i > 0) {
        cout << i << endl;
        cin >> i;
    }
    return 0;
}

```

Zobacz co się stanie, gdy jako dane wejściowe podane zostaną liczba 12 a następnie litera **a**. Dlaczego tak się dzieje?

W kolejnych zadaniach, podobnie jak w przykładzie można założyć, że dane są liczbami.

**Zadanie 17.** Napisz program, który przepisuje na standardowe wyjście kolejne liczby podane przez użytkownika aż do momentu, gdy użytkownik poda tę samą liczbę dwa razy pod rząd.

**Zadanie 18.** Proszę porównać następujące programy (efekt działania każdego z nich jest taki sam).

Przykład 11 przedstawia najbardziej typowy zapis powodujący wykonanie grupy instrukcji określoną liczbę razy (tutaj 7 razy):

**Przykład 11.**

```
#include<iostream>
using namespace std;
int main() {
    for (int i = 0; i < 7; ++i) {
        cout << "Witaj_przyjacielu" << endl;
    }
    return 0;
}
```

Poniżej, w przykładzie 12 wykorzystano równoważny zapis z użyciem pętli **while**. Jest on bardziej rozwlekły i dlatego zwykle w tej sytuacji używamy **for**.

**Przykład 12.**

```
#include<iostream>
using namespace std;
int main() {
    int i = 0;
    while (i < 7)
    {
        cout << "Witaj_przyjacielu" << endl;
        ++i;
    }
    return 0;
}
```

Nic nie stoi na przeszkodzie, aby zastosować odliczanie do zera, jest to bardziej naturalne dla pętli **while**. Zapis taki może być wygodny, gdy liczba powtórzeń jest wartością pochodzącą z wcześniejszych obliczeń, nie musimy wtedy deklarować kolejnego licznika.

**Przykład 13.**

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

int main() {
    int i = 7;
```

```
while (i > 0)
{
    cout << "Witaj przyjacielu" << endl;
    i--;
}
return 0;
}
```

Dekrementację licznika można przenieść do warunku:

#### Przykład 14.

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

int main() {
    int i = 7;
    while (i-- > 0) {
        cout << "Witaj przyjacielu" << endl;
    }
    return 0;
}
```

Kolejne przykłady zawierają niezalecane elementy.

#### Przykład 15.

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

int main() {
    int i = 7;
    while (i--) // brzydko ,
        // zapis i>0 lepiej
        // oddaje intencję programisty
        cout << "Witaj przyjacielu" << endl;
        // brak nawiasów zmniejsza czytelność
        // i sprzyja powstawaniu błędów
        // przy późniejszych modyfikacjach
    return 0;
}
```

Liczba całkowita interpretowana jest jako wartość logiczna. Dlatego w przykładzie 15 pętla będzie się wykonywać tak długo jak wartość zmiennej *i* pozostanie niezerowa. Jednak użycie operatora porównania logicznego *>* poprawia czytelność programu.

#### Przykład 16.

```
#include<iostream>
using namespace std;
int main() {
    for (int i = 7; i > 0; i--)
    {
```



```
    cout << "Witaj_przyjacielu" << endl;
}
return 0;
}
```

W przykładzie 16 licznik zmniejsza swoją wartość. Jest to dopuszczalny zapis również w pętli `for`, jednak w tym przypadku nieco gorzej oddaje intencje programisty, gdyż wprowadzenie nietypowego odliczania nie ma to żadnego uzasadnienia.

W przykładzie 17 nietypowe jest przeniesienie instrukcji inkrementacji licznika, do warunku, co pogarsza czytelność.

#### Przykład 17.

```
#include<iostream>
using namespace std;
int main() {
    for (int i = 7; i-- > 0; ) {
        cout << "Witaj_przyjacielu" << endl;
    }
    return 0;
}
```

Ostatni przykład pokazuje bardzo zwarty zapis, który również nie jest typowy. Nie jest również czytelny, a dodatkowo prowokuje trudny do znalezienia błąd, gdy nie napiszemy średnika (kompilator tego błędu nie zauważy, po prostu wykona w pętli kolejną instrukcję).

#### Przykład 18.

```
#include<iostream>
using namespace std;
int main() {
    for (int i = 7; i-- > 0; cout << "Witaj_przyjacielu" << endl);
    return 0;
}
```

## 6 Wybór wielowariantowy

**Zadanie 19.** Przeanalizuj poniższy przykład, uzupełnij program o zliczanie znaków interpunkcyjnych: `.,,:;`

#### Przykład 19.

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

int main() {
    int samogloski = 0;
    int biale = 0;
    int inne = 0;
    int cyfry = 0;
    char z;
```

```

while(cin >> noskipws >> z) {
    switch (z) {
        case 'a': case 'e': case 'i': case 'o': case 'u': case 'y':
        case 'A': case 'E': case 'I': case 'O': case 'U': case 'Y':
            samogloski++;
            break;
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            cyfry++;
            break;
        case '_': case '\t': case '\n':
            biale++;
            break;
        default:
            inne++;
    }
}
cout << "Wczytano" << endl;
cout << "_samoglosek_" << samogloski << endl;
cout << "_cyfr_" << cyfry << endl;
cout << "_bialych_znakow_" << biale << endl;
cout << "_pozostalych_" << inne << endl;
cout << "*****" << endl;
return 0;
}

```

**Przykład 20.**

```

#include<iostream>
using namespace std;
int main() {
    int samogloski = 0;
    int biale = 0;
    int inne = 0;
    int cyfry = 0;
    char z;
    while(cin >> noskipws >> z) {
        switch (z) {
            case 'a': case 'e': case 'i': case 'o': case 'u': case 'y':
            case 'A': case 'E': case 'I': case 'O': case 'U': case 'Y':
                samogloski++;
                break;
            case '_': case '\t': case '\n':
                biale++;
                break;
            default:
                if (z >= '0' && z <= '9') cyfry++;
                else inne++;
        }
    }
}

```

```
}  
cout << "Wczytano" << endl;  
cout << " _samoglosek _:" << samogloski << endl;  
cout << " _cyfr _:" << cyfry << endl;  
cout << " _bialych _znakow _:" << biale << endl;  
cout << " _pozostalych _:" << inne << endl;  
cout << " _*****" << endl;  
return 0;  
}
```

**Zadanie 20.** W przykładzie 20 zastąpiono 10 instrukcji `case` jedną instrukcją warunkową. Było to możliwe, gdyż kody ASCII cyfr są kolejnymi liczbami. Podobnie jest dla liter. Wykorzystaj ten fakt i uzupełnij rozwiązanie zadania 19 o zliczanie spółgłosek.

**Zadanie 21\*.** Napisz program, który wczyta zapis dodawania dwóch liczb zapisanych trzynastkowo (cyfry odpowiadające liczbom 10, 11 i 12 będą zapisane jako A, B i C) wykona działanie i wypisze wynik dziesiętnie. Na przykład dla danych: 1A045 + CA76 wypisze 78739.

---

## Praca na zajęciach

---

**Zadanie 22.** Czterech graczy:  $G_1$ ,  $G_2$ ,  $G_3$  i  $G_4$  uczestniczy w pewnej grze planszowej. Gracze na przemian rzucają kostką do gry, na której wypada : 1, 2, 3, 4, 5 albo 6 oczek. Zaczyna gracz  $G_1$ . Każdy z graczy ma jeden pionek stojący początkowo na polu startowym. Wyrzucenie 1 i w następnym ruchu 6 pozwala graczowi wystartować. Liczba oczek uzyskana w kolejnych rzutach oznacza liczbę pól, o które gracz przemieszcza swój pionek. Na planszy, oprócz pola startowego jest  $X$  pól. Osiągnięcie ostatniego pola oznacza koniec gry i zwycięstwo gracza, którego pionek osiągnął ostatnie pole.

Mając dane rozmiar planszy i wyniki rzutów kostką. Przeprowadź symulację gry.

### Format danych wejściowych:

Najpierw  $X$  - rozmiar planszy a następnie w kolejnych wierszach litera M, odstęp i wartość kolejnego rzutu albo litera P - żądanie wyświetlania stanu gry.

### Format danych wyjściowych:

Dla każdej linii zawierającej P wypisz numery pól zajmowanych przez graczy  $G_1$ ,  $G_2$ ,  $G_3$  i  $G_4$  oddzielonych odstępami. Jeśli nastąpi koniec gry, przestań przetwarzać dane i wyświetl komunikat: END OF GAME.

### Przykład

#### Dla danych wejściowych:

```
5
M 1
M 4
M 4
M 5
P
M 6
M 2
M 4
M 1
P
M 3
P
M 4
```

#### Odpowiednie dane wyjściowe:

```
0 0 0 0
0 0 0 0
3 0 0 0
```

**Zadanie 23.** Dana jest gra taka jak w zadaniu 22, ale dodatkowo każdy z graczy ma przypisane dwie diody. Dioda  $S_i$  sygnalizuje, czy gracz  $i$  już wystartował, a dioda  $W_i$  sygnalizuje, że gracz  $i$  przebywa na polu o nieparzystym numerze. Zapamiętaj ich stan w  $z$  - zmiennej typu `unsigned char` w porządku  $S_1W_1S_2W_2S_3W_3S_4W_4$ . Do ustawiania odpowiednich bitów w  $z$  użyj operatorów bitowych `&` i `|`.

W danych wyjściowych, dla każdej komendy `P` dodatkowo wyświetl binarnie stan rejestru  $z$ .

### Przykład

Dla danych wejściowych:

```
5
M 1
M 4
M 4
M 5
P
M 6
M 2
M 4
M 1
P
M 3
M 4
P
```

Odpowiednie dane wyjściowe:

```
0 0 0 0 00000000
0 0 0 0 10000000
3 0 0 0 11000000
```

**Zadanie 24.** Dana jest gra taka jak w zadaniu 23, ale dodatkowo każdy ruch, który kończy się na zajętych polach powoduje, że stojący tam wcześniej pionek trafia z powrotem na pole startowe i dopiero po uzyskaniu sekwencji startowej może się znowu poruszać.

## 1 Quiz

1. Podaj co wydrukuje poniższy program wiedząc, że liczby ujemne są kodowane w kodzie U2:

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

int main() {
    char x = 7;
    char z = x - 9;
    unsigned char c = z;
    cout << (int)c << " ";
```

```
    cout << (int)(z * c) << endl;  
    return 0;  
}
```

- (a) 254 -508
- (b) -2 4
- (c) 254 64516
- (d) 254 508

2\* Przyjmijmy, że po uruchomieniu poniższego programu

```
#include<iostream>  
using namespace std;  
  
int main() {  
    int i;  
    cin >> i;  
    float x = i;  
    while (x < (float)(i + 16)) {  
        cout << x << endl;  
        x++;  
    }  
    return 0;  
}
```

użytkownik prawidłowo podał liczbę z zakresu typu `int` w reprezentacji 4 bajtowej, wtedy:

- (a) za każdym razem program wypisze 16 kolejnych liczb,
  - (b) w zależności od podanej liczby program albo wypisze 16 kolejnych liczb albo wpadnie w nieskończoną pętlę,
  - (c) w zależności od podanej liczby może się zdarzyć, że program nic nie wypisze albo wpadnie w nieskończoną pętlę,
  - (d) za każdym razem program wypisze 16 liczb, ale niekoniecznie kolejnych.
3. Podaj co wydrukuje poniższy program przy założeniu, że typ `char` jest kodowany w ASCII.

```
#include<iostream>  
using namespace std;  
  
int main() {  
    cout << 3.0 / 2 << "␣";  
    cout << 0x10 / 2 << "␣";  
    cout << (int)('b' - 'a') << endl;  
    return 0;  
}
```

- (a) 1 5 EOF
- (b) 1.5 8 1
- (c) 1.5 5 ?
- (d) 1.5 ? ?
- (e) 1 8 NAN

Symbol ? w odpowiedzi oznacza wartość nieokreśloną.

4. Podaj co wydrukuje poniższy program

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

int main() {
    int a = 2;
    int b = 6;
    int *p = &a;
    int *s = &b;
    int **q = &p;
    int **r = &s;
    *r = &a;
    cout << (*s) * a;
    a++;
    cout << " " << (*p) * a << endl;
    return 0;
}
```

- (a) ? ?
- (b) 4 9
- (c) 2 12
- (d) 2 6
- (e) 4 4

#### Odpowiedzi do quizu

1 a, 2 c, 3 b, 4 b.

---

## Zadania domowe

---

**Zadanie 25.** Dodaj pewien nowy element do gry z zadania 24. Na przykład: wyrzucenie pewnej liczby powoduje dodatkowy ruch; wybrane pole/pola mają specyficzne właściwości itp.