Laboratorium 1: „Podstawy języka C”

dr inż. Arkadiusz Chrobot
dr inż. Grzegorz Łukawski

4 września 2026

1. Wprowadzenie

Instrukcja zawiera informacje o podstawowych konstrukcjach w języku C. Część pierwsza wyjaśnia w jaki sposób mogą być umieszczane komentarze w kodzie programu. W części drugiej opisano sposób deklarowania zmiennych oraz podano informacje na temat podstawowych typów danych. Część trzecia poświęcona jest operatorom dostępnym w języku C. Część czwarta traktuje o instrukcjach warunkowych. Ostatnia część dotyczy funkcji umożliwiających komunikację z użytkownikiem.

2. Komentarze

Język C udostępnia trzy sposoby tworzenia komentarzy w kodzie źródłowym. Pierwszy z nich polega na umieszczeniu treści komentarza między znakami `/*` i `*/`. Taki komentarz może obejmować wiele wierszy i uznawany jest przez każdy standard języka C. Komentarzy tego typu nie wolno zagnieżdżać. Drugi sposób został zapożyczony z języka C++ i wprowadzony w standardzie ISO C99. Taki komentarz rozpoczyna się od znaków `//` i kończy się wraz z końcem wiersza. Trzeci rodzaj komentarza stosowany jest rzadko, zazwyczaj tam, gdzie nie możemy zastosować komentarza pierwszego typu. Ten komentarz związany jest z instrukcjami preprocesora. Listing 1 pokazuje sposób tworzenia takich komentarzy.

```c
int main(void)
{
    #if 0
            Tak wygląda trzeci rodzaj komentarzy w języku C.
            Jak widać, taki komentarz może obejmować wiele wierszy kodu.
            /* A nawet można w nim zagnieżdżać komentarze pierwszego rodzaju. */
    #endif
    return 0;
}
```

Listing 1: Komentowanie kodu z użyciem instrukcji preprocesora

3. Zmienne

W języku C zmienne są deklarowane według następującego schematu:

`typ_zmiennej nazwa_zmiennej;`

Zmienne zadeklarowane na zewnątrz funkcji `main()` są zmiennymi globalnymi. Tabela 1 zawiera informacje na temat podstawowych typów danych w języku C. Domyślnie typy `int`, `long int`, `short int`

Nazwa	Rozmiar (w bajtach)	Wartości
`int`	4	liczby całkowite
`short int`	2	liczby całkowite
`long int`	8	liczby całkowite
`char`	1	znaki lub liczby całkowite
`float`	4	liczby zmiennoprzecinkowe
`double`	8	liczby zmiennoprzecinkowe
`long double`	12	liczby zmiennoprzecinkowe

Tabela 1: Podstawowe typ danych w języku C

oraz `char` przechowują liczby ze znakiem. Możemy je przekształcić na typy przechowujące wyłącznie liczby naturalne umieszczając przed ich nazwą słowo kluczowe `unsigned`. Istnieje komplementarne do niego słowo `signed`, ale zazwyczaj się go nie stosuje. Rozmiar poszczególnych typów danych zależny jest od platformy sprzętowej dla której są pisane i kompilowane programy. Dane z tabeli dotyczą 32 i 64-bitowych komputerów klasy PC. W ogólnym przypadku rozmiar typu zmiennej możemy zbadać za pomocą operatora `sizeof`. Większość kompilatorów pozwala na samodzielne użycie słów kluczowych

`long` i `short` do określenia typu zmiennej i w takim przypadku zmienna przechowuje liczby całkowite. Oprócz wymienionych w tabeli 1 typów w języku C stosowany jest również typu `void`, ale nie służy on do deklarowania zmiennych, z wyjątkiem wskaźników, które będą opisane w innych instrukcjach. Używany jest także w funkcjach, które będą przedmiotem innych zajęć. Ciekawostką jest to, że operator `sizeof` podaje wielkość typu `void` jako jeden bajt, ale próba kompilacji programu zawierającego zmienną typu `void` kończy się komunikatem błędu, który podaje, że wielkość zmiennej jest nieokreślona.

4. Operatory

W tym punkcie opisane zostaną podstawowe operatory używane w języku C. Jeden z nich został już wymieniony w poprzednim punkcie. Jest to operator wyznaczający rozmiar zmiennej na podstawie jej nazwy lub nazwy jej typu, czyli `sizeof`. Wartość przez niego zwracana jest liczbą naturalną. Operatorem jest również, oznaczana w języku C znakiem `=`, instrukcja przypisania. Dzięki temu można użyć jej tak, jak pokazuje listing 2.

```
int a, b, c;
int main(void)
{
    a=b=c=4;    //Zmiennym a, b i c przypisywana jest wartość 4.
    return 0;
}
```

Listing 2: Przykład użycia instrukcji przypisania

Tabele 2, 3, 4 oraz 5 zawierają informacje na temat operatorów odpowiednio: arytmetycznych, bitowych, logicznych i relacyjnych.

Operatory	**Opis działania**
`++`, `--`	Jednoargumentowe operatory inkrementacji i dekrementacji. Mogą być stosowane zarówno przed, jaki i po argumencie np. `++a;` (preinkrementacja), `a++` (postinkrementacja), `--a;` (predekrementacja), `a--;` (postdekrementacja). Operatory te zwracają wartość zmiennej. Różnica między wersją „post”, a „pre” polega na sposobie jej zwracania (patrz wyjaśnienie w tekście).
`+`, `-`	Operatory te mogą być dwuargumentowe, i wtedy oznaczają dodawanie i odejmowanie, lub jednoargumentowe. W tym ostatnim przypadku operator `-` oznacza zmianę znaku wartości zmiennej, a `+` nie ma żadnego efektu.
`*`, `/`	Operatory mnożenia i dzielenia. **Uwaga**: Jeśli argumentami operatora dzielenia są liczby (wyrażenia) całkowite, to i wynik jest liczbą całkowitą, niezależnie od tego w jakiej zmiennej go zapiszemy.
`%`	Operator *modulo* - reszta z dzielenia. W języku C działa on również dla liczb ujemnych.

Tabela 2: Operatory arytmetyczne

Jeśli w wyrażeniu operator dwuargumentowy użyty jest z tą samą zmienną, do której ma być przypisany wynik, to język C pozwala na skrócony zapis takiego wyrażenia. Listing 3 zawiera kilka przykładów.

```c
int a=2, b=2;
int main(void)
{
    b+=a; //Zamiast b=b+a;
    b-=a; //Zamiast b=b-a;
    b*=a; // Zamiast b=b*a;
    b/=a; // Zamiast b=b/a;
    b%=a; // Zamiast b=b%a;
    return 0;
}
```

Listing 3: Skrócony zapis wyrażeń dla dwuargumentowych operatorów arytmetycznych

Listing 4 ilustruje różnice między działaniem operatorów inkrementacji i dekrementacji w wersjach „post” i „pre”.

```c
#include<stdio.h>
int a = 4, b;
int main(void)
{
    b=++a; //Po wykonaniu: zmienna b ma wartość 5, zmienna a ma wartość 5.
    printf("a: %d, b: %d\n",a,b);
    a=4;
    b=a++; //Po wykonaniu: zmienna b ma wartość 4, zmienna a ma wartość 5.
    printf("a: %d, b: %d\n",a,b);
    a=4;
    b=--a; //Po wykonaniu: zmienna b ma wartość 3, zmienna a ma wartość 3.
    printf("a: %d, b: %d\n",a,b);
    a=4;
    b=a--; //Po wykonaniu: zmienna b ma wartość 4, zmienna a ma wartość 3.
    printf("a: %d, b: %d\n",a,b);
    a=4;
    a++; //Po wykonaniu: zmienna a ma wartość 5.
    printf("a: %d\n",a);
    a=4;
    ++a; //Po wykonaniu: zmienna a ma wartość 5.
    printf("a: %d\n",a);
    a=4;
    a--; //Po wykonaniu: zmienna a ma wartość 3.
    printf("a: %d\n",a);
    a=4;
    --a; //Po wykonaniu: zmienna a ma wartość 3.
    printf("a: %d\n",a);
    return 0;
}
```

Listing 4: Operatory inkrementacji i dekrementacji

Dla operatorów bitowych też można stosować zapis skrócony, podobny do tego zaprezentowanego w listingu 3.

Patrząc na tabele 3 i 4 łatwo zauważyć, że operatory sum i iloczynów bitowych i logicznych są bardzo podobne w zapisie. Aby uniknąć pomyłek zapamiętać zasadę mnemotechniczną jaką zaproponował Bruce Eckel: *Bity są małe, więc operatory bitowe potrzebują tylko jednego znaku do zapisu.* W przypadku opera-

Operatory	Opis działania
`|`, `&`,`^`	Operatory sumy bitowej (or), iloczynu bitowego (and) oraz bitowej różnicy symetrycznej (xor).
`~`	Jednoargumentowy operator negacji bitowej.
`>>`, `<<`	Operatory przesunięcia bitowego w prawo i w lewo. **Uwaga**: W języku C operatory te działają także dla liczb ujemnych, w szczególności przesunięcie w prawo liczby ujemnej daje w wyniku liczbę ujemną - najstarszy bit uzupełniany jest jedynką, zgodnie z kodem U2.

Tabela 3: Operatory bitowe

Operatory	Opis działania
`||`, `&&`	Operatory sumy logicznej (or) i iloczynu logicznego (and).
`!`	Jednoargumentowy operator negacji logicznej.

Tabela 4: Operatory logiczne

torów logicznych można również stosować skrócony zapis wyrażeń, podobny do tego, jaki zaprezentowano w listingu 3.

W standardach języka C poprzedzających ISO C99 nie wprowadzono osobnego typu logicznego, zatem wszystkie wartości różne od zera są traktowane jako prawda, a równe zero jako fałsz. Od wprowadzenia wyżej wymienionego standardu dostępny jest typ `bool`, razem z wartościami `true` i `false`, ale dopiero po włączeniu pliku nagłówkowego `stdbool.h`. Sposób kodowania prawdy i fałszu nie uległ zmianie.

W złożonych wyrażeniach z użyciem operatorów logicznych stosowane jest tzw. skracanie obliczeń. Ilustruje je listing 5. Uzasadnienie wyników pokazanych wyrażeń jest następujące: operator `&&` daje w wyniku prawdę, wtedy i tylko wtedy, gdy oba jego argumenty są prawdziwe. Jeśli więc pierwszy argument jest fałszywy, to całość wyrażenia musi być fałszywa, a więc wartość drugiego argumentu nie jest już wyznaczana. W przypadku operatora `||` uzasadnienie jest podobne: daje on fałsz wtedy i tylko wtedy, gdy oba jego argumenty są fałszywe, więc jeśli pierwszy jest prawdziwy, to drugi już nie jest sprawdzany.

```c
int a=0,b=0;
int main(void)
{
    (a=0)&&(b=4); //Obie zmienne mają wartość 0 po wykonaniu wyrażenia.
    (a=4)&&(b=3); //Po wykonaniu wyrażenia zmienna a ma wartość 4,
                  // a zmienna b wartość 3.
    (a=0)||(b=0); //Obie zmienne mają wartość 0 po wykonaniu wyrażenia.
    (a=3)||(b=4); //Po wykonaniu wyrażenia zmienna a ma wartość 3,
                  //a zmienna b wartość 0.
    return 0;
}
```

Listing 5: Skrócone obliczenia

Operatory	Opis działania
`==`, `!=`	Operator porównania i operator „różne”. Pierwszy daje w wyniku 1 (prawdę), jeśli oba argumenty są takie same, drugi jeśli są różne.
`<`, `>`, `<=`, `>=`	operatory „mniejsze”, „większe”, „mniejsze lub równe”, „większe lub równe”

Tabela 5: Operatory relacyjne

Niniejsza instrukcja nie zawiera informacji na temat priorytetów operatorów. Można je znaleźć w innych źródłach. Warto jednak nadmienić, że podobnie jak w wyrażeniach matematycznych, w języku

C możemy zmienić kolejność działań za pomocą nawiasów okrągłych.

Niektóre operacje matematyczne są przeprowadzane za pomocą funkcji dostępnych po włączeniu pliku nagłówkowego `math.h`. Tabela 6 zawiera informacje o niektórych elementach dostępnych za jego pośrednictwem.

Operatory	Opis działania
`sin(x)`, `cos(x)`, `tan(x)`	Sinus, kosinus i tangens. **Uwaga**: Kąt (zmienna `x`) jest podawany w radianach.
`pow(x,y)`	Liczy potęgę x^y.
`sqrt(x)`	Liczy pierwiastek kwadratowy z `x`.
`M_PI`	Stała π. Nie jest ona dostępna w standardzie ISO C99, ale w rozszerzeniu GNU C.

Tabela 6: Funkcje i stałe z biblioteki math.h

5. Operator trójargumentowy

Ogólna postać operatora trójargumentowego jest następująca:

```
warunek?instrukcja_1:instrukcja_2;
```

Jeśli warunek stojący przed znakiem zapytania jest prawdziwy, to operator zwraca wartość wyrażenia stojącego bezpośrednio za pytajnikiem, w przeciwny przypadku, wartość wyrażenia stojącego za dwukropkiem. Warunek przed pytajnikiem może składać się z mniejszych warunków połączonych dowolnymi operatorami, choć najczęściej używa się relacyjnych i logicznych. W takim przypadku nazywamy go warunkiem złożonym. **Uwaga**: język C dopuszcza użycie w warunku instrukcji przypisania `=`, która podobna jest w zapisie do operatora porównania `==`. Te operatory działają w odmienny sposób i dosyć często użycie przypisania zamiast porównania jest pomyłką programisty. Czasem jednak jest to efekt zamierzony, bo taki zapis może okazać z pewnych względów wygodny. Listing 6 zawiera instrukcję operator `?` użyty do wybrania największej z dwóch wartości zmiennych. W tym zapisie warunek ma postać `a>b` instrukcja przypisania jest użyta poza warunkiem i służy do nadania zmiennej `max` wartości zwróconej przez operator trójargumentowy.

```c
int a=5, b=4, max;
int main(void)
{
    max=(a>b)?a:b;
    return 0;
}
```

Listing 6: Operator trójargumentowy

6. Komunikacja z użytkownikiem

Oprócz poznanej w poprzedniej instrukcji funkcji `puts()` do komunikacji z użytkownikiem mogą służyć funkcje `scanf` i `printf`, dostępne także po włączeniu pliku nagłówkowego `stdio.h`.

Funkcja `scanf()` pozwala użytkownikowi wprowadzić wartość z klawiatury, którą zapamiętuje we wskazanej zmiennej. Listing 7 pokazuje przykład użycia takiej funkcji.

Funkcja ta przyjmuje (co najmniej) dwa argumenty. Pierwszym jest tzw. ciąg formatujący, który określa jak ma być traktowana wartość wprowadzona przez użytkownika za pośrednictwem klawiatury. Drugi argument jest adresem zmiennej, do której wartość ma być wprowadzona. Ares ten uzyskiwany jest za pomocą operatora wyłuskania (`&`), który zapisywany jest tym samym symbolem, co operator iloczynu bitowego. Kompilator rozpoznaje taki zapis po tym, że operator wyłuskania jest jednoargumentowy, a operator iloczynu logicznego dwuargumentowy. Tabela 7 zawiera opis kilku przykładowych ciągów formatujących.

```c
int a;
int main(void)
{
    scanf("%d",&a);
    return 0;
}
```

Listing 7: Funkcja `scanf()`

Ciąg formatujący	Opis
%d	Liczba całkowita typu `int`.
%ld	Liczba całkowita typu `long int`.
%u	Liczba naturalna typu `unsigned int`.
%c	Znak.
%f	Liczba zmiennoprzecinkowa typu `float`.
%lf	Liczba zmiennoprzecinkowa typu `double`.

Tabela 7: Ciągi formatujące dla funkcji `scanf()`

Funkcja `printf()` pozwala wypisać na ekranie wartości zmiennych. Podobnie jak `scanf()` wymaga użycia ciągów formatujących (tabela 8), które określają jak wartość ma być wypisana na ekranie. Dodatkowo pozwala na użycie znaków sterujących (tabela 9). Listing 8 zawiera kilka przykładów użycia funkcji `printf()`.

```c
int a=4, b=5;

int main(void)
{

    printf("%d",a); //Wypisanie wartości zmiennej a.
    printf("%d\n",a); //Wypisanie wartości zmiennej a i przejście do następnego
                      //wiersza.

    /*
     * Funkcja printf() potrafi wypisywać bardziej złożone komunikaty.
     * W tych komunikatach ciągi formatujące są zastępowane wartościami zmiennych.
     */

    printf("Zmienna \"a\" ma wartość: %d\n",a);
    printf("Zmienna \"a\" ma wartość: %d, a zmienna \"b\" ma wartość: %d\n",a,b);
    return 0;
}
```

Listing 8: Przykłady użycia funkcji `printf()`

Ciąg formatujący	Opis
`%d`	Liczba całkowita typu `int`.
`%ld`	Liczba całkowita typu `long int`.
`%u`	Liczba naturalna typu `unsigned int`.
`%c`	Znak.
`%f`	Liczba zmiennoprzecinkowa typu `double` lub `float`. Może zawierać dodatkowe informacje o formatowaniu np. `%.3f` - wartość będzie wypisana z dokładnością do trzech miejsc po przecinku.
`%lf`	Liczba zmiennoprzecinkowa typu `double`. Podobnie jak wyżej, może zawierać dodatkowe informacje o formatowaniu.
`%Lf`	Liczba zmiennoprzecinkowa typu `long double`. Podobnie jak wyżej, może zawierać dodatkowe informacje o formatowaniu.
`%e`, `%E`	Liczba zmiennoprzecinkowa typu `double` lub `float` w notacji wykładniczej, np. `3e-9`, jeśli użyte jest pierwsze formatowanie, lub `3E-9` , jeśli drugie.
`%le`, `%lE`	Liczba zmiennoprzecinkowa typu `double` w notacji wykładniczej, np. `3e-9`, jeśli użyte jest pierwsze formatowanie, lub `3E-9`, jeśli drugie.
`%Le`, `%LE`	Liczba zmiennoprzecinkowa typu `long double` w notacji wykładniczej, np. `3e-9`, jeśli użyte jest pierwsze formatowanie, lub `3E-9` , jeśli drugie.
`%x`, `%X`	Liczba naturalna zapisana w kodzie szesnastkowym, np. `a5`, jeśli użyte jest pierwsze formatowanie, lub `A6`, jeśli użyte jest drugie formatowanie.
`%o`	Liczba naturalna w kodzie ósemkowym.

Tabela 8: Ciągi formatujące dla funkcji `printf()`

Znak sterujący	Opis
`\n`	Znak nowego wiersza.
`\r`	Znak początku wiersza.
`\\`	Znak \ (wypisanie na ekranie).
`\"`	Znak ".
`\t`	Tabulator

Tabela 9: Znaki sterujące dla funkcji `printf()`

7. Zadania

1. Sprawdź przy pomocy debuggera jak zmieniają się wartości zmiennych `a` i `b` podczas krokowego wykonania programu z Listingu 5.

2. Sprawdź przy pomocy debuggera jaka będzie końcowa wartość zmiennej `max` w programie z Listingu 6.

3. Napisz program, który pobierze od użytkownika trzy liczby całkowite i wypisze na ekranie tę, która jest największa spośród nich.

4. Powtórz zadanie z punktu pierwszego, ale tym razem zamiast liczb niech użytkownik podaje pojedyncze znaki. Program powinien wybrać ten, który najpóźniej występuje w alfabecie. **Uwaga!** Jeśli w programie umieszczamy kilka następujących po sobie wywołań funkcji `scanf()` z ciągiem formatującym "%c", to zaobserwujemy, że nie wszystkie znaki z klawiatury są odczytywane. Dzieje się tak dlatego, że `scanf("%c");` odczytuje nie tylko „zwykłe" znaki, ale również znaki nowego wiersza pozostawione przez klawisz `Enter`. Aby poradzić sobie z tym problemem wystarczy, począwszy od drugiego wywołania funkcji `scanf()`, zamiast ciągu "%c" zastosować ciąg " %c" (ze spacją przed znakami % i c).

5. Napisz program, który pobierze do użytkownika dwie liczby, zapisze je w zmiennych typu `unsigned char`, wypisze je na ekranie, a następnie poda wyniki działań *and*, *or* i *xor* wykonanych na tych

liczbach. Wszystkie wartości na ekranie powinny być wypisane w kodzie dziesiętnym i szesnastkowym.

6. Zadeklaruj w programie trzy zmienne typu `int`, `short int` i `double` i wypisz na ekranie ich wartości, nie przypisując im nic wcześniej.

7. Napisz program, który pobierze od użytkownika dwie liczby typu `int`, następnie je podzieli i wypisze wynik na ekranie. Czy wyniki są zawsze takie, jakich się spodziewałaś/spodziewałeś?

8. Napisz program, który pobierze od użytkownika dwie liczby całkowite, a następnie wypisze na ekranie pierwszą z nich jeśli jest ona podzielna przez drugą lub drugą, jeśli pierwsza nie jest podzielna przez drugą.

9. Jak sprawdzić, czy liczba jest parzysta bez używania operatora modulo? Napisz program, który to pokaże.

10. Zadeklaruj zmienną typu `unsigned char`. Odejmij od jej początkowej wartości jeden. Wypisz na ekranie wartość tej zmiennej. Jak wytłumaczyć ten wynik?

11. (trudniejsze) Poszukaj informacji o prawach de Morgana. Napisz program, który oblicza iloczyn bitowy dwóch zmiennych, ale bez używania operatora `&`. Powtórz to zadanie dla sumy bitowej.