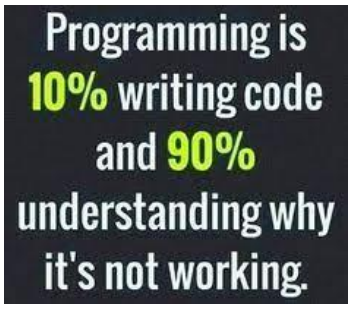
Podstawy programowania w języku C

```c
#include <math.h>
#include <stdio.h>
#include <conio.h>


double X(double t) {
    return (1.0 + sin(2.0 * M_PI * 10000.0 * t) + cos(2.0 * M_PI * 10000.0 * t));

    struct complexx {
        double Re;
        double Im;
    };

    struct complexx cmplx(double A, souble B) {
        struct complexx result;
        result.Re = A;
        result.Im = B;
        return (result);
    }


    struct complex cMult(struct complexx wA, struct complexx B) {

        struct complexx result;

        result.Re = A.Re * B.Im + A.Im * B.Re;
```

Wprowadzenie do języka C.

Lab1

Język C został opracowany jako strukturalny język programowania do celów ogólnych. Przez całą swą historię służył do tworzenia przeróżnych programów – od systemów operacyjnych po programy nadzorujące pracę urządzeń przemysłowych. C, jako język dużo szybszy od języków interpretowanych (Perl, Python) oraz uruchamianych w maszynach wirtualnych (np. C#, Java) może bez problemu wykonywać złożone operacje nawet wtedy, gdy nałożone są dość duże limity czasu wykonywania pewnych operacji. Jest on przy tym bardzo przenośny – może działać praktycznie na każdej architekturze sprzętowej pod warunkiem opracowania odpowiedniego kompilatora.



Wykonał: Dawid Kosior
kosiordawid@gmail.com

Pierwszy program

Rozpocznijmy od stworzenia krótkiego programu który wyświetla „Hello world!”. W C, użyjemy polecenia ***printf*** do wyświetlenia tekstu. Poniżej przedstawiono gotowy program.

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Wynikiem wykonania oczywiście jest wyświetlenie na konsoli napisu:
Hello world!

Zastanówmy się co oznacza poniższa linia kodu:

```c
#include <stdio.h>
```

nagłówek <stdio.h> jest niezbędne aby móc używać funkcji printf().

Ostatnią instrukcją w programie jest instrukcja ***return***. Linia ***return 0*** kończy główną funkcję jaką jest funkcja ***main()*** i powoduje, że zwraca wartość 0 do procesu wywołującego. Wartość niezerowa (zwykle 1) sygnalizuje nieprawidłowe zakończenie działania programu.

Struktura programu w C:

```c
#include "nazwa_pliku"
//wstawianie plików - w/w wiersz jest zastępowany plikiem o nazwie nazwa_pliku
#include <nazwa_pliku>
//efekt zastosowania tej instrukcji jw. z tym, że dodatkowo zleca się kompilatorowi
//poszukiwanie pliku w pewnym wyróżnionym katalogu (w Unix jest to skorowidz
/usr/include)
int main ()
{
.....
}
```

Komentarze

Komentarze można wykorzystać aby opisać kod: co robi i do czego służy. Komentarze jednolinijkowe poprzedzane są dwoma slashami **//** informują one kompilator aby ignorował on wszystko od tych dwóch slashów aż do końca bieżącej linii.

```c
#include <stdio.h>

int main() {
    // wypisywanie  Hello World!
    printf("Hello, World!\n");
    return 0;
}
```

Gdy zachodzi potrzeba za komentowania więcej niż jednej linii kodu należy na początku komentarza wstawić **/*** i odpowiednio zamknąć komentarz znakami ***/**

Zmienne

Utworzenie zmiennej rezerwuje miejsce w pamięci lub przestrzeń w pamięci do przechowywania wartości. Kompilator wymaga podania typu danych dla każdej zadeklarowanej zmiennej. C oferuje bogaty asortyment wbudowanych oraz zdefiniowanych przez użytkownika typów danych.

Liczba całkowita, typ wbudowany, reprezentuje liczbę całkowitą. Zdefiniuj liczbę całkowitą, używając słowa kluczowego **int**. C wymaga określenia typu i identyfikatora dla każdej zdefiniowanej zmiennej. Identyfikator to nazwa zmiennej, funkcji lub dowolnego innego elementu zdefiniowanego przez użytkownika. Identyfikator rozpoczyna się od litery (A-Z lub a-z) lub znaku podkreślenia (_), po którym występują dodatkowe litery, podkreślenia i cyfry (od 0 do 9). Na przykład zdefiniuj zmienną o nazwie ***myVariable***, która może zawierać wartości całkowite i spróbuj wypisać jej wartość w następujący sposób:

```c
#include <stdio.h>

int main() {
    int myVariable = 10;
     printf("%d\n", myVariable);
    return 0;
}
```

W języku programowania C rozróżniana jest wielkość liter, więc myVariable i myvariable to dwie różne zmienne

Deklaracje podstawowych typów zmiennych

Zmienna/słowo kluczowe	Typ
char	znak
int	liczba całkowita
float	liczba zmiennoprzecinkowa
double	liczba zmiennoprzecinkowa podwójnej precyzji
void	brak wartości

Przykłady deklaracji w programie

```c
char c = 'c';
int liczba = 20;
float l1 = 30.5;
double l2 = 20.5;
```

Wypisywanie zmiennych

```c
#include <stdio.h>

int main() {
        int age = 18;
        printf("Wiek przed: %d\n", age);

        age = 22;
        printf("Wiek po: %d\n", age);
        return 0;
}
```

Po deklaracji zmiennej (typ nazwa zmiennej) i przypisaniu jej wartości podczas jej ponownego użycia odwołujemy się do niej tylko przez nadaną jej wcześniej nazwę

W powyższym programie została utworzona zmienna ***age*** wraz z przypisaniem jej wartości. Pierwsza instrukcja ***printf()*** wypisuje komunikat wzbogacony o wartość przechowywaną w zmiennej ***age*** w tym celu w komunikacie wyświetlanym znajduje się znak przekształcenia ***%d*** wyznaczający miejsce wstawienia wartości odczytanej ze zmiennej której nazwa znajduje się po przecinku(***age***).

Lista znaków przekształceń

d	argument będzie przekształcony do postaci dziesiętnej
o	argument będzie przekształcony do postaci ósemkowej
x	argument będzie przekształcony do postaci szesnastkowej
c	argument będzie traktowany jako jeden znak
s	argument jest tekstem znakowym
e	argument będzie traktowany jako liczba typu float ([-]m.nnnnnE[+-]xx)
f	argument będzie traktowany jako liczba typu float ([-]mmm.nnn)
lf	argument będzie traktowany jako liczba typu double ([-]mmm.nnn)

Wypisywanie większej ilości zmiennych w pojedynczym komunikacie odbywa się w następujący sposób:

```c
printf("zmienna age= %d, zmienna x= %d\n", age, x );
```

Operatory

Operator numeryczne	Opis	Operator porównania	Opis	Operatory przypisania	Zapis równoważny
x + y	suma x oraz y	x != x	różne	x+=2	x=x+2
x - y	różnica x oraz y	x == x	równe	x-=2	x=x-2
x * y	iloczyn x oraz y	x > x	większe	x*=2	x=x*2
x / y	iloraz x oraz y	x < x	mniejsze	x/=2	x=x/2
x % y	reszta z dzielenia x / y	x <= x	mniejsze lub równe	x%=2	x=x%2

Język C posiada też specjalne operatory, takie jak "**++**" będący inkrementacją tzn. że wartość zmiennej zostaje zwiększona o 1 oraz "**--**" nazywany dekrementacją czyli zmniejszenie wartości zmiennej o 1.

Istnieje dwie możliwości zapisu operatorów inkrementacji i dekrementacji:

- Prefiks zwiększa wartość, a następnie kontynuuje wyrażenie.
- Postfiks ocenia wyrażenie, a następnie wykonuje inkrementację.

```c
++x;    //prefix
x++;    //postfix.
```

Przykład:

```c
int x = 5;
y = ++x;
// x=6, y=6
```

```c
int x = 5;
y = x++;
// x=6, y=5
```

Priorytet operatora określa grupowanie termów w wyrażeniu, które wpływa na sposób oceny wyrażenia. Niektóre operatory mają pierwszeństwo przed innymi; na przykład operator mnożenia ma wyższy priorytet niż operator dodawania:

```c
        int x = 5 + 2 * 2;
        printf("%d\n", x );
```

Jeśli żadne z wyrażeń nie znajduje się w nawiasach, to operatory multiplikatywne (mnożenie, dzielenie, dzielenie modulo) będą oceniane przed operacjami dodawania (odejmowania).

Wprowadzanie danych

Do pobierania danych od użytkownika wykorzystuje się funkcję ***scanf()***.Poniższy przykład pokazuje, jak zaakceptować dane wprowadzane przez użytkownika i zapisać je w zmiennej ***num***:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
        int num;
        printf("Podaj liczbe: ");

        scanf("%d",&num);
        printf("\nWprowadziles liczbe %d\n", num);
        return 0;
}
```

*Przez **&num** odwołujemy się do adresu w pamięci jaki został zaalokowany do przechowywania wartości zmiennej o nazwie **num**.*

Instrukcja warunkowa IF

Możesz użyć instrukcji if, aby wykonać fragment kodu, jeśli spełniony jest określony warunek. Jeśli wyrażenie ma wartość True, wykonywane są pewne instrukcje. W przeciwnym razie są one pomijane. Instrukcja if wygląda następująco:

```
if (wyrażenie warunkowe) {
   instrukcje
}
```

W C wykorzystywane są nawiasy klamrowe, aby zgrupować blok instrukcji należących do instrukcji warunkowej.

Przykład:

```
if (10 > 5) {
    printf("warunek jest prawdziwy\n");
}
```

Aby wykonać bardziej złożone warunki, instrukcje mogą być zagnieżdżone, jedna w drugiej. Oznacza to, że wewnętrzne instrukcje if są częścią instrukcji zewnętrznej. Jest to jeden ze sposobów sprawdzenia, czy spełnione są warunki wielokrotne.

Przykład:

```
if (num > 5) {
    printf("więcej niż 5\n");
    if (num <= 45) {
        printf("wartość z przedziału (5;45]\n");
    }
}
```

Instrukcja else wykonuje instrukcję if i zawiera kod, który jest wywoływany, gdy instrukcja if zwraca wartość False.

Przykład:

```
int x = 4;
if (x == 5){
    printf("Tak");
}else {
    printf("Nie");
}
```

Instrukcje if i else można łączyć łańcuchowo, aby określić, która opcja w serii możliwości jest prawdziwa.

Przykład:

```
int num = 12;
if (num == 5)
    printf("Numerem jest 5");
else
    if (num == 10)
        printf("Numerem jest 10");
    else
        printf("Numerem nie jest 5 ani 10");
```

Zadania do wykonania

1. Przetestuj zamieszczone wyżej fragmenty kodu i sprawdź ich działanie.
2. Stwórz po jednej zmiennej typów int, char, float, double i sprawdź w jaki sposób zostaną wyświetlone ich wartości przy użyciu każdego z znaków przekształceń.
3. Napisz program sumujący dwie liczby podane z klawiatury.
4. Napisz program rozpoznający, czy podana liczba jest pierwsza.
5. Napisz program rozpoznający, czy podana liczba jest pierwsza.
6. Napisz prosty program który wypisze wartość bezwzględną z podanej liczby (nie używając funkcji abs).
7. Przetestuj kilka podstawowych obliczeń (+, -, *, /). Spróbuj wykonać kilka różnych obliczeń z różnymi operatorami.
8. Korzystając ze zdobytych wiadomości napisz program do wyznaczania pierwiastków równania kwadratowego.