

**WYDZIAŁ INŻYNIERII MECHANICZNEJ I OKRĘTOWNICTWA**  
**INSTYTUT MECHANIKI I KONSTRUKCJI MASZYN**



**Mechatronika (sem. 2)**

**Programowanie Systemów Komputerowych**  
- część I

- programowanie strukturalne w języku C

**PROJEKT (pracownia komputerowa)**

dr hab. inż. Marek Galewski  
mgr inż. Piotr Duba

2023

Uwaga: Niniejsze materiały przeznaczone są dla studentów wyłącznie jako pomoc do zajęć dydaktycznych prowadzonych przez pracowników i doktorantów Instytutu Mechaniki i Konstrukcji Maszyn Wydziału Inżynierii Mechanicznej i Okrętownictwa Politechniki Gdańskiej. Jakiegokolwiek ich wykorzystywanie w innych celach i / lub przez inne osoby bez zgody autorów jest zabronione.

## 1 Cel przedmiotu

Celem przedmiotu jest praktyczna nauka podstaw programowania strukturalnego w języku C oraz podstaw programowania zorientowanego obiektowo w języku Java.

## 2 Wstęp

Podczas zajęć projektowych wykorzystywane jest środowisko programowania Apache NetBeans. Instrukcja instalacji narzędzi programistycznych jest dostępna na stronie kursu w portalu eNauczanie.



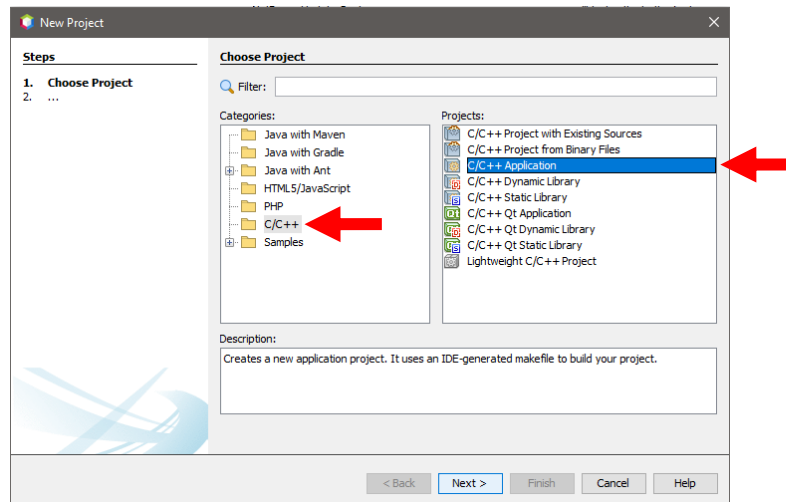
Prace należy zapisywać na dysku sieciowy X: lub w folderze lokalnym C:\Prace\XYZ\ gdzie za XYZ wybrać dowolną nazwę własnego programu. Programy umieszczone w innych folderach będą usuwane! Po zakończeniu poszczególnych zajęć zalecane jest kopiowanie wykonanych programów – nie ma gwarancji, że ktoś inny ich nie skasuje lub zmodyfikuje. Wykonane ćwiczenia będą przydatne przy pisaniu programów zaliczających zajęcia projektowe. Do wykonania zadań przydatne będą materiały z wykładów.

Instrukcja do zajęć projektowych jest podzielona na dwie części:

- część I  
Programowanie strukturalne pokazującą klasyczne podejście do zagadnienia programowania. Wykorzystano w niej język C w standardzie C99. Wszystkie zadania będą uruchamiane w konsoli tekstowej.
- część II  
Programowanie obiektowe obejmuje zagadnienia najpopularniejszej obecnie metody programowania. Została opracowana z wykorzystaniem języka Java. W ramach tej części przedstawiono również zagadnienia tworzenia programów z interfejsem graficznym GUI.

## 3 Tworzenie nowego projektu

W celu stworzenia nowego projektu otwórz menu *File > New Project...*. Następnie, w oknie *New Project* wybierz kategorię projektu *C/C++* i rodzaj *C/C++ Application* (Rys. 1) po czym kliknij *Next >*.

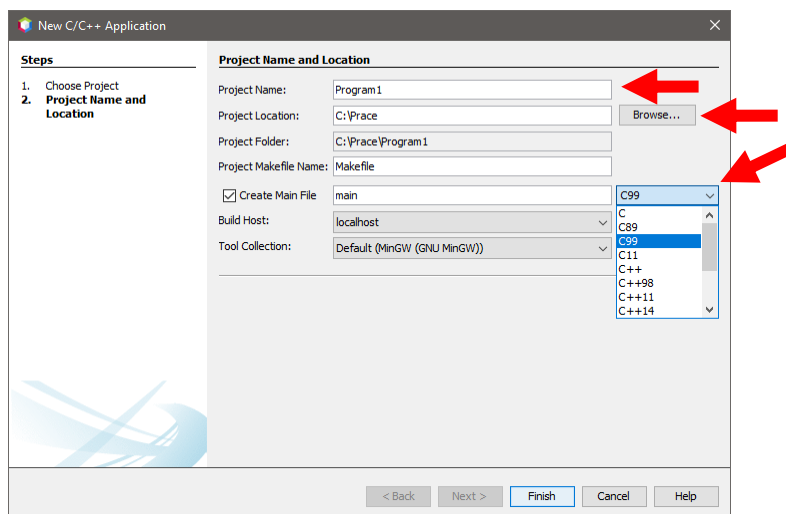


Rys. 1

W kolejnym oknie (Rys. 2) ustaw odpowiednie parametry tworzonego projektu:

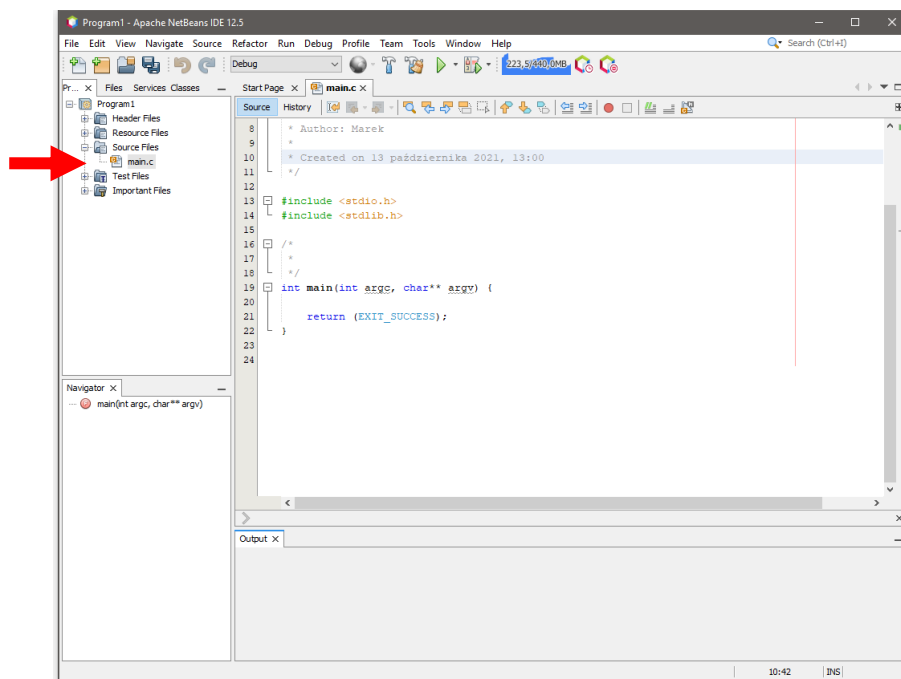
- *Project Name*: nazwa projektu,
- *Project Location*: kliknij *Browse...* i wybierz lokalizację, w której zostanie utworzony projekt. Pamiętaj, żeby wszystkie pliki zapisywać tylko na dysku X: lub w *C:\Prace\*.
- wersja języka: C99 lub C11.

Zakończ tworzenie projektu klikając *Finish*.



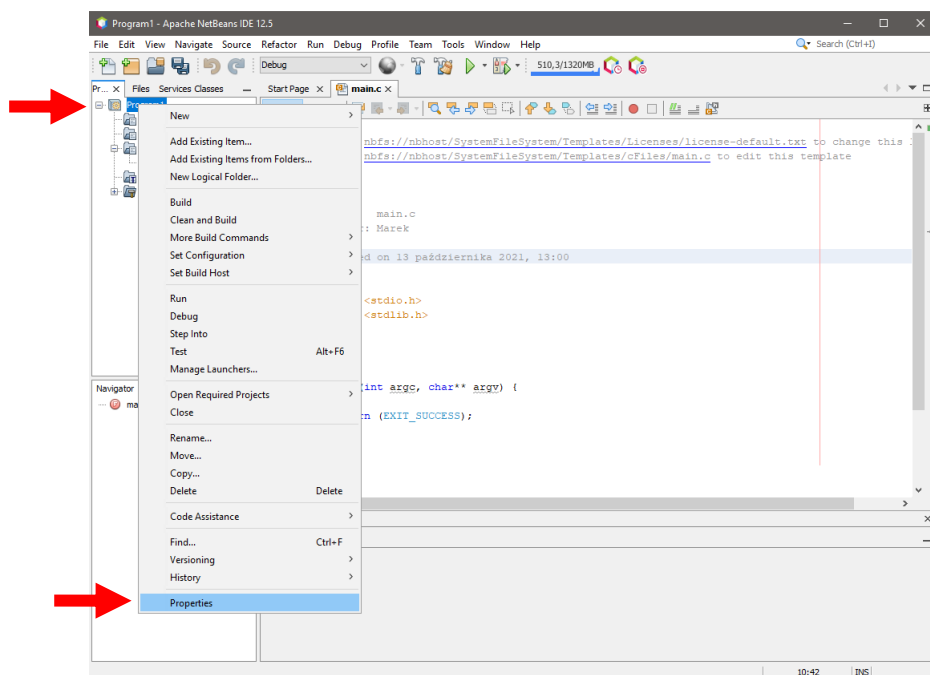
Rys. 2

W efekcie zostanie utworzony nowy projekt z jednym plikiem źródłowym *main.c*. Aby wyświetlić jego zawartość należy w zakładce *Projects* rozwinąć grupę *Source Files* i dwukrotnie kliknąć LPM na pliku *main.c*.



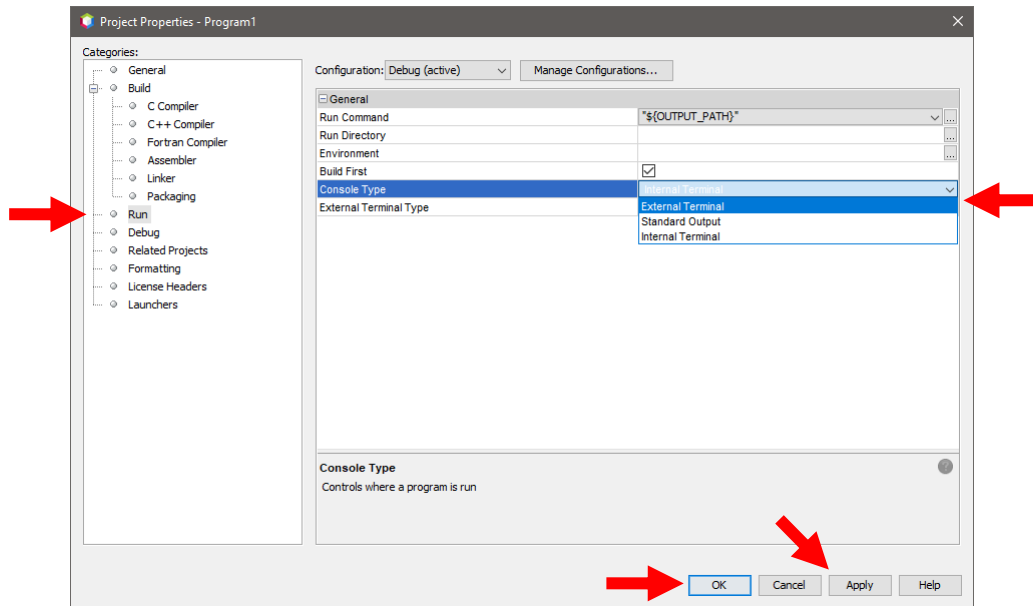
Rys. 3

Ostatnim krokiem jest zmiana domyślnej konsoli, w której program będzie uruchamiany. W tym celu kliknij PPM na nazwie projektu, a następnie wybierz *Properties* (Rys. 4).



Rys. 4

W nowym oknie wybierz *Run > Console Type > External Terminal* (Rys. 5). Zatwierdź zmiany przyciskiem *Apply*, a następnie *OK*.



Rys. 5



Zmiana konsoli na zewnętrzną konieczna będzie w przypadku każdego nowo utworzonego projektu. Zmianę należy wprowadzić dla każdego typu kompilacji (*Debug*, *Release*) osobno.



Po uruchomieniu pierwszego programu (patrz Zadanie 1) będzie można zauważyć, że czcionka w oknie konsoli jest bardzo mała (w Windows 10 standardowa czcionka konsoli ma rozmiar 7) przez co komunikaty są trudne do odczytania. Żeby zmienić rozmiar czcionki należy kliknąć prawym klawiszem myszy ikonkę w lewym górnym rogu okna konsoli i z menu wybrać *Domyślne > Czcionka > Rozmiar > np. 16* lub więcej.

## 4 Zadania projektowe do wykonania

### 4.1 Ćwiczenie 1 – Pierwszy program

#### 4.1.1 Przykład

Utwórz nowy projekt (patrz rozdział 3) i napisz program, który wyświetli komunikat „Program działa” i będzie oczekiwał na wciśnięcie klawisza. Oczekiwanie można zrealizować z użyciem funkcji `getchar()` lub `_getch()` znajdującej się w bibliotece `conio.h`. Z kolei w bibliotece `stdio.h` znajduje się funkcja `printf()` odpowiedzialna za wyświetlanie tekstu w konsoli systemowej. Odpowiedni kod źródłowy znajduje się poniżej:

```
/*
 * Pierwszy program
 * i przykład komentarza
 * w wielu liniach
 */

/* komentarz jednoliniowy */


// od wersji C99 można komentarz umieszczać także tak
```

```
// - styl komentarza zaczerpnięty z C++

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    /* wyświetlenie ciągu znaków na domyślnym strumieniu (konsola) */
    printf("Program działa!");

    getchar();
    return (EXIT_SUCCESS);
}
```

Skompiluj (polecenie *Run > Build Project* lub klawisz *F11*) program. Kompilacja powinna zakończyć się komunikatem „BUILD SUCCESSFUL” w oknie *Output*. Następnie uruchom program (polecenie *Run > Run Project*, lub klawisz *F6*, lub ikona ). Efektem działania programu powinno być pojawienie się w oknie konsoli komunikatu:

Program działa !



Funkcja *main()* jest główną funkcją programu. To od niej rozpoczyna się wykonywanie programu po jego uruchomieniu. Zmienne *int argc* i *char\*\* argv* odpowiadają za przyjęcie opcjonalnych parametrów z konsoli podanych przy wywołaniu programu. Możemy również zapisać tylko *int main(void)*. Wówczas program nie będzie obsługiwał parametrów pochodzących z konsoli.

#### 4.1.2 Zadanie

Zmodyfikuj program tak, by wyświetlił dzisiejszą datę wg poniższego formatu. Przykładowy efekt działania programu:

Dzisiaj jest 30 marca 2022 roku.

### 4.2 Ćwiczenie 2 – Komunikacja z użytkownikiem

#### 4.2.1 Przykład

Utwórz nowy projekt i napisz program obliczający sinus kąta podanego przez użytkownika w stopniach. Do pobrania danych używamy funkcji *scanf()*. Parametr *%lf* oznacza, że podany przez użytkownika ciąg znaków (cyfr) ma być zinterpretowany jako liczba zmiennopozycyjna i przypisana na zmienną typu *double*.



Funkcja *sin()* przyjmuje jako parametr wejściowy kąt podany w radianach. Dlatego konieczna jest wcześniejsza konwersja wartości kąta ze stopni na radiany.

Funkcja *scanf()* przyjmuje jako parametr adres zmiennej, a nie samą zmienną. Dlatego konieczne jest zastosowanie operatora wyłuskania *&*, który zwróci adres w pamięci, pod którym znajduje się zmienna. Jeżeli korzystamy ze zmiennej wskaźnikowej, wówczas nie używamy operatora *&*. Więcej o wskaźnikach dowiesz się na wykładzie.

Odpowiedni kod źródłowy programu znajduje się poniżej:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const double pi = 3.14159265; //definicja stalej

int main(int argc, char** argv) {
    double wynik, katStopnie; //definicje zmiennych

    printf("Wpisz wartosc kata w stopniach [deg]: ");
    scanf("%lf", &katStopnie);

    wynik = sin(katStopnie * pi / 180); //konwersja na [rad] i wyznaczenie wyniku

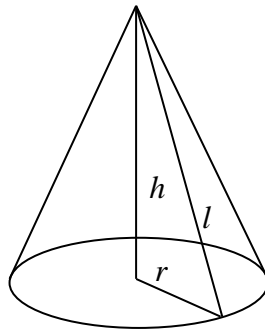
    printf("sin (%5.2lf [deg]) = %5.3lf\n", katStopnie, wynik);

    getchar();
    return (EXIT_SUCCESS);
}
```

Uruchom program i sprawdź efekty jego działania.

#### 4.2.2 Zadanie

Utwórz nowy projekt i napisz program obliczający na podstawie długości tworzącej  $l$  i wysokości  $h$  pole powierzchni całkowitej  $P$  i objętość stożka  $V$ .



Rys. 6

Przydatne wzory:

Pole podstawy:  $P_p = \pi r^2$ ,

Pole boku:  $P_b = \pi r l$ ,

Objętość:  $V = \frac{P_p h}{3}$ .

Zwróć uwagę, że do obliczeń potrzebne jest wprowadzenie tylko dwóch (a nie trzech) danych wejściowych. Dane powinny być wprowadzane przez użytkownika podczas działania programu.

Przykładowy efekt działania programu:

```
Podaj tworzaca l: 5
Podaj wysokosc h: 3
Pole = 113.097
Objetosc = 50.265
```

## 4.3 Ćwiczenie 3 – Funkcje i prototypy funkcji

### 4.3.1 Przykład

Utwórz nowy projekt i napisz, z wykorzystaniem funkcji, program obliczający pole prostokąta. W pierwszej kolejności należy utworzyć prototyp funkcji wskazujący kompilatorowi, że w tym pliku znajduje się dana funkcja (fragment A w kodzie źródłowym). Prototypy funkcji umieszcza się na początku pliku `.c` lub w osobnym pliku nagłówkowym `.h`, który trzeba później dołączyć do programu poleceniem `#include`.

We fragmencie B następuje wywołanie funkcji z podaniem parametrów (nazwy zmiennych mogą być inne niż w prototypie, ale ich typy muszą być ze sobą zgodne). Właściwy kod źródłowy funkcji znajduje się we fragmencie C.

Odpowiedni kod źródłowy programu znajduje się poniżej:

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

/* A
- prototyp funkcji poleProstokata */
float poleProstokata(float a, float b);

int main(void) {
    /* definicja zmiennych */
    float pole, dlA, dlB;

    /* wczytanie danych */
    printf("Podaj dlugosc boku a: ");
    scanf("%f", &dlA);
    printf("Podaj dlugosc boku b: ");
    scanf("%f", &dlB);

    /* B
- wywolanie funkcji pole_prostokata i przypisanie wyniku do zm. pole */
    pole = poleProstokata(dlA, dlB);

    /* wyswietlenie wyniku */
    printf("Pole prostokata = %9.3f\n", pole);

    getchar();
    return (EXIT_SUCCESS);
}

/* C
- wlasciwy kod zrodlowy funkcji obliczajacej pole prostokata */
float poleProstokata(float a, float b) {
    float wynik;
    wynik = a * b;

    /* zwrocenie wartosci wynikowej
- musi byc ona zgodna z typem zmiennej podanym przed nazwa funkcji */
    return wynik;
}
```

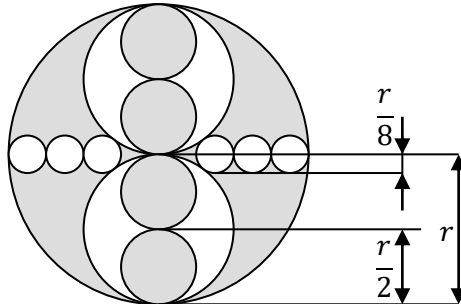
Przykładowy efekt działania programu:

```
Podaj dlugosc boku a: 2
Podaj dlugosc boku b: 5.5
Pole prostokata =    11.000
```



### 4.3.2 Zadanie dodatkowe \*

Utwórz nowy projekt i napisz program obliczający pole powierzchni szarej części figury pokazanej na Rys. 7. Koniecznie stwórz i wykorzystaj funkcję *pKola()* liczącą pole pojedynczego elementu (koła). Wykorzystaj ją do obliczania pól elementów składowych wywołując ją kilkakrotnie z odpowiednimi parametrami.



Rys. 7

Przykładowe wyniki:

dla  $r = 10$  – pole = 206.167

dla  $r = 20$  – pole = 824.668

## 4.4 Ćwiczenie 4 – Instrukcja warunkowa *if...else...*

### 4.4.1 Przykład

Utwórz nowy projekt i napisz program, który sprawdzi, czy wczytana liczba mieści się w określonym zakresie.

Odpowiedni kod źródłowy programu znajduje się poniżej:

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

int main(int argc, char** argv) {
    int liczba;

    printf("Podaj liczbę całkowitą z zakresu 0-99: ");
    scanf("%d", &liczba);

    if (liczba < 0) { // jeśli liczba jest mniejsza od zera...
        printf("Poza zakresem. Liczba jest mniejsza od 0.\n");
    } else {        // ... w przeciwnym wypadku...
        if (liczba > 99) { // ...sprawdź, czy liczba jest większa od 99
            printf("Poza zakresem. Liczba jest większa od 99.\n");
        } else {        // jeśli wszystkie poprzednie warunki nie zostały spełnione
            //to znaczy, że liczba jest w zakresie
            printf("OK. Liczba mieści się w zakresie.\n");
        }
    }
    getchar();
    return (EXIT_SUCCESS);
}
```

Przykładowy efekt działania programu:

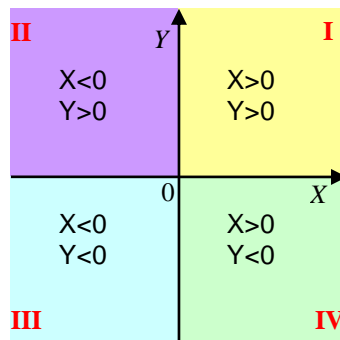
```
Podaj liczbę całkowitą z zakresu 0-99: 501
Poza zakresem. Liczba jest większa od 99.
```

#### 4.4.2 Zadanie

Utwórz nowy projekt i napisz program, który na podstawie współrzędnych X i Y punktu określi, w której ćwiartce układu współrzędnych jest on położony (zgodnie z Rys. 8). Jeśli punkt leży na osi lub w środku układu współrzędnych program powinien o tym poinformować, podając na której osi się znajduje.

Użyj konstrukcji *if...else....*. Konieczne będzie jej kilkukrotne zagnieżdżenie.

Aby połączyć dwa warunki, które mają być spełnione jednocześnie użyj operatora iloczynu logicznego `&&`, np.: *if ((x>0)&&(y>0))*. W przypadku, gdy wystarczy spełnienie jednego z kilku warunków, użyj operatora sumy logicznej `//` np.: *if ((x==0)//(y==0))*.



Rys. 8

Przykładowe efekty działania programu:

```
Podaj x : -10
Podaj y : 13
Punkt znajduje sie w II ćwiartce

Podaj x : 0
Podaj y : 13
Punkt lezy na osi OY
```

#### 4.4.3 Zadanie dodatkowe \*

Utwórz nowy projekt i napisz program, który na podstawie długości trzech odcinków (boków) sprawdzi:

- czy da się z nich zbudować trójkąt,
- czy jest to trójkąt równoboczny, równoramienny czy zwykły,
- czy trójkąt jest ostrokątny, prostokątny czy rozwartokątny.

Uwagi:

- kolejność wpisywania boków przez użytkownika może być dowolna!
- pamiętaj, że trójkąt może mieć kilka cech jednocześnie, np. równoramienny i prostokątny.

Przykładowe efekty działania programu:

```
Podaj dlugosc boku a= 1
Podaj dlugosc boku b= 1
Podaj dlugosc boku c= 2
Z podanych bokow nie da sie zbudowac trojkata.
```

```

Podaj dlugosc boku a= -1
Podaj dlugosc boku b= 2
Podaj dlugosc boku c= 2
Z podanych bokow nie da sie zbudowac trojkata.

Podaj dlugosc boku a= 3
Podaj dlugosc boku b= 4
Podaj dlugosc boku c= 5
Z podanych bokow da sie zbudowac trojkat prostokatny.

Podaj dlugosc boku a= 51
Podaj dlugosc boku b= 100
Podaj dlugosc boku c= 51
Z podanych bokow da sie zbudowac trojkat rownoramienny, rozwartokatny.

Podaj dlugosc boku a= 23
Podaj dlugosc boku b= 32
Podaj dlugosc boku c= 11
Z podanych bokow da sie zbudowac trojkat rozwartokatny.

Podaj dlugosc boku a= 9
Podaj dlugosc boku b= 9
Podaj dlugosc boku c= 9
Z podanych bokow da sie zbudowac trojkat rownoboczny, ostrokatny.

```

## 4.5 Ćwiczenie 5 – Instrukcja warunkowa *switch*

### 4.5.1 Przykład

Utwórz nowy projekt i napisz program, który na podstawie numeru dnia tygodnia wyświetli jego nazwę. W przypadku liczb 6 i 7 zamiast nazwy dnia napisze "Weekend".



Zwykle na końcu każdego wyrażenia *case* należy wstawić instrukcję *break* co zapobiega wykonaniu kolejnych przypadków z *case*. Czasem jednak, jest to pożądané, np. celowo zostało to wykorzystane do połączenia dwóch przypadków dla liczb 6 i 7 w jeden.

Odpowiedni kod źródłowy programu znajduje się poniżej:

```

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    unsigned short int nrDnia;

    printf("Podaj numer dnia tygodnia: ");
    scanf("%hu", &nrDnia); // %hu - liczba unsigned short
    switch (nrDnia) {
        case 1:
            printf("Poniedzialek\n");
            break;
        case 2:
            printf("Wtorek\n");
            break;
        case 3:
            printf("Sroda\n");

```

```
        break;
    case 4:
        printf("Czwartek\n");
        break;
    case 5:
        printf("Piątek\n");
        break;
    case 6:
    case 7:
        printf("Weekend\n");
        break;
    default:
        printf("Tydzien ma 7 dni, poprawna liczba 1-7\n");
        break;
}
getchar();
return (EXIT_SUCCESS);
}
```

Przykładowe efekty działania programu:

```
Podaj dlugosc boku a= 1
Podaj numer dnia tygodnia : 2
Wtorek
Podaj numer dnia tygodnia : 6
Weekend
```

#### 4.5.2 Zadanie zbiorcze – Użycie instrukcji warunkowych

Utwórz nowy projekt i napisz program, który wczyta liczby całkowite  $B$ ,  $W$ ,  $Z$ , a następnie poprawnie napisze tekst: „Na łące [pasła / pasły / pasło] się  $B$  [owca / owce / owiec]. Wieczorem [przyszł / przyszły / przyszło]  $W$  [wilk / wilki / wilków] i [zjadł / zjadły]  $Z$  [owcę / owce / owiec]. Rano na łące [nie było / była / były / było] już tylko [jedna /  $B - Z$ ] [owca / owce / owiec].”

Rozwiązanie powinno być tak skonstruowane, że wpisać można dowolne liczby naturalne – należy znaleźć ogólne zasady dopasowania formy rzeczowników i czasowników do podanych liczb. Program powinien także zawierać kontrolę danych:  $B \geq 1$ ,  $W \geq 1$ ,  $Z \geq 0$ ,  $B \geq Z$ . Zadanie można wykonać zarówno korzystając z instrukcji *if* jak i *switch*. Przydatny będzie także operator modulo  $rd = x \% y$  wyznaczający resztę  $rd$  z dzielenia liczby  $x$  przez  $y$ .

Przed oddaniem zadania dokładnie sprawdź różne kombinacje liczb, zwłaszcza z zakresów 1-5, 10-15, 21-25, 100-120, 1000-1020, 1100-1120.

Przykładowe efekty działania programu:

```
Podaj liczbe owiec           : 102
Podaj liczbe wilkow          : 10
Podaj liczbe zjedzonych owiec: 2
Na lace pasly sie 102 owce. Wieczorem przyszlo 10 wilkow i zjadly 2 owce.
Rano na lace bylo juz tylko 100 owiec.
```

## 4.6 Ćwiczenie 6 – Pętla *for*

### 4.6.1 Przykład

Utwórz nowy projekt i napisz program odliczający czas do „autodestrukcji”.



W zadaniu wykorzystana zostanie pętla *for*, która pozwala wykonać blok instrukcji zadaną liczbę razy. W momencie rozpoczęcia pętli liczba powtórzeń musi być znana. W naszym przypadku warunek ten jest spełniony, gdyż wcześniej użytkownik poda na ile sekund „ustawić zapalnik”.

Uwagi:

- funkcja *Sleep(czas)* z biblioteki *windows.h* służy do odczekania określonego w milisekundach czasu. Odliczanie czasu nie jest dokładne, tzn. w rzeczywistości może upłynąć go nieco więcej niż zadamy,
- funkcja *Beep(częstotliwość, czas)* generuje dźwięk o określonej częstotliwości i czasie trwania (uwaga: nie na wszystkich komputerach działa / generuje dźwięk – zależy to od konkretnej konfiguracji sprzętu i oprogramowania).

Odpowiedni kod źródłowy programu znajduje się poniżej:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char** argv) {
    int i, zakres;

    printf("Ustaw zapalnik - czas w sekundach: ");
    scanf("%d", &zakres);

    for (i = zakres; i > 0; i--) {
        printf("%d, ", i);
        Beep(1000, 100);
        Sleep(1000);
    }

    printf("Autodestrukcja!\n");
    Beep(300, 500);

    getchar();
    return (EXIT_SUCCESS);
}
```

Przykładowy efekt działania programu:

```
Ustaw zapalnik - czas w sekundach: 5
5, 4, 3, 2, 1, Autodestrukcja!
```

### 4.6.2 Zadanie

Utwórz nowy projekt i napisz program, który wyświetli tabliczkę mnożenia (konieczne będzie użycie dwóch pętli *for*, zagnieżdżonych jedna w drugiej). Pamiętaj o wyrównaniu kolumn z liczbami. Użyj odpowiedniego formatowania w funkcji *printf( )*.

Efekt działania programu:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

#### 4.6.3 Zadanie dodatkowe \*

Utwórz nowy projekt i napisz program wypisujący wszystkie dzielniki zadanej liczby naturalnej, liczbę tych dzielników oraz ich sumę. Program powinien podawać komunikat, jeżeli podana liczba jest liczbą pierwszą.

Przykładowe efekty działania programu:

```
Podaj liczbe           : 123
Dzielniki naturalne tej liczby : 1, 3, 41, 123,
Znaleziono dzielnikow      : 4
Suma dzielnikow           : 168

Podaj liczbe           : 13
Dzielniki naturalne tej liczby : 1 , 13,
Znaleziono dzielnikow      : 2
Suma dzielnikow           : 14
Podana liczba jest liczba pierwsza

Podaj liczbe           : 52
Dzielniki naturalne tej liczby : 1, 2, 4, 13, 26, 52,
Znaleziono dzielnikow      : 6
Suma dzielnikow           : 98
```

Podpowiedzi:

Algorytm: w pętli należy sprawdzać czy reszta z dzielenia zadanej liczby  $a$  przez kolejne liczby z zakresu od 1 do  $a/2$  są równe 0. Jeśli jest, wówczas należy wyświetlić znaleziony dzielnik, zwiększyć wartość zmiennej, na której zliczana jest liczba dzielników, oraz dodać znaleziony dzielnik do sumy poprzednich dzielników. Po zakończeniu pętli należy jeszcze uwzględnić dodatkowo, że liczba  $a$  jest zawsze także swoim własnym dzielnikiem. Na koniec należy wyświetlić odpowiednie komunikaty zgodnie z poleceniem oraz, na podstawie liczby znalezionych dzielników, sprawdzić czy liczba  $a$  jest liczbą pierwszą.

W programie przydatne mogą być następujące funkcje i operacje:

- operator  $a \% b$  lub funkcja  $fmod()$  z biblioteki *math.h* – wyznaczanie reszty z dzielenia  $a/b$ ,
- $x = \text{ceil}(y)$  – zaokrąglanie w górę (funkcja z biblioteki *math.h*),
- rzutowanie typów zmiennych.

## 4.7 Ćwiczenie 7 – Tablice, pętla *for*

### 4.7.1 Przykład

Utwórz nowy projekt i napisz program, który ponumeruje pola w tablicy o rozmiarze 4x4 wg wzoru:

A			
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

W programie użyj pętli *for*.

Najważniejszą częścią programu jest para zagnieżdżonych pętli *for* wpisujących wartości do tablicy. Pętla zewnętrzna (z licznikiem *w*) odpowiedzialna jest za zmianę numeru wiersza. Pętla wewnętrzna (z licznikiem *k*) odpowiedzialna jest za zmianę numeru kolumny. Wpisywanie wartości odbywa się w ten sposób, że w kolejnych iteracjach (wykonaniach) pętli wewnętrznej wpisywane są kolejne wartości w danym wierszu. Po wypełnieniu wiersza program przechodzi do następnej iteracji pętli zewnętrznej, co powoduje „przejsie” do kolejnego wiersza i ponowne rozpoczęcie pętli wewnętrznej. Wyświetlenie wyników zawarte w funkcji `wyswietlTab()` oparte jest na tej samej zasadzie. Jedynie, zamiast wpisywać wartości do tablicy, są one z niej pobierane, zaś po wypisaniu wszystkich elementów z jednego wiersza wstawiany jest znak końca linii by przejść do następnej linii na ekranie.

Odpowiedni kod źródłowy programu znajduje się poniżej:

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

#define LW (4) // deklarowane stale rozmiary tablicy z uzyciem #define
#define LK (4)

void wyswietlTab(int tablica[LW][LK]);

int main(int argc, char** argv) {
    //deklaracje zmiennych lokalnych
    int tablica[LW][LK]; //tablica do wypelnienia
    int w, k;             //numer wiersza i kolumny
    int nr;               //numer do wpisania

    nr = 0;
    //wstawianie wartosci do tablicy
    for (w = 0; w < LW; w++) { //petla "po wierszach"
        for (k = 0; k < LK; k++) { //petla "po kolumnie w danym wierszu"
            nr++;
            tablica[w][k] = nr;
        }
    }

    wyswietlTab(tablica); //wyswietlanie gotowej tablicy
    getchar();
    return (EXIT_SUCCESS);
}

//wyswietlanie tablicy
void wyswietlTab(int tablica[LW][LK]) {
    int w, k; //numer kolumny i wiersza, zmienne lokalne
```

```
//wyswietlanie gotowej tablicy
for (w = 0; w < LW; w++) {
    for (k = 0; k < LK; k++) {
        printf("%3i", tablica[w][k]);
    }
    printf("\n");
}
}
```

Efekt działania programu:

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

#### 4.7.2 Zadanie

Zmodyfikuj program, tak by ponumerował pola tablic wg wzoru zadanego poniżej. Program powinien być napisany w taki sposób, by działał dla tablicy o dowolnych rozmiarach (a nie tylko 4x4). Zmiana rozmiaru będzie następowała poprzez zmianę wartości stałych *LW* i *LK* z użyciem *#define*. Nowe tablice powinny być wyświetlane razem z poprzednią.

B			
1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

C			
1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

Zwróć uwagę, że wstawiana wartość może być zależna od numeru wiersza i kolumny. Pamiętaj również, że licznik pętli może być zmniejszany. Może być także konieczne użycie instrukcji warunkowej i więcej niż jednej pętli zagnieżdżonej.

Efekt działania programu:

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

```
1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
```

```
1 2 3 4
8 7 6 5
9 10 11 12
16 15 14 13
```



### 4.7.3 Zadanie dodatkowe \*

Zmodyfikuj program, tak by ponumerował pola kolejnych tablic wg wzorów:

D			
1	3	6	10
2	5	9	13
4	8	12	15
7	11	14	16

E			
1	2	5	10
4	3	6	11
9	8	7	12
16	15	14	13

F			
1	2	3	46
12	13	14	5
11	16	15	6
10	9	8	7

Uwaga: kolejność wpisywania liczb do tablicy jest dowolna – musisz sam, znaleźć najlepszą metodę.

## 4.8 Ćwiczenie 8 – Pętla *while*

### 4.8.1 Przykład

Utwórz nowy projekt i napisz program zbierający od użytkownika datki tak długo, aż uzbiera określoną kwotę.



W zadaniu wykorzystana zostanie pętla *while*, która pozwala wykonywać blok instrukcji tak długo, jak długo spełniony jest zadany warunek (warunek trwania pętli, a nie jej zakończenia). Pętli *while* używamy, gdy w momencie rozpoczęcia pętli liczba powtórzeń nie jest znana (np. zależy od danych lub decyzji podawanych / podejmowanych przez użytkownika w trakcie trwania pętli). W naszym przypadku nie wiemy, jakie datki będzie przekazywać użytkownik i dlatego nie można użyć pętli *for*.

Odpowiedni kod źródłowy programu znajduje się poniżej:

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

int main(int argc, char** argv) {
    int suma = 0, cel, datek;

    printf("Podaj kwote do zebrania: ");
    scanf("%d", &cel);

    while (suma <= cel) {
        printf("Prosze o datek: ");
        scanf("%d", &datek);
        printf("Dziekuje\n");
        suma = suma + datek;
    }

    printf("Zebralem = %d\n", suma);
    getchar();
}
```

Przykładowy efekt działania programu:

```
Podaj kwote do zebrania: 50
Prosze o datek: 20
Dziekuje
Prosze o datek: 20
```

```
Dziekuje  
Prosze o datek: 15  
Dziekuje  
Zebralem = 55
```

#### 4.8.2 Zadanie

Utwórz nowy projekt i napisz program „Zgadywanka”, który wylosuje pewną liczbę całkowitą z zakresu  $\langle 0; 100 \rangle$ , a następnie będzie pytał użytkownika jaka liczba została wylosowana. Po podaniu przez użytkownika liczby, program powinien komunikować, czy liczba ta jest większa czy mniejsza od liczby wylosowanej. Dodatkowo program powinien po zakończeniu zgadywania podać w ilu próbach użytkownikowi udało się zgadnąć wylosowaną liczbę.

Do wylosowania liczby możesz użyć poniższego fragmentu kodu źródłowego:

```
int lZagadka, n = 101;  
srand(time(NULL)); // inicjalizacja funkcji losujacej  
lZagadka = rand() % n; // pseudolosowa liczba z zakresu 0 do n-1
```

Funkcje wykorzystywane powyżej znajdują się w bibliotece *time.h*. Pamiętaj o jej dołączeniu poleceniem *#include*.

Przykładowy efekt działania programu:

```
Podaj liczbe z zakresu 0-100: 50  
Za malo!  
Podaj nowa liczbe: 75  
Za duzo!  
Podaj nowa liczbe: 60  
Za malo!  
Podaj nowa liczbe: 62  
Brawo - trafiles w = 4 probach!
```

### 4.9 Ćwiczenie 9 – Instrukcje warunkowe, pętla *do..while*, przechwytywanie kodów wciśniętych klawiszy z klawiatury

#### 4.9.1 Przykład

Utwórz nowy projekt i napisz program, który będzie wyświetlał kody wciśniętych klawiszy. Program ma działać tak długo, aż użytkownik wciśnie klawisz *Esc*.

Najważniejsze uwagi dotyczące obsługi klawiatury:

- Każdy klawisz / znak na klawiaturze ma przypisany kod liczbowy.
- Kod należy odczytywać funkcją *\_getch()*.
- Kody dla małej i wielkiej litery są różne (np. K – 75, k – 107).
- Część klawiszy (np. klawisze kursora, funkcyjne, *PgUp*, *PgDown* itp.) ma podwójne kody, tzn. pierwszy kod jest == 0 lub 224, a po nim następuje drugi kod. Dopiero na podstawie pary kodów można określić, który klawisz został wciśnięty. Oznacza to, że jeżeli pierwszy odczytany kod == 0 lub 224 należy dodatkowo odczytać kod jeszcze raz i na tej podstawie określić wciśnięty klawisz.
- Przy naciskaniu „zwykłych” klawiszy, sterownik klawiatury umieszcza w buforze klawiatury ich kody. W przypadku klawiszy „specjalnych” umieszczane są 2 kody. Oba muszą być odczytane. Jeśli w programie nie znajdzie się fragment kodu odpowiedzialny za odczytywanie kodów takich klawiszy (nawet gdy nie są

potrzebne do obsługi danego programu) wówczas po odczytaniu pierwszego kodu drugi kod nadal pozostanie w buforze klawiatury i w kolejnej pętli programu zostanie zinterpretowany jako zupełnie inny, „zwykły” klawisz.

- W niektórych klawiaturach (zwłaszcza w laptopach) może wystąpić „zwykły” klawisz o kodzie 224, dlatego, dodatkowo, po odczycie takiego kodu należy sprawdzić funkcją *kbhit()* czy w buforze klawiatury rzeczywiście znajduje się kolejny znak, co oznacza, że wciśnięto znak specjalny.
- Do przechowywania kodów w pamięci należy użyć zmiennej typu *wchar\_t*. Jest to odpowiednik typu *char* dla znaków kodowanych w kodzie Unicode.

Uwagi dotyczące działania programu:

- Program ma działać tak długo, aż użytkownik wciśnie klawisz *Esc*. Oznacza to, że czytanie klawiszy i wyświetlanie ich kodu musi odbywać się w pętli. W tym przypadku najlepiej użyć pętli *do..while*, która działa tak samo jak pętla *while*, z tą różnicą, że warunek trwania pętli sprawdzany jest na końcu każdej iteracji, a nie na początku. Powoduje to, że pętla wykonywana jest zawsze co najmniej jeden raz.

Gotowy kod źródłowy programu znajduje się poniżej. Odczytuje on i wyświetla kody klawiszy, dodatkowo w przypadku wciśnięcia *a*, *A*, *b*, *B*, *F1* lub *F2* wyświetla odpowiedni napis.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main(int argc, char** argv) {

    wchar_t klawisz1, klawisz2;

    printf("Wcisnij klawisze; koniec = ESC\n");
    do {
        printf("\n");
        klawisz1 = _getch(); //pobranie klawisza
        printf("kod klawisza = %d", klawisz1);
        if ((klawisz1 == 0) || (klawisz1 == 224)) {
            if (kbhit()){
                klawisz2 = _getch(); //pobranie klawisza
                printf(", %d", klawisz2);
            }
        }
        printf("\n");

        switch (klawisz1) { //kod wcisnietego klawisza
            case 65: //a lub A
            case 97:
                printf("A\n");
                break;
            case 66: //b lub B
            case 98:
                printf("B\n");
                break;
            case 27: //ESC
                break;
            case 0: //klawisze specjalne
                switch (klawisz2) {
                    case 59: //F1
                        printf("F1\n");
                        break;
                    case 60: //F2
                        printf("F2\n");
                        break;
                }
        }
    }
```

```

        break;
    case 224: //klawisze specjalne
        break;
    default:
        break;
}

} while (klawisz1 != 27);
printf("\n");

return (EXIT_SUCCESS);
}

```

Przykładowy efekt działania programu dla sekwencji klawiszy *a*, *A*, *F1*,  $\uparrow$ , *x*, *Esc*:

Wciskaj klawisze; koniec = ESC

kod klawisza = 97

A

kod klawisza = 65

A

kod klawisza = 0, 59

F1

kod klawisza = 224, 72

kod klawisza = 120

kod klawisza = 27

## 4.9.2 Zadanie

Zmodyfikuj program tak, by wyświetlał wprowadzane samogłoski jako wielkie litery (a nie ich kody), a w miejsce spółgłosek i innych znaków wstawiał znak „\_”. Klawisze specjalne (*F1...F12*, *Delete* itd.) również powinny wyświetlać pojedynczy znak „\_”.

Uwagi:

- Przy modyfikacji kodu źródłowego będzie trzeba usunąć część programu (ale uważaj, by nie usunąć zbyt dużo, np. obsługa kodów 0 i 224 musi pozostać).
- Kody samogłosek:

	Wielka litera	Mała litera
A, a	65	97
E, e	69	101
I, i	73	105
O, o	79	111
U, u	85	117
Y, y	89	121

- Do zamiany kodu małej litery na wielką służy funkcja *toupper()*, zamiana powinna być wykonana przed strukturą *switch*. Będzie można wówczas pozbyć się połowy przypadków *case* (dla małych liter).

Efekt działania programu przy wprowadzeniu tekstu „Alicja ma białego kota”:

Napisz tekst; koniec = ESC  
A\_I\_A\_A\_IA\_E\_O\_O\_A

## 4.10 Ćwiczenie 10 – Tablice i zmienne wskaźnikowe

### 4.10.1 Przykład

Utwórz nowy projekt i napisz program, który, podobnie jak w ćwiczeniu 7, ponumeruje pola w tablicy o dowolnym zadany rozmiarze wg wzoru:

A			
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



Ponieważ tablica ma mieć dowolne rozmiary, konieczne będzie użycie tablicy dynamicznej, do której dostęp możliwy jest tylko za pośrednictwem zmiennych wskaźnikowych.

W przypadku zmiennych dynamicznych nie można zadeklarować tablicy dwuwymiarowej. Należy zarezerwować obszar pamięci o odpowiednim rozmiarze będącym iloczynem obu wymiarów tzn:  $rozmiar = rozmiarX \cdot rozmiarY$ . Następnie, przy odwoływaniu się do poszczególnych komórek zarezerwowanego obszaru należy pamiętać o odpowiednim korygowaniu adresów względem początku obszaru pamięci wskazywanego przez wskaźnik. Np. by w macierzy o 3 kolumnach odwołać się do elementów z wiersza 0 korekta = 0, dla wiersza 1 korekta = +3, dla wiersza 2 korekta = +6 itd.

Adresy w tablicy statycznej

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]

Adresy względne w tablicy dynamicznej

+0	+1	+2
+3	+4	+5
+6	+7	+8

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

void wyswietlTab(int* tablica, int lW, int lK);
void wypelnijA(int* tablica, int lW, int lK);

int main(int argc, char** argv) {
    //deklaracje zmiennych lokalnych
    int* tablica; //tablica do wypelnienia
    int lW, lK; //liczba wierszy i kolumn

    printf("Podaj liczbe wierszy: ");
    scanf("%i", &lW);
    printf("Podaj liczbe kolumn: ");
    scanf("%i", &lK);

    tablica=(int*)calloc(lW*lK,sizeof(int)); //rezerwacja pamieci
    if (tablica!=NULL) {                      //kontrola przydzialu pamieci
        wypelnijA(tablica,lW,lK);             //wstawianie wartosci do tablicy
        wyswietlTab(tablica,lW,lK);           //wyswietlanie gotowej tablicy
    }
    free(tablica);                            //zwolnienie pamieci
} else {
```

```
    printf("Bład przydziału pamieci!\n");
}

getchar();
return (EXIT_SUCCESS);
}

//wyswietlanie tablicy
void wyswietlTab(int* tablica, int lW, int lK) {
    int w, k; //numer kolumny i wiersza, zmienne lokalne

    //wyswietlanie gotowej tablicy
    for (w = 0; w < lW; w++) {
        for (k = 0; k < lK; k++) {
            printf("%3i", tablica[w*lK+k]);
        }
        printf("\n");
    }
}

//wypelnianie tablicy wzorem A
void wypelnijA(int* tablica, int lW, int lK){
    int w, k;        //numer kolumny i wiersza, zmienne lokalne
    int nr;           //numer do wpisania

    nr = 0;
    for (w = 0; w < lW; w++) {        //petla "po wierszach"
        for (k = 0; k < lK; k++) {    //petla "po kolumnie w danym wierszu"
            nr++;
            tablica[w*lK+k] = nr;    //wstawianie elementow w odpowiednie miejsca
                                     //w pamieci, wzgledem poczatku tablicy
        }
    }
}
}
```

Przykładowy efekt działania programu:

```
Podaj liczbe wierszy: 3
Podaj liczbe kolumn: 4
 1  2  3  4
 5  6  7  8
 9 10 11 12
```

#### 4.10.2 Zadanie

Zmodyfikuj program, tak by ponumerował pola tablic wg wzoru zadanego poniżej. Nowa tablica powinna być wyświetlana razem z poprzednią.

W				
1	0	2	0	3
0	0	0	0	0
4	0	5	0	6
0	0	0	0	0
7	0	8	0	9

## 4.11 Ćwiczenie 11 – Obsługa plików, typ strukturalny

### 4.11.1 Przykład

Utwórz nowy projekt i napisz program, który odczyta dane z pliku binarnego do struktury, a następnie ją wyświetli. Plik binarny *zwierz.bin* pobierz ze strony kursu w eNauczaniu i umieść w głównym katalogu projektu.

Odpowiedni kod źródłowy programu znajduje się poniżej:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
/* sciezka do pliku binarnego */
#define PLIK "./zwierz.bin"

/* zdefiniowanie struktury */
struct istota {
    char nazwa[20];
    float wzrost;
    int masa;
    int wiek;
    bool ogon;
};

int main(int argc, char** argv) {
    /* wskaznik do strumienia pliku */
    FILE *plik;
    /* stworzenie struktury zwierz */
    struct istota zwierz;

    /* otwarcie pliku w trybie odczytu binarnego */
    plik = fopen(PLIK, "rb");

    /* sprawdza, czy udalo sie otworzyc plik - czy strumien nie jest pusty */
    if (plik == NULL) {
        printf("Blad. Nie mozna otworzyc pliku!\n");
        return (EXIT_FAILURE); //zakonczenie programu w przypadku braku pliku
    } else {
        /* dopoki nie ma konca pliku czytaj */
        while (!feof(plik)) {
            fread(&zwierz, sizeof (zwierz), 1, plik);
        }
        /* zamkniecie strumienia */
        fclose(plik);
    }

    /* wyswietlenie danych */
    printf("Ten zwierz to %s\n", zwierz.nazwa);
    printf("Zyje do %i lat\n", zwierz.wiek);
    printf("Osiaga wzrost %2.2f m i mase do %i kg\n", zwierz.wzrost, zwierz.masa);
    if (zwierz.ogon) {
        printf("Posiada ogon\n");
    } else {
        printf("Nie posiada ogona\n");
    }

    _getch();
    return (EXIT_SUCCESS);
}
```

Zwróć uwagę, że mimo, iż w pliku *zwierz.bin* zapisano dane tylko jednego zwierzęcia, to w powyższym programie zawartość pliku jest odczytywana w pętli. Takie zachowanie

programu będzie przydatne w dalszej części zdania, gdy plik będzie mógł zawierać większą liczbę danych.

Efekt działania programu:

```
Ten zwierz to lama
Zyje do 20 lat
Osiaga wzrost 1.80 m i mase do 200 kg
Posiada ogon
```

#### 4.11.2 Zadanie

Utwórz nowy projekt i napisz program, który na podstawie danych wpisanych przez użytkownika zapisze do pliku binarnego strukturę definiującą inne zwierzę. Struktura musi być zgodna ze strukturą zdefiniowaną w przykładzie (zobacz kod źródłowy w przykładzie oraz podany poniżej) tak, by możliwe było odczytanie jej za pomocą poprzedniego programu. Program powinien tworzyć plik z rozszerzeniem *.bin*. Do zapisu danych do pliku użyj funkcji *fwrite()*.

```
struct istota {
    char nazwa[20];
    float wzrost;
    int masa;
    int wiek;
    bool ogon;
};
```

Efekt działania programu:

```
Podaj nazwe nowego pliku : nowe_zwierze.bin
Nazwa          : alpaka
Maksymalny wzrost : 0.99
Maksymalna masa   : 84
Maksymalny wiek   : 25
Czy posiada ogon ? (0- nie , 1-tak ) : 1
Dane zapisane do pliku nowe_zwierze.bin
```

#### 4.11.3 Zadanie dodatkowe \*

Utwórz nowy projekt i napisz program, który umożliwi stworzenie i edycję prostej bazy zwierząt. Struktura musi być zgodna z poniższym schematem (zwróć uwagę na dodatkowe pole *id*).

```
struct istota {
    int id;
    char nazwa[20];
    float wzrost;
    int masa;
    int wiek;
    bool ogon;
};
```

Wymagane funkcjonalności programu:

- odczyt rekordów z pliku binarnego,
- zapis danych do pliku,



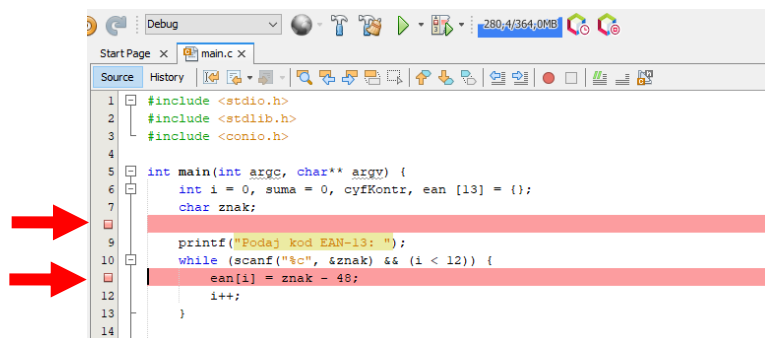
- wyświetlenie listy z nazwami wszystkich zwierząt posegregowanych wg *id*,
- usunięcie rekordu o danym *id*,
- *id* musi być nadawane automatycznie i nie może ulegać zmianie po usunięciu innego rekordu,
- program musi posiadać proste menu, umożliwiające wybór odpowiedniej funkcji.

Można przyjąć, że maksymalna liczba struktur w pliku nie przekroczy 30.

## 4.12 Ćwiczenie 12 – Debugowanie programu

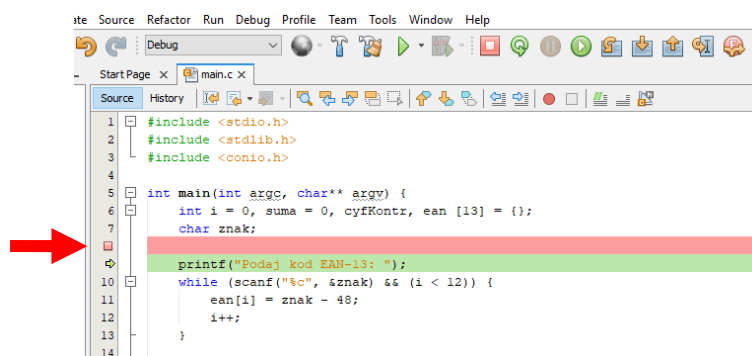
### 4.12.1 Korzystanie z debuggera

Debugger jest narzędziem ułatwiającym wyszukiwanie błędów i testowanie programu. Żeby z niego skorzystać, należy w pierwszej kolejności wskazać punkty, w których wykonywanie programu ma być wstrzymane (punkty zatrzymania, tzw. *breakpoint*). Kliknięcie LPM na numerze linii kodu programu utworzy *breakpoint*. Możliwe jest dodanie wielu punktów zatrzymania. Linia zostanie podświetlona na czerwono i pojawi się ikona czerwonego kwadratu (Rys. 9). Ponowne kliknięcie dezaktywuje *breakpoint*.



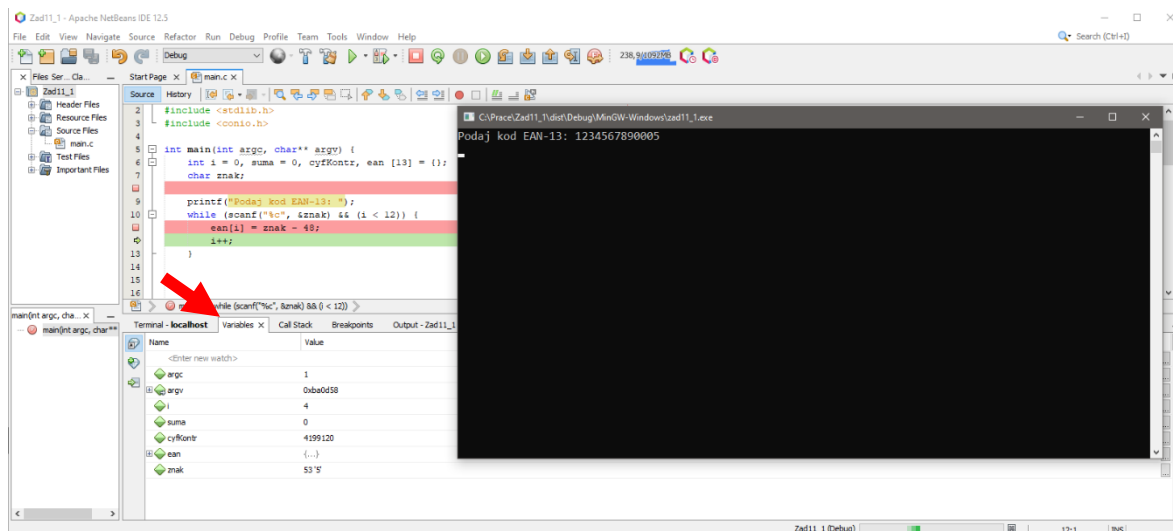
Rys. 9

Następnie należy wybrać z menu *Debug > Debug Project* lub wcisnąć klawisze *Ctrl+F5* lub kliknąć ikonę . Program zostanie uruchomiony, po czym jego wykonanie zostanie wstrzymane na pierwszej instrukcji z / za pierwszym napotkanym punktem zatrzymania (Rys. 10).



Rys. 10

Aktualne wartości zmiennych programu można sprawdzić w zakładce *Variables* (Rys. 11). Jeśli jakiejś zmiennej nie ma na liście, można ją dodać poprzez zaznaczenie jej nazwy w kodzie źródłowym, kliknięcie PPM i wybranie z menu *New Watch...*



Rys. 11

Do kontroli przebiegu programu w trybie debugowania służą polecenia dostępne w menu *Debug* lub przyciski na pasku narzędzi NetBeans (Rys. 12):

1. *Finish debugger session* - przerywa wykonywanie programu i debugowanie (*Shift+F5*),
2. *Restart* - ponownie uruchamia program,
3. *Pause* - wstrzymuje wykonanie programu,
4. *Continue* - wznowia wykonanie programu (*F5*),
5. *Step Over* - wykonuje linię, jeśli jest tam funkcja, wchodzi do niej i zatrzymuje się na jej początku (*F8*),
6. *Step Into* - wykonuje linię, jeśli jest tam funkcja, wykonuje ją w całości (*F7*),
7. *Step Out* - wykonuje aktualną funkcję do końca, wychodzi z niej i zatrzymuje się na kolejnym poleceniu (*Ctrl+F4*),
8. *Run to Cursor* - wykonuje program do miejsca zaznaczonego kursorem (*F4*).



Rys. 12

By przerwać pracę debuggera i zakończyć program należy wcisnąć *Shift+F5*. Należy pamiętać, by przerywać pracę debuggera zawsze, gdy chcemy powrócić do edycji kodu źródłowego, np. aby wprowadzić w nim poprawki. Ponowne uruchomienie programu bez zakończenia poprzedniej sesji debugowania może powodować trudności podczas jego wykonania.

#### 4.12.2 Zadanie

Utwórz nowy projekt i za pomocą debuggera znajdź błędy w programie sprawdzającym poprawność kodów kreskowych EAN13 (Rys. 13). Popraw program tak, by działał poprawnie. Wskaż (za pomocą komentarzy w kodzie) miejsca i typy poprawionych błędów.



Rys. 13

Algorytm sprawdzania poprawności kodu EAN13:

- Każdy kod posiada 13 cyfr. Zliczane są od lewej do prawej strony. 12 cyfr jest właściwym kodem, ostatnia (13-ta) jest cyfrą kontrolną.
- Cyfrom przyporządkowuje się odpowiednie wagi (mnożniki):
  - na pozycjach nieparzystych 1,
  - na pozycjach parzystych 3,
  - cyfra kontrolna nie posiada wagi.
- Oblicza się sumę 12-stu cyfr przemnożonych przez ich wagi.
- Oblicza się resztę z dzielenia (modulo) sumy cyfr. Jeśli wynik jest zerem, obliczona cyfra kontrolna równa jest 0. W przeciwnym przypadku cyfra kontrolna równa jest różnicy liczby 10 i otrzymanego wyniku.
- Poprawność kodu stwierdza się porównując zgodność obliczonej cyfry kontrolnej z ostatnią (13-tą) cyfrą w kodzie kreskowym.

Kod źródłowy programu, w którym należy znaleźć i poprawić błędy znajduje się poniżej:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main(int argc, char** argv) {
    int i = 0, suma = 0, cyfKontr, ean [13] = {};
    char znak;

    printf("Podaj kod EAN-13: ");
    while (scanf("%c", &znak) && (i < 12)) {
        ean[i] = znak - 48;
        i++;
    }

    for (i = 1; i < 12; i++) {
        if (i % 2) {
            suma += 1 * ean[i];
        } else {
            suma += 3 * ean[i];
        }
    }

    suma %= 10;
    if (suma) {
        cyfKontr = 0;
    } else {
        cyfKontr = 10 - suma;
    }

    if (cyfKontr == ean[12]) {
        printf("OK. Kod EAN-13 poprawny\n");
    } else {
        printf("Bład. Kod EAN-13 niepoprawny\n");
    }

    _getch();
    return (EXIT_SUCCESS);
}
```

Przykładowy efekt działania programu:

```
Podaj kod EAN-13: 1234567890005
OK. Kod EAN-13 poprawny
```

```
Podaj kod EAN-13: 4049649000039
OK. Kod EAN-13 poprawny
```

```
Podaj kod EAN-13: 5900334001818
OK. Kod EAN-13 poprawny
```

```
Podaj kod EAN-13: 5079667000011
Bład . Kod EAN-13 niepoprawny
```

## 4.13 Ćwiczenie 13 – Obsługa plików i wskaźniki

### 4.13.1 Zadanie dodatkowe \*

Utwórz nowy projekt i napisz program, który:

- odczyta plik tekstowy zawierający liczby zmiennopozycyjne umieszczone w kolejnych wierszach pliku,
- posortuje je malejąco,
- wyznaczy minimum, maximum,
- obliczy ich średnią arytmetyczną i medianę,
- zapisze posortowane dane do innego pliku tekstowego.

Program musi być zrealizowany z użyciem wskaźników (tablica dynamiczna, przydatne funkcje: *malloc( )*, *free( )*). Uwzględnij, że liczba próbek nie jest znana (być może przydatna będzie funkcja *realloc( )*). Pamiętaj także o kontroli przydziału pamięci, kontroli otwarcia pliku itp.

Zabronione jest korzystanie ze standardowych funkcji sortujących języka C – można zastosować jeden ze znanych algorytmów sortowania, ale trzeba go zaprogramować samodzielnie.

Przykładowe efekt działania programu:

dla pliku dane13\_1.txt

```
Liczba probek: 10
Min          : -2.655812
Max          : 3.549089
Srednia      : -0.376079
Mediana      : -0.727836
```

dla pliku dane13\_2.txt

```
Liczba probek: 1000
Min          : -4.997436
Max          : 6.995605
Srednia      : 0.913272
Mediana      : 0.874020
```

dla pliku dane13\_3.txt

```
Liczba probek: 568
Min          : -4.992981
Max          : 4.999390
Srednia      : 0.117849
Mediana      : 0.197760
```

## Spis treści

1	Cel przedmiotu .....	2
2	Wstęp.....	2
3	Tworzenie nowego projektu.....	2
4	Zadania projektowe do wykonania .....	5
4.1	Ćwiczenie 1 – Pierwszy program .....	5
4.1.1	Przykład.....	5
4.1.2	Zadanie .....	6
4.2	Ćwiczenie 2 – Komunikacja z użytkownikiem .....	6
4.2.1	Przykład.....	6
4.2.2	Zadanie .....	7
4.3	Ćwiczenie 3 – Funkcje i prototypy funkcji .....	8
4.3.1	Przykład.....	8
4.3.2	Zadanie * .....	9
4.4	Ćwiczenie 4 – Instrukcja warunkowa <i>if...else</i> .....	9
4.4.1	Przykład.....	9
4.4.2	Zadanie .....	10
4.4.3	Zadanie dodatkowe * .....	10
4.5	Ćwiczenie 5 – Instrukcja warunkowa <i>switch</i> .....	11
4.5.1	Przykład.....	11
4.5.2	Zadanie zbiorcze – Użycie instrukcji warunkowych .....	12
4.6	Ćwiczenie 6 – Pętla <i>for</i> .....	13
4.6.1	Przykład.....	13
4.6.2	Zadanie .....	13
4.6.3	Zadanie dodatkowe * .....	14
4.7	Ćwiczenie 7 – Tablice, pętla <i>for</i> .....	15
4.7.1	Przykład.....	15
4.7.2	Zadanie .....	16
4.7.3	Zadanie dodatkowe * .....	17
4.8	Ćwiczenie 8 – Pętla <i>while</i> .....	17
4.8.1	Przykład.....	17
4.8.2	Zadanie .....	18
4.9	Ćwiczenie 9 – Instrukcje warunkowe, pętla <i>do..while</i> , przechwytywanie kodów wciśniętych klawiszy z klawiatury .....	18
4.9.1	Przykład.....	18
4.9.2	Zadanie .....	20
4.10	Ćwiczenie 10 – Tablice i zmienne wskaźnikowe .....	21

4.10.1	Przykład.....	21
4.10.2	Zadanie .....	22
4.11	Ćwiczenie 11 – Obsługa plików, typ strukturalny .....	23
4.11.1	Przykład.....	23
4.11.2	Zadanie .....	24
4.11.3	Zadanie dodatkowe * .....	24
4.12	Ćwiczenie 12 – Debugowanie programu .....	25
4.12.1	Korzystanie z debuggera .....	25
4.12.2	Zadanie .....	26
4.13	Ćwiczenie 13 – Obsługa plików i wskaźniki .....	28
4.13.1	Zadanie dodatkowe * .....	28