

## Wstęp do programowania – laboratorium

1.	Środowisko pracy .....	3
1.1	Gcc online .....	3
1.2	Środowisko DevC++ .....	4
1.3	Rozpoczęcie pracy i główne opcje .....	5
1.4	Edycja programu .....	6
1.4.1	Główne funkcje edycji .....	6
1.4.2	Przenoszenie bloków .....	7
1.4.3	Formatowanie kodu i wcięcia .....	8
1.5	Uruchamianie programu .....	10
1.5.1	Uruchamianie programu - metoda elementarna .....	10
1.5.2	Tworzenie projektu .....	12
1.5.3	Opcje projektu .....	14
1.5.4	Błędy kompilacji – sygnalizacja i poprawa .....	15
1.6	Debugowanie kodu .....	16
1.6.1	Przygotowanie gdy program jest pojedynczym plikiem .....	16
1.6.2	Przygotowanie gdy posługujemy się projektem .....	17
1.6.3	Przebieg uruchamiania .....	17
1.7	Tworzone pliki .....	20
1.8	Zadania .....	21
1.8.1	Uruchomienie programu .....	21
1.8.2	Użycie edytora .....	21
1.8.3	Kompilacja i uruchomienie .....	21
1.8.4	Kompilacja i uruchomienie - projekt .....	21
1.8.5	Debugowanie kodu .....	22
2.	Podstawowe operacje wejścia wyjścia .....	23
2.1	Standardowe wejście i wyjście .....	23
2.2	Funkcja putchar() .....	23
2.3	Funkcja getchar() .....	24
2.4	Funkcja printf .....	24
2.5	Funkcja scanf .....	28
3.	Typy danych, stałe, zmienne, wyrażenia arytmetyczne i logiczne .....	30
3.1	Pole i obwód koła .....	30
3.2	Pole trójkąta .....	30
3.3	Kalkulator .....	30
3.4	Kody ascii .....	30
3.5	Reprezentacja liczb jako dec, hex .....	30
3.6	Operator sizeof() .....	30
3.7	Funkcje matematyczne .....	30
3.8	Porównania 2 .....	31
4.	Binarna reprezentacja informacji .....	32
4.1	Rodzaje informacji reprezentowane w komputerach .....	32
4.2	Pozycyjne systemy liczbowe .....	32
4.3	Przekształcanie liczb między systemami pozycyjnymi .....	33
4.4	Kodowanie tekstu .....	35
4.5	Ćwiczenia .....	37
4.5.1	Zamiana liczb dziesiętnych na binarne , ósemkowe i szesnastkowe .....	37
4.5.2	Zamiana liczb binarnych na dziesiętne, ósemkowe i szesnastkowe .....	38
4.5.3	Kodowanie znaków ASCII .....	39
5.	Instrukcje sterujące i instrukcje iteracji .....	40
5.1	Ciąg Fibonacciego .....	40
5.2	Pętla i warunek .....	40
5.3	Pętla i wartość średnia .....	40
5.4	Ile jest liczb parzystych w zakresie? .....	40
5.5	Liczenie jedynek w reprezentacji binarnej .....	40

5.6	Trójkąt .....	41
5.7	Zliczanie znaków .....	41
5.8	Pętla while.....	41
5.9	Tablica sum liczb.....	42
5.10	Tabliczka mnożenia.....	42
5.11	Szyfr Cezara .....	43
5.12	Sklejanie tekstów .....	44
6.	Tablice i Funkcje.....	45
6.1	Znajdowanie maksimum w tablicy liczb.....	45
6.2	Znajdowanie maksimum w tablicy liczb – funkcja.....	45
6.3	Sumowanie elementów tablicy - funkcja.....	45
6.4	Sortowanie bąbelkowe .....	46
6.5	Sortowanie bąbelkowe – funkcja .....	46
7.	Łańcuchy i napisy.....	47
7.1	Reprezentacja łańcuchów w języku C.....	47
7.2	Obliczanie długości napisu .....	52
7.3	Zliczanie słów w łańcuchu tekstowym .....	52
7.4	Liczenie podłańcuchów w łańcuchu .....	54
7.5	Sprawdzanie hasła .....	54
7.6	Argumenty funkcji main - kalkulator .....	55
7.7	Sortowanie tablicy łańcuchów.....	56
7.8	Sortowanie 2 tablicy łańcuchów.....	57
8.	Tablice, wektory, macierze.....	59
8.1	Implementacja stosu w oparciu o tablicę.....	59
8.2	Szukanie binarne (ang. <i>binary search</i> ).....	61
8.3	Dodawanie i mnożenie wektorów.....	64
8.4	Funkcje i tablice 2 wymiarowe .....	64
8.5	Działania na macierzach .....	66
8.6	Zarządzanie bazą danych opartą na tablicy .....	67
9.	Operacje na plikach – funkcje niskiego poziomu .....	70
9.1	Odczyt pliku .....	70
9.2	Kopiowanie plików.....	70
9.3	Baza danych osobowych oparta na tablicy, rozszerzenie o zapis do pliku.....	70
9.4	Przykład użycia plików – baza danych Hotel.....	72
10.	Operacje na plikach – biblioteka standardowa .....	75
10.1	Zapisywanie linii tekstu do pliku.....	75
10.2	Odczyt z pliku tekstowego i liczenie linii.....	75
10.3	Baza danych - tablica, rozszerzenie o zapis do pliku bibl. standard.....	76
10.4	Odczyt z pliku tekstowego i liczenie słów kluczowych .....	78
10.5	Baza danych hotel.....	84
11.	Wskaźniki.....	87
11.1	Dostęp do elementów tablicy.....	87
11.2	Lista jednokierunkowa .....	89
12.	Źródła.....	95

# 1. Środowisko pracy

## 1.1 Gcc online

Gcc online jest środowiskiem kompilacji i uruchamiania programów w języku C (a także w innych językach) poprzez przeglądarkę internetową. Dostępny jest pod adresem:

[https://www.onlinegdb.com/online\\_c\\_compiler](https://www.onlinegdb.com/online_c_compiler)

Po uruchomieniu pojawi się następujący ekran:

```

1  /* *****
2
3
4      Online C Compiler.
5      Code, Compile, Run and Debug C program online.
6      Write your code in this editor and press "Run" button to compile and execute it.
7      *****
8
9      #include <stdio.h>
10
11     int main()
12     { printf("Hello World");
13       return 0;
14     }
15
16

```

Ekran 1-1 Środowisko onlinegdb

Po wpisaniu kodu do okna edycji i wciśnięciu klawisza Run program zostanie skompilowany i uruchomiony. Wynik będzie widoczny w oknie jak poniżej.

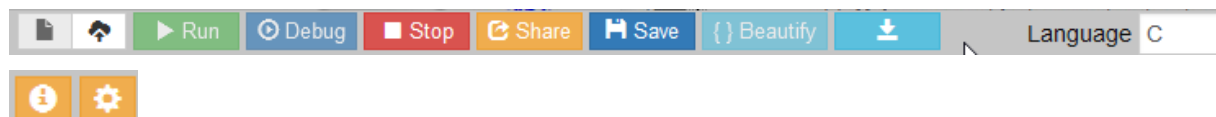
```

Hello World

...Program finished with exit code 0
Press ENTER to exit console.

```

W górnej części okna znajduje się belka z menu głównym.









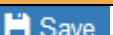




	Nowy plik
	Wczytaj nowy plik z dysku
	Skompiluj i uruchom program
	Uruchamiaj (ang. debug) program
	Zatrzymaj program
	Pobierz link do programu (w celu udostępnienia)
	Zapisz program na dysku lokalnym
	Uporządkuj kod
	Zapisz program
	Informacje o ustawieniach i skrótach klawiszowych
	Ustawienia środowiska i kompilatora

Tabela 1-1 Głównie opcje środowiska gdbonline

## 1.2 Środowisko DevC++

Dev-C++ jest kompletnym środowiskiem programistycznym IDE (ang. *Integrated Development Enviroment*) rozwijanym w ramach projektu Bloodshed Software<sup>1</sup>. Środowisko to wykorzystuje środowisko programistyczne GNU (ang. Gnu is Not Unix) w tym kompilator GCC pracujący w środowisku emulatora Linuxa o nazwie MinGW (ang. *Minimalist GNU for Windows*). Całość jest dostępna za darmo na licencji GPL. Pakiet instalacyjny można pobrać ze strony:

<http://www.bloodshed.net/index.html>

Środowisko między innymi zawiera:

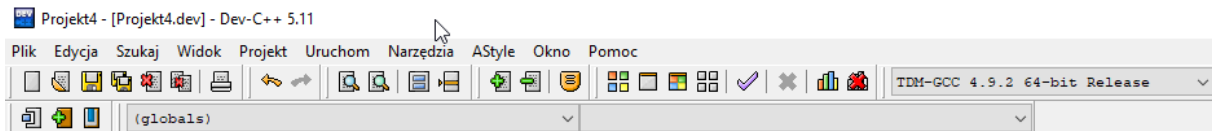
- Kompilator GCC
- Zintegrowany program uruchomieniowy (ang. *Debugger*)
- Zintegrowany Edytor podświetlający składnię
- Wsparcie dla wielu języków narodowych
- Narzędzie kompletowania kodu
- Listowanie funkcji
- Administrator projektów
- Możliwość instalacji pakietów rozszerzających

### 1.3 Rozpoczęcie pracy i główne opcje

Pracę z programem rozpoczynamy klikając w jego ikonę na pulpicie.



Główna belka programu umożliwia wybór poszczególnych opcji.

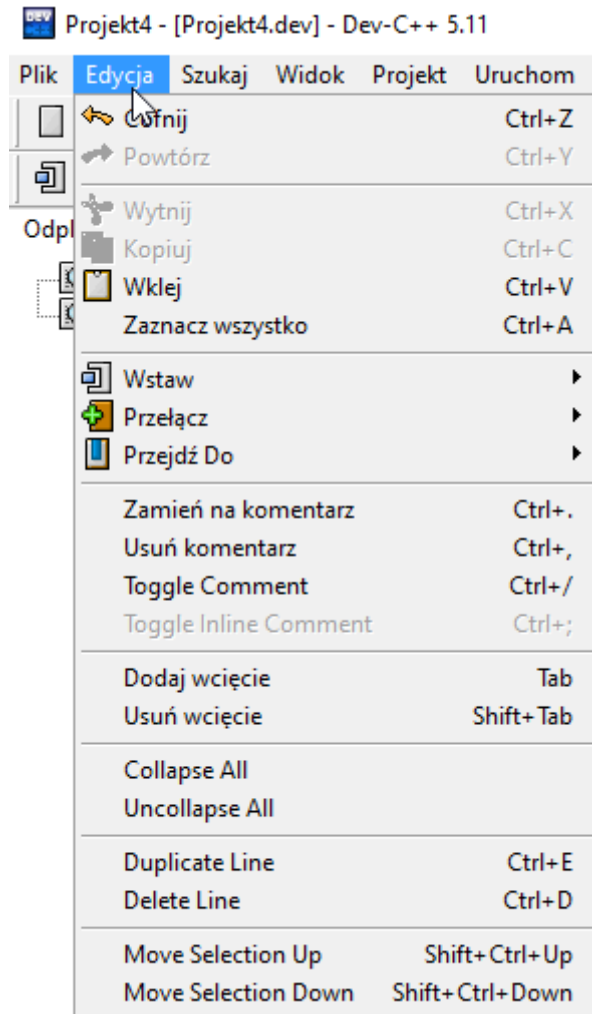


- Plik - Otwieranie i zamykanie plików i projektów
- Edycja - Edycja kodu programu
- Szukaj - Edycja, znajdowanie i zastępowanie tekstu
- Widok - Konfiguracja interfejsu IDE
- Projekt - Zarządzanie projektem, dodawanie i usuwanie plików
- Uruchom - Kompilacja, uruchamianie, debugowanie kodu
- Narzędzia - Ustawianie opcji kompilatora i środowiska
- AStyle - Ustawianie opcji wyglądu i wyrównania kodu
- Okno - Zarządzanie oknami środowiska
- Pomoc - Pomoc (niezbyt obszerna niestety)

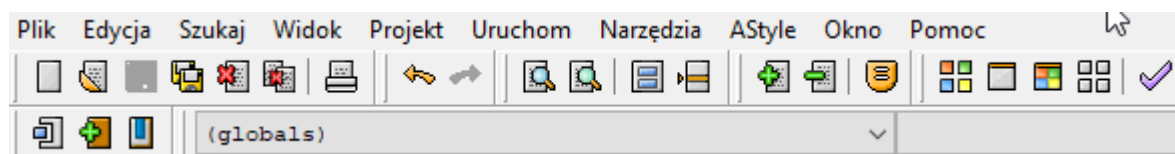
## 1.4 Edycja programu

### 1.4.1 Główne funkcje edycji






Środowisko DevC++ posiada wbudowany, dość rozbudowany edytor. Umożliwia on wykonywanie edycji programu i zawiera liczne ułatwienia.






Pod górną belką znajduje się belka narzędzi zawierająca skróty najczęściej wykonywanych czynności.

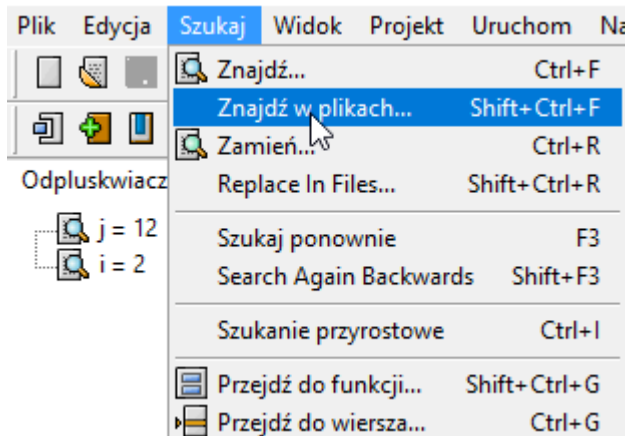


Wyjaśnienie niektórych ikon dane jest poniżej.

-  - Nowy plik
-  - Otwórz plik lub projekt
-  - Zapisz plik
-  - Zapisz wszystko
-  - Zamknij plik

-  - Cofnij ostatnią zmianę (przydatne)
-  - Szukaj tekstu
-  - Zastąp tekst innym

Wiele przydatnych funkcji edycyjnych znajduje się w podmenu Szukaj co pokazano poniżej.



#### 1.4.2 Przenoszenie bloków

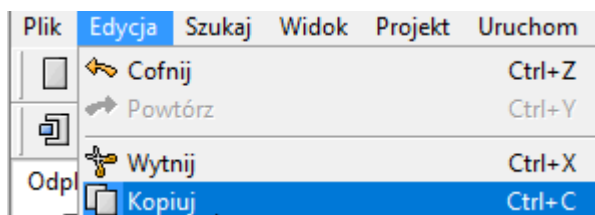
Przy pisaniu programów często występuje potrzeba kasowania i przenoszenia bloków.

```

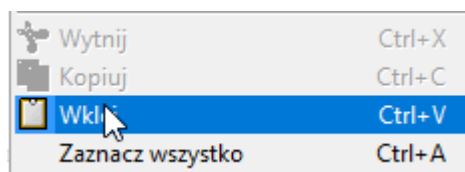
10 main() {
11     int i;
12     for(i=0;i<10;i++) {
13         if() {
14             }
15     }
16 }
17

```

Wykonuje się to zaznaczając blok w kodzie źródłowym a następnie naciskając (Ctrl + C) lub Edycja / Kopiuj.

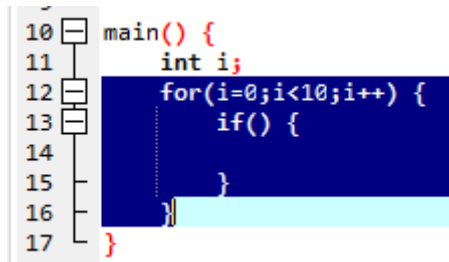


Następnie ustawiamy kursor w miejscu docelowym i naciśnij (Ctrl + V) lub użyj prawego klawisza myszy i wybierz opcję wklej.



### 1.4.3 Formatowanie kodu i wcięcia

Czytelność kodu jest niezwykle ważna. Zasięg działania bloku instrukcji uwidaczniana jest przez wcięcia. Jest to ważne gdyż blok określa zasięg instrukcji, tak więc trzeba widzieć wyraźnie gdzie blok się zaczyna i gdzie się kończy.



```

10 main() {
11     int i;
12     for(i=0; i<10; i++) {
13         if() {
14
15         }
16     }
17 }

```

Stosowane są dwie główne metody formatowania kodu których przykłady podano poniżej. Sposób 1 jest bardziej zwarty.

```

if(...) {
    ...
} else {
    ...
}
for(...) {
    ...
}

```

Przykład 1-1 Sposób 1 formatowania kodu

```

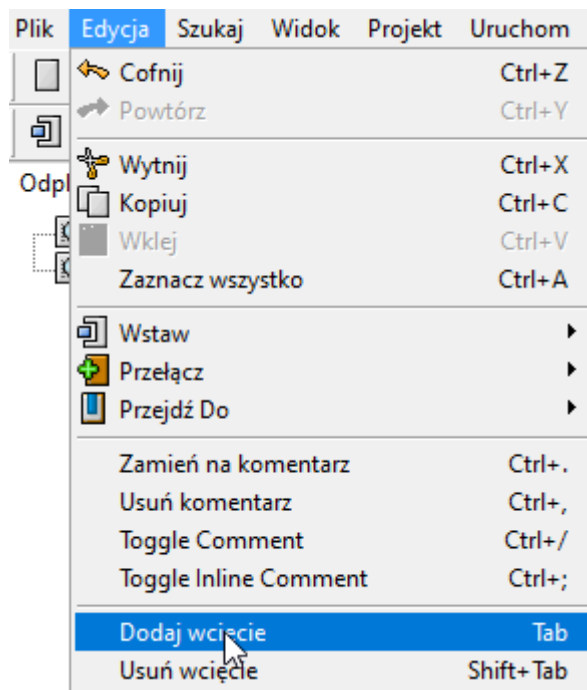
if(...)
{
    ...
} else
{
    ...
}
for(...)
{
    ...
}

```

Przykład 1-2 Sposób 1 formatowania kodu

Aby uzyskać wcięcie zaznaczamy określoną liczbę linii i używamy opcji Edycja / Dodaj wcięcie.



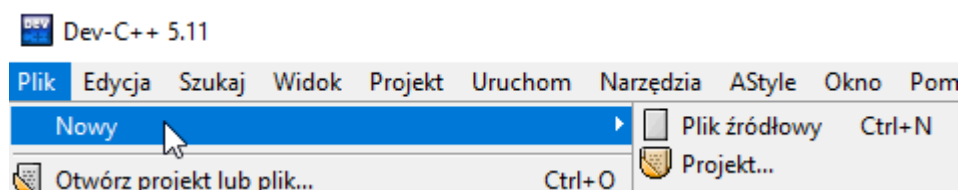


## 1.5 Uruchamianie programu

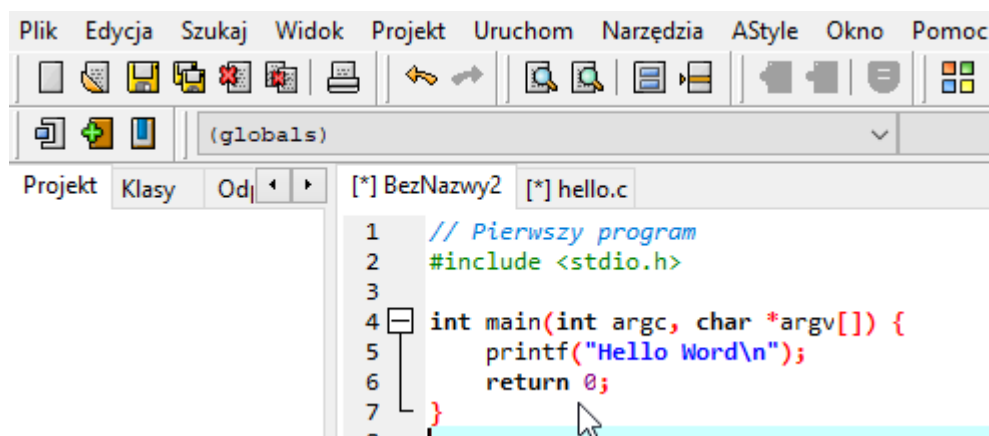
### 1.5.1 Uruchamianie programu - metoda elementarna

Aby utworzyć prosty program składający się z jednego pliku należy wykonać następujące kroki:

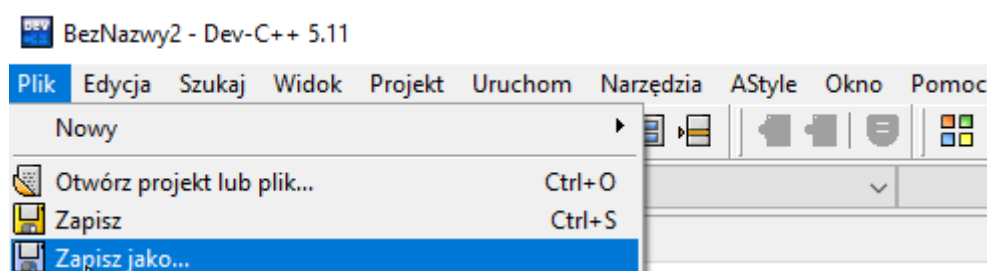
- 1) Z górnej belki środowiska wybrać opcje: Plik / Nowy /Plik źródłowy, tak jak pokazuje poniższy ekran.



Po uruchomieniu edycji wpisać tekst programu **hello.c** jak niżej.



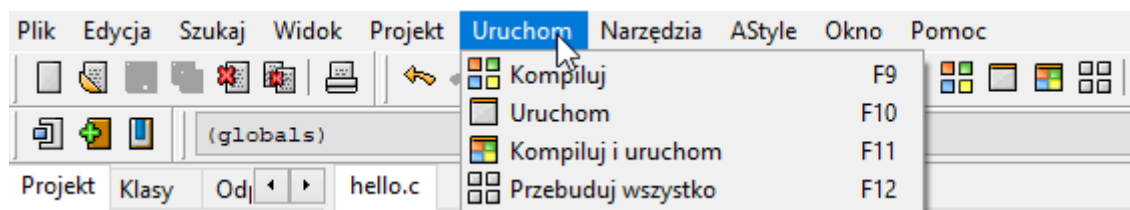
- 2) Wybrać opcję: Plik / Zapisz jako



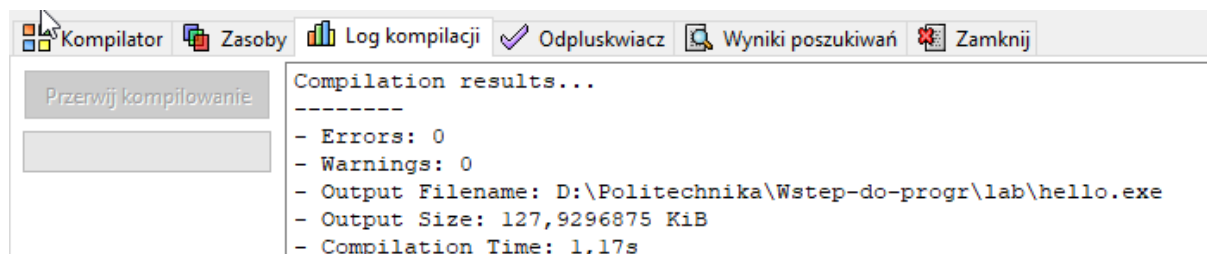
Z listy Zapisz jako typ: należy wybrać opcję C source files (\*.c) i jako nazwę pliku podać **hello.c**.



- 3) Z górnej belki wybrać opcję: Uruchom / Kompiluj (lub nacisnąć F9).



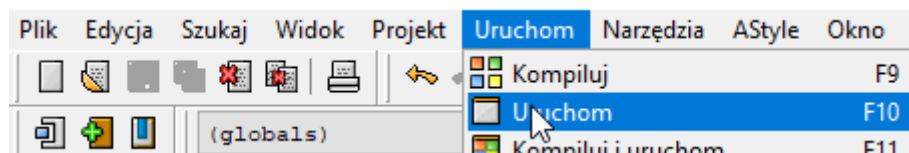
Raport z kompilacji pojawi się w dolnej części ekranu.



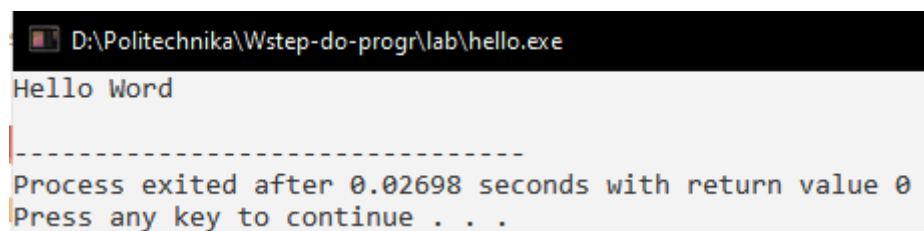
Jakby wystąpiły jakieś błędy należy wrócić do edytora i je poprawić. W oknie kompilacji pojawi się komunikat o liczbie błędów kompilacji (ang. Errors). W tym przypadku błędów nie było, gdyż Errors: 0. Znaczenie komunikatów dane zostało poniżej.

- Errors - liczba błędów
- Warnings - liczba ostrzeżeń
- Output Filename - nazwa pliku wykonywalnego
- Output Size - wielkość pliku wykonywalnego
- Compilation Time - czas kompilacji

4) Gdy błędów nie było należy program uruchomić wybierając z menu opcję; Uruchom / Uruchom lub naciskając klawisz F10.



Gdy wszystko poszło dobrze, zobaczymy okno terminala jak niżej z wynikiem działania programu.



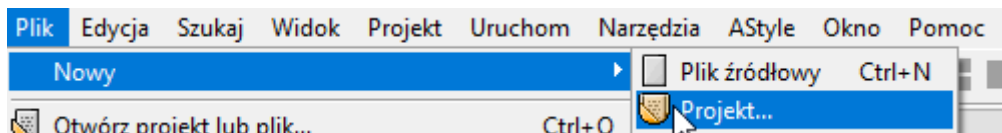
Naciśnięcie dowolnego klawisza zamknie okno terminala.

### 1.5.2 Tworzenie projektu

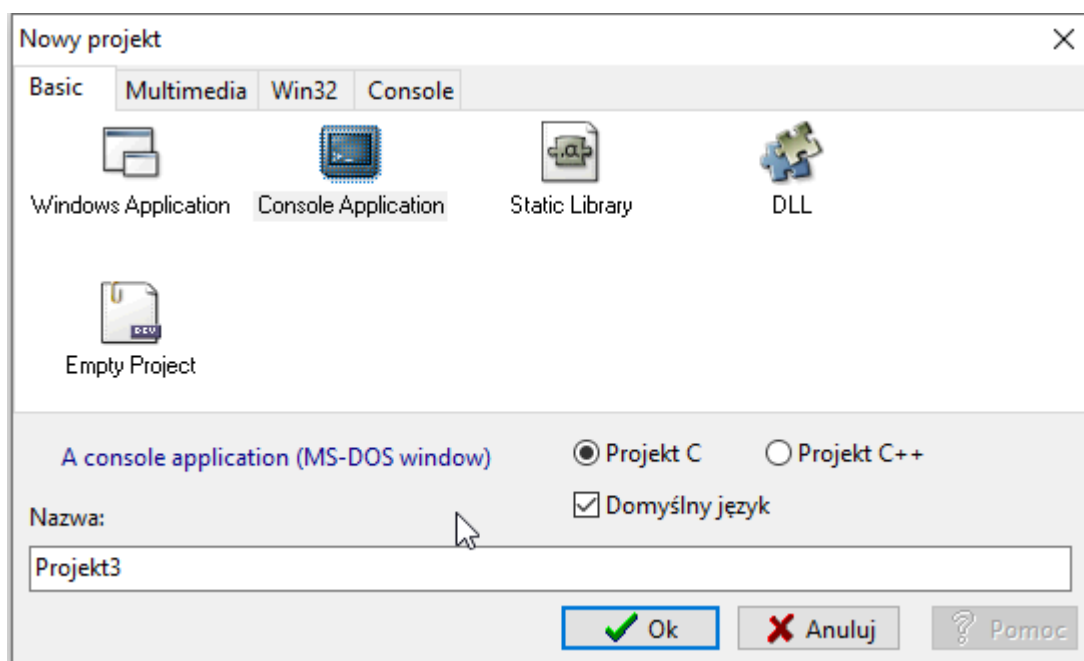
Program nie zawsze jest tak prosty jak w poprzednim przypadku. Często składa się on z wielu plików, wymaga też podania wielu dodatkowych informacji jak opcje kompilatora, linkera, informacje o bibliotekach i innych. Wtedy posługujemy się projektem. Oprócz pliku z programem, zawiera on być może inne pliki a także informacje dodatkowe.

Aby utworzyć projekt wykonujemy następujące kroki.

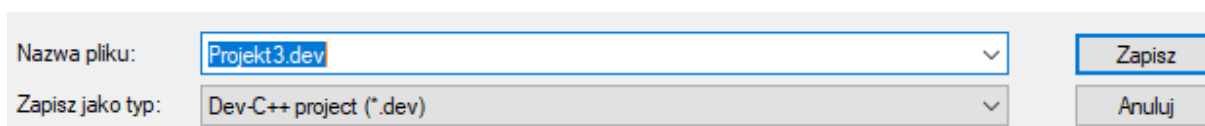
1. Z menu głównego wybieramy opcje: Plik / Nowy / Projekt.



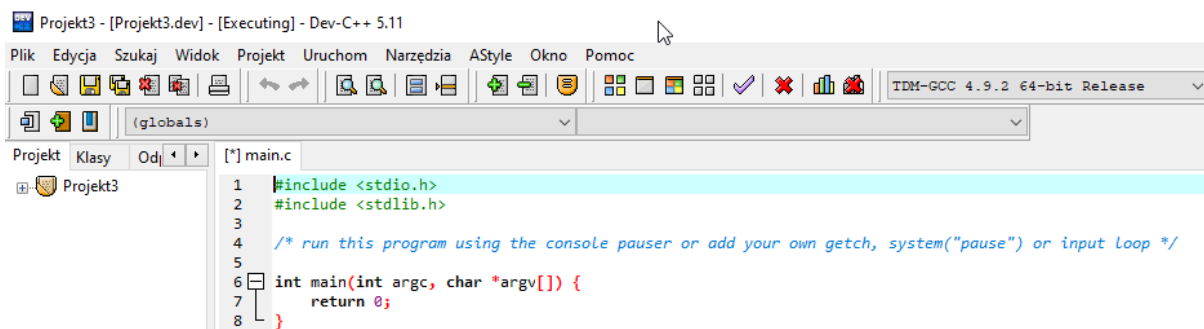
Pojawi się nowe okno jak poniżej.



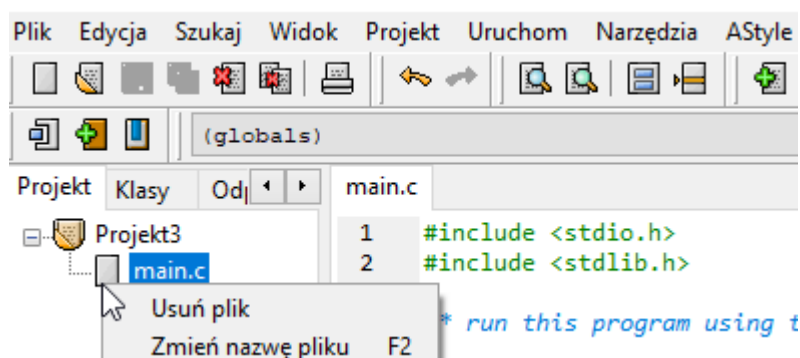
2. Wybieramy opcje: Basic / Console Application. Klikamy w guzik Projekt C. W oknie Nazwa: wpisujemy nazwę projektu (może być dowolna). Dalej klikamy Ok. Pojawi się zapytanie o nazwę pliku z projektem.



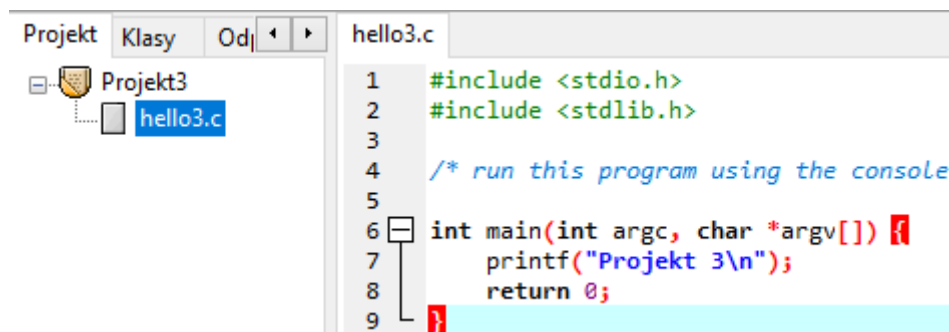
Możemy zaakceptować podaną nazwę lub wpisać inną. Klikamy w klawisz Zapisz. Gdy program ma się składać z wielu plików dobrze jest utworzyć dla tego projektu nowy folder i tam zapisywać wszystkie jego pliki. Po zapisie pojawi się formularz jak poniżej.



Zawiera on szkic kodu. Domyślna nazwa pliku głównego (zawierającego funkcję main) nazywa się main.c. Lepiej zmienić tę nazwę. Wykonujemy to rozwijając ikonę Projekt3 i klikając prawym klawiszem myszki lub wciskamy klawisz F2.

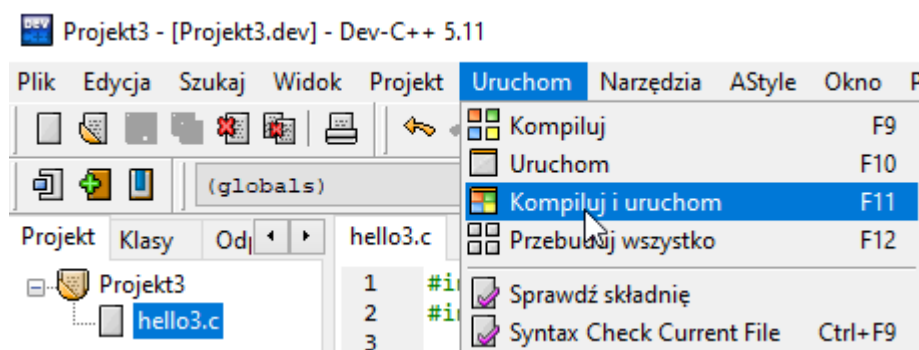


3. Uzupełniamy kod wpisując taki kod jaki chcemy.

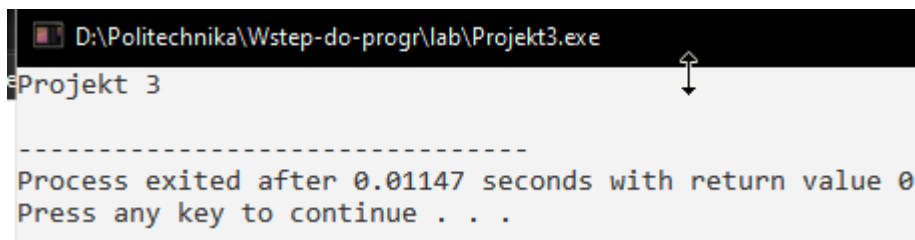


Zauważmy że w lewej części formularza widzimy pliki wchodzące w skład projektu.

4. Uruchamiamy kompilację a potem wykonanie wybierając opcje: Uruchom / Kompiluj (F9) a potem Uruchom / Uruchom (F10). Można też wybrać opcję Uruchom / Kompiluj i uruchom (F11). Pokazuje to poniższy ekran.



Po uruchomieniu pokaże się ekran konsoli jak poniżej.



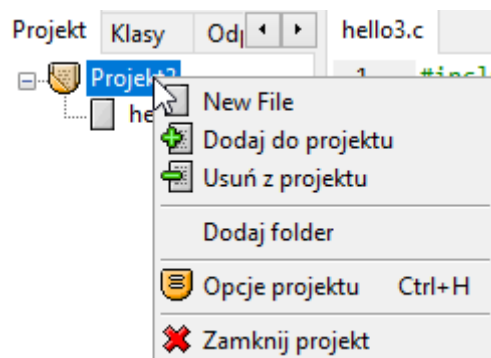
```

D:\Politechnika\Wstep-do-progr\lab\Projekt3.exe
Projekt 3
-----
Process exited after 0.01147 seconds with return value 0
Press any key to continue . . .

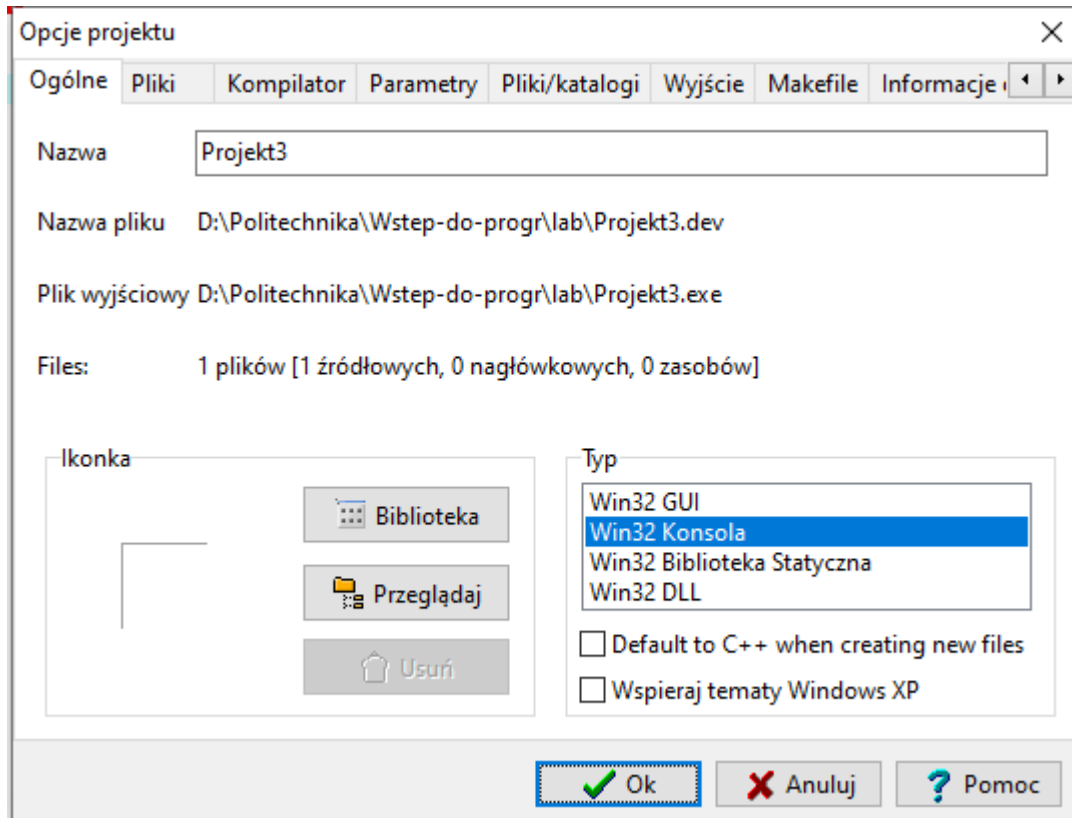
```

### 1.5.3 Opcje projektu

Opcje projektu możemy obejrzeć i ustawić klikając prawym klawiszem myszy w ikonę projektu co pokazano poniżej.



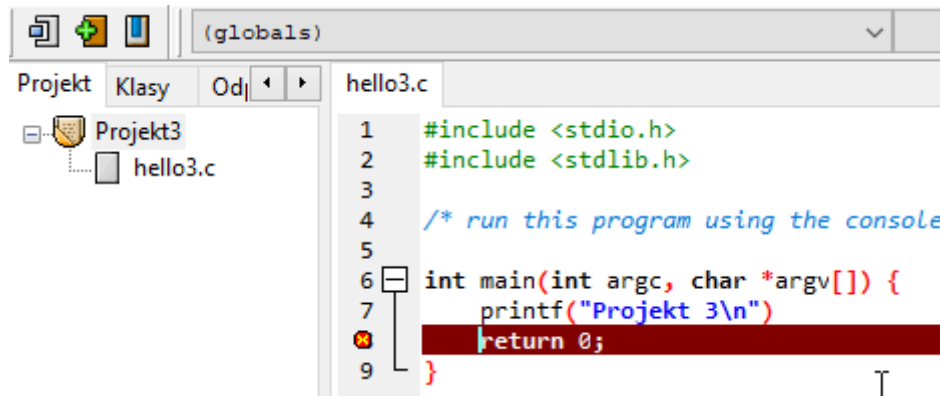
Pojawi się formularz projektu.



W formularzu są zakładki które zawierają różne opcje projektu.

### 1.5.4 Błędy kompilacji – sygnalizacja i poprawa

W trakcie kompilacji zwykle pojawiają się błędy. Tak też jest w poniższym przykładzie gdzie brak średnika po funkcji printf.



```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* run this program using the console */
5
6  int main(int argc, char *argv[]) {
7      printf("Projekt 3\n")
8      return 0;
9  }
  
```

Błąd sygnalizowany jest znakiem x linii kodu. Opis błędu podany jest w oknie kompilacji.

		Kompilator (3)		Zasoby		Log kompilacji		Odpluskwiacz		Wyniki poszukiwań		Zamknij
Wi...	Kol	Plik	Komunikat									
		D:\Politechnika\Wstep-do-progr\lab\hello3.c	In function 'main':									
8	2	D:\Politechnika\Wstep-do-progr\lab\hello3.c	[Error] expected ';' before 'return'									
28		D:\Politechnika\Wstep-do-progr\lab\Makefile.win	recipe for target 'hello3.o' failed									

W tym przypadku jest to komunikat: expected ';' before 'return'. Poprawiamy błąd wracając do edytora i ponownie uruchamiamy kompilację.

## 1.6 Debugowanie kodu

Nawet jak program skompiluje się poprawnie to zwykle nie zachowuje się tak jak chcemy – zawiera różne błędy. Proces usuwania błędów nazywa się debugowaniem.

Nazwa pochodzi od angielskiego słowa *bug* – pluskwa, czyli jest to odpluskwanie programu. Słowo to ma swoją historię. We wczesnej epoce komputerów (lampowych jeszcze), pewien komputer przejawiał błędne działanie i nie udawało się odnaleźć przyczyny. Szczegółowe poszukiwania w szafach ze sprzętem ujawniły, że pomiędzy przewody dostała się pluskwa. Jako że lampy zasilane były dość wysokim napięciem (ok. 200 V), nieszczęsna pluskwa uległa zwęgleniu, powodując tym samym zwarcie w obwodach komputera.

Debugowanie kodu źródłowego odbywa się w podany dalej sposób. Do wykonania ćwiczeń posłużymy się danym niżej przykładem.

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

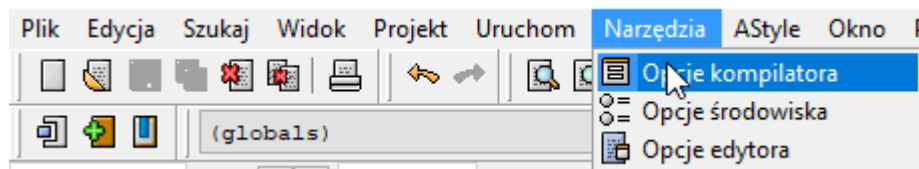
int main(void) {
    int i,j;
    printf("Witamy w Lab WSH\n");
    for(i=0;i<10;i++) {
        j=i+10;
        printf("Krok  %d\n",i);
    }
    printf("Koniec\n");
    return EXIT_SUCCESS;
}
```

Przykład 1-3 Program dbg\_test.c

### 1.6.1 Przygotowanie gdy program jest pojedynczym plikiem

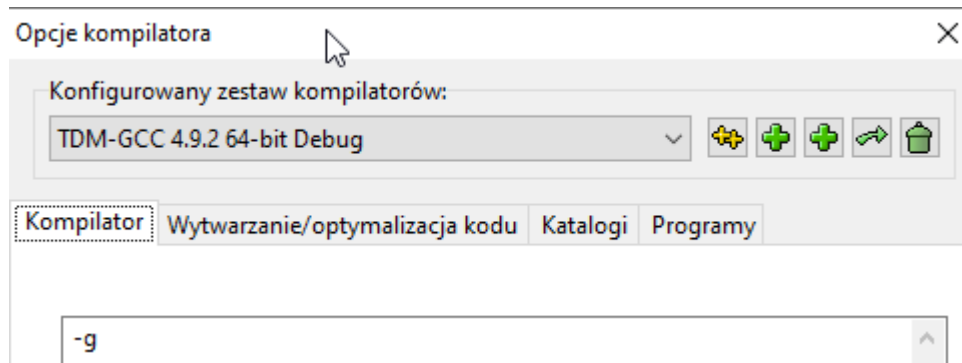
Aby przygotować program do uruchamiania należy wykonać podane poniżej kroki.

- 1) Otwórz lub utwórz plik z kodem źródłowym
- 2) W oknie Narzędzia/Opcje wybierz Opcje Kompilatora



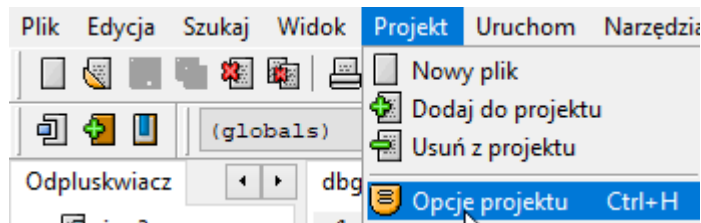
- 3) Z dostępnych kompilatorów wybierz taki który wspiera debugowanie. Może to być taki jak w poniższym ekranie. W oknie opcji kompilatora wpisz –g.



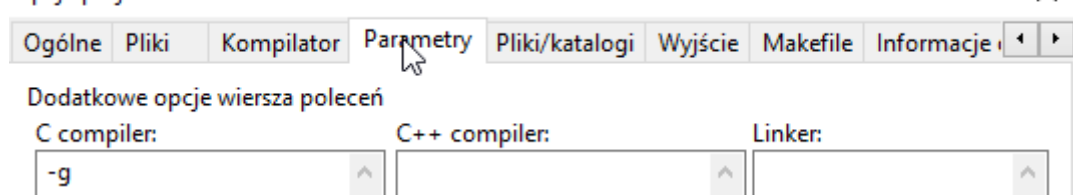


### 1.6.2 Przygotowanie gdy posługujemy się projektem

Gdy posługujemy się projektem, aby przygotować program do debugowania należy: Wybrać opcję Projekt / Opcje projektu.



W opcjach wybieramy zakładkę Parametry i w oknie C compiler wpisujemy `-g`



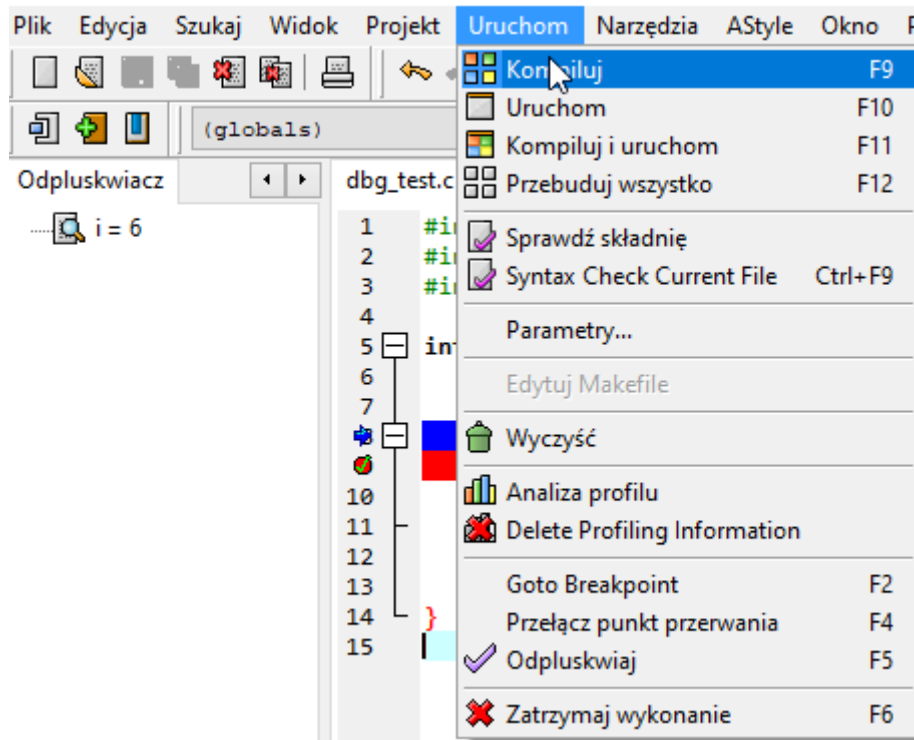
Dalej klikamy w klawisz Ok.

Na tym kończymy.

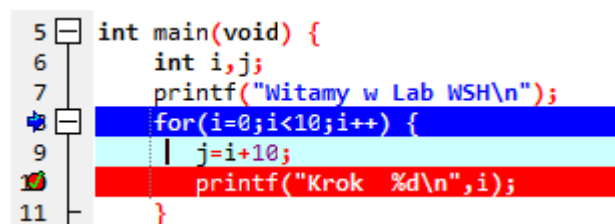
### 1.6.3 Przebieg uruchamiania

Gdy program przygotowany jest do debugowania możemy je już rozpocząć. W tym celu wykonujemy następujące kroki:

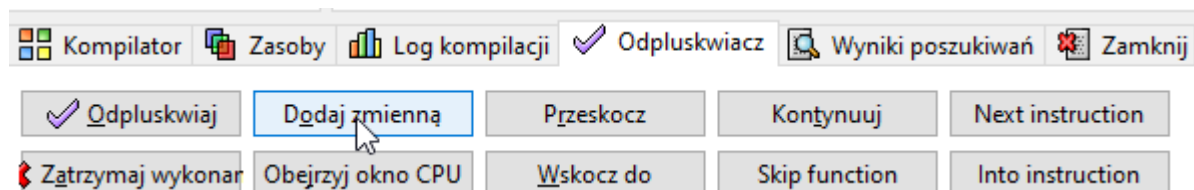
A) Wciśnij klawisz F5 lub wybierz opcję Odpluskwiał co pokazuje poniższy ekran.



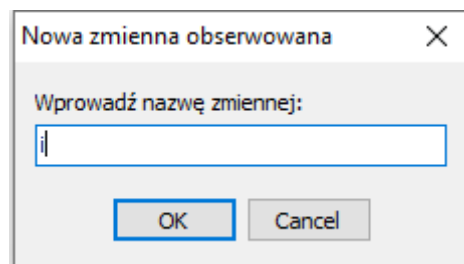
B) Umieść w kodzie źródłowym punkty zatrzymania (ang. *Breakpoint*). Można to zrobić klikając w numer wybranej linii w oknie kodu źródłowego.



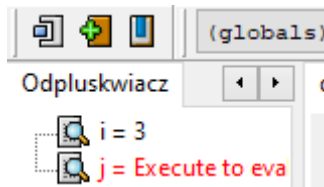
Dodaj zmienne które chcesz obserwować. Wykonuje się to klikając w przycisk dodaj zmienną.



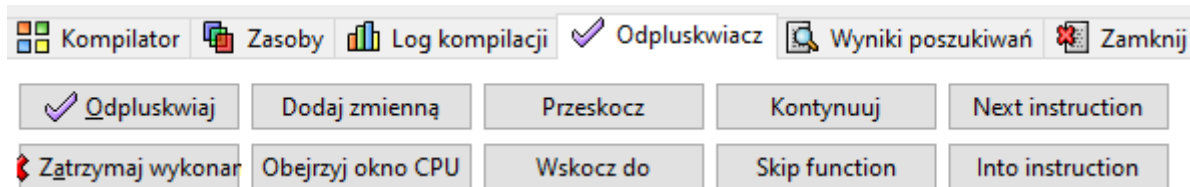
Pojawi się okienko w które należy wpisać nazwę zmiennej której wartość chcemy obserwować. W przykładzie jest to i.



Wybrane zmienne (i oraz j) pojawią się w lewej części formularza jak pokazano niżej.



Uruchamiamy debugowanie wciskając przycisk odpluskwiaj. W dole pojawi się okno debuggera.

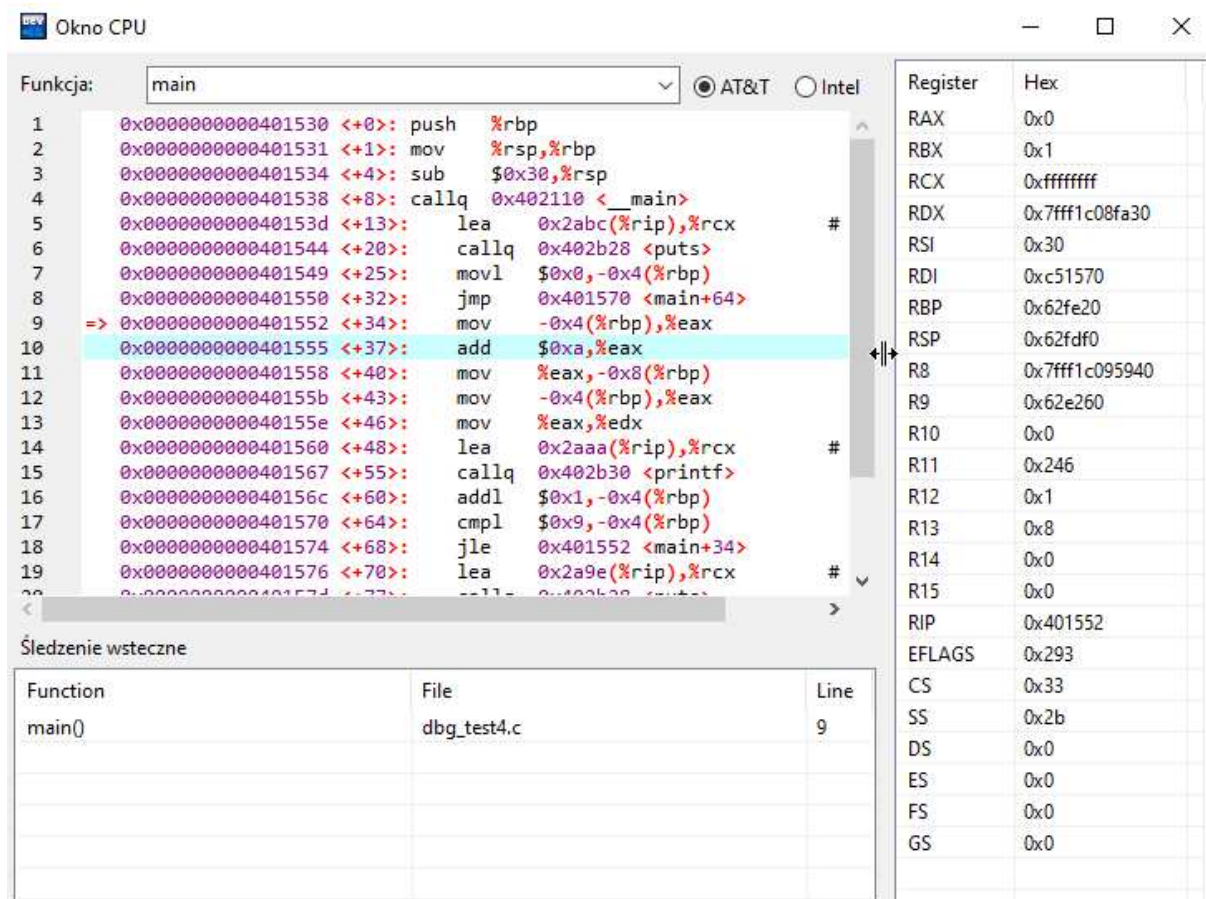


Pokazane tam przyciski służą do kontroli wykonywania programu.

- Odpluskwiaz - uruchomienie programu, zatrzyma się na najbliższym punkcie zatrzymania.
- Dodaj zmienną – dodajemy zmienną do obserwacji
- Przeskocz - wykonaj następną instrukcję ale nie wchodź do funkcji
- Kontynuuj - idź do następnego punktu zatrzymania
- Next instruction – wykonaj następną instrukcję
- Zatrzymaj wykonanie – zatrzymaj program
- Obejrzyj okno CPU- pokaż kod assemblera i rejestry procesora
- Wskocz do - wejdź do funkcji
- Skip function - nie wchodź do funkcji
- Into instruction – wykonaj instrukcję

W oknie zmiennych obserwujemy jaki jest stan zmiennych. W oknie kodu obserwujemy przebieg wykonania instrukcji.

Zakończenie debugowania następuje gdy program się zakończy lub gdy wciśniemy klawisz Zatrzymaj wykonanie.



## 1.7 Tworzone pliki

Środowisko tworzy w wybranym katalogu różne pliki których typy identyfikowane są przez rozszerzenia nazwy pliku. Używane rozszerzenia plików dane są w poniższej tabeli.

File	Extension	Description
<b>Project file</b>	<b>.dev</b>	Project configuration data
<b>Makefile</b>	<b>.win</b>	Required for the compilation process. Manages program dependencies and includes instructions for the linker
<b>Source code file</b>	<b>.c</b>	Source code
<b>Object file</b>	<b>.o</b>	Object code resulting from the compilation of the source code. Each .c has a corresponding .o after the compilation
<b>Executable file</b>	<b>.exe</b>	Executable application

Tabela 1-2 Pliki tworzone w ramach projektu

## 1.8 Zadania

### 1.8.1 Uruchomienie programu

Wprowadź z klawiatury poniższy program. Zapisz go jako plik `scanf.c`, następnie skompiluj i uruchom.

```
/ /Wczytywanie liczby int z konsoli
#include <stdio.h>
int main() {
    int number;
    printf( "Wprowadz liczbe: " );
    scanf( "%d", &number );
    printf( "Liczba jest: %d", number );
    return 0;
}
```

Przykład 0-1 Program `scanf.c`

### 1.8.2 Użycie edytora

Wprowadź z klawiatury poniższy program

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

int main(void) {
    int i,j,k;
    puts("Witamy w Lab WSH");
    for(k=1;k<100;k++) {
        printf("Nowa epoka %d\n",k);
        for(i=0;i<10;i++) {
            j=i+10;
            printf("Epoka %d krok  %d\n",k,i);
            sleep(1);
        }
    }
    printf("Koniec\n");
    return EXIT_SUCCESS;
}
```

Przykład 0-2 Program `test.c`

### 1.8.3 Kompilacja i uruchomienie

Skompiluj i uruchom powyższy program `test.c` jako pojedynczy plik

### 1.8.4 Kompilacja i uruchomienie - projekt

Skompiluj i uruchom powyższy program `test.c` jako projekt.

### 1.8.5 Debugowanie kodu

Uruchom debugger dla powyższego programu `test.c`. Dodaj jeszcze jedną zmienną:

```
int k;  
k=i*i;
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
int main(void) {  
    int i,j;  
    printf("Witamy w Lab WSH\n");  
    for(i=0;i<10;i++) {  
        j=i+10;  
        printf("Krok  %d\n",i);  
    }  
    printf("Koniec\n");  
    return EXIT_SUCCESS;  
}
```

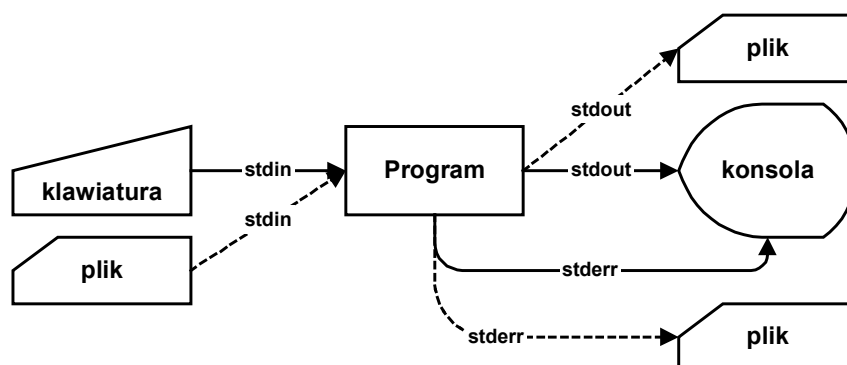
Przykład 0-3 Program testowy `dbg_test`

## 2. Podstawowe operacje wejścia wyjścia

### 2.1 Standardowe wejście i wyjście

W systemach klasy UNIX wiele zasobów traktowanych jest jako pliki. Dotyczy to także urządzeń wejścia wyjścia takich jak klawiatura i monitor które są plikami specjalnymi. Dla każdego wykonywanego programu system automatycznie otwiera trzy pliki:

- Standardowe wejście – domyślnie klawiatura
- Standardowe wyjście – domyślnie monitor
- Wyjście komunikatu o błędach – domyślnie monitor



Rys. 2-1 Standardowe wejście i wyjście programu

Opis	Wartość uchwytu	Przypisanie początkowe
Standardowe wejście	0	Klawiatura
Standardowe wyjście	1	Ekran
Wyjście komunikatów o błędach	2	Ekran

Tab. 2-1 Standardowe wejście / wyjście programu

Przyporządkowanie to można jednak zmienić, łącząc standardowe wejście, wyjście i wyjście błędów z plikami. Przykłady takiego przekierowania dla systemu Linux podaje poniższy przykład.

```

$./prog < plik_we
$./prog > plik_wy
$./prog > plik_wy > plik_err
$./prog < plik_we > plik_wy
$./prog1 | prog2

```

### 2.2 Funkcja putchar()

Funkcja `int putchar(int c)` wyprowadza pojedynczy znak `c` na standardowe wyjście. Znak `c` przekazany jako liczba `int`. Przekazywany jest więc kod tego znaku. Funkcja zwraca ten sam kod.

```
int putchar(int c)
```

Przykład działania funkcji `putchar()` pokazany został poniżej.

```
// program w C demontrujący putchar()
#include <stdio.h>

int main() {
    // Okresl zna ch
    char ch = 'G';
    // pisz znak ch na stdout
    putchar(ch);
    return (0);
}
```

Przykład 2-1 Działanie funkcji `putchar`

## 2.3 Funkcja `getchar()`

Funkcja `int getchar()` czyta jeden znak ze standardowego wejścia i go stamtąd usuwa. Wynik zwracany jest jako liczba `int`. Przykład programu który czyta jeden znak ze `stdin` i wyprowadza go na `stdout` podano poniżej.

```
// program w C demontsrujący getchar() u putchar()
#include <stdio.h>

int main() {
    int c;
    // czytaj znak ze stdin
    c = getchar();
    // pisz znak ch na stdout
    putchar(c);
    return (0);
}
```

Przykład 2-2 Użycie funkcji `getchar()` i `putchar()`

## 2.4 Funkcja `printf`

Funkcja `printf(...)` służy wyprowadzaniu napisów na standardowe wyjście (konsolę, monitor). Funkcja `printf(...)` pisze na standardowe wyjście zawartość kolejnych zmiennych zgodnie z łańcuchem formatującym. Funkcja zwraca liczbę wypisanych znaków.

```
int printf(łańcuch_formatujący, zmienna1, zmienna2, ..., zmiennaN);
```

Łańcuch formatujący zawiera znaki które mają być wprost wyprowadzone na konsolę i informacje w jaki sposób wyprowadzać kolejne zmienne. Taka informacja jest potrzebna gdyż np. trzeba określić na ilu pozycjach wyprowadzać zmienną. Są to tak zwane specyfikatory formatu. W łańcuchu formatującym może się pojawić:

- Zwykły tekst – jest on bez zmian wyprowadzony na `stdout`.
- Pola poprzedzone znakiem `%` - są to tak zwane specyfikatory formatu. Odnoszą się do kolejnych zmiennych.

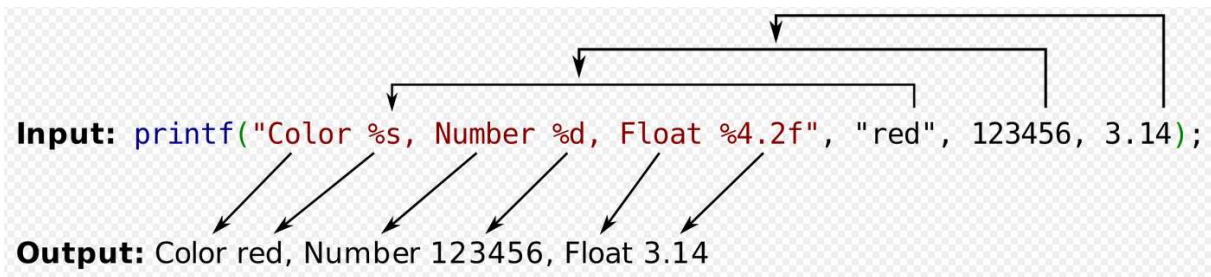
Ważniejsze specyfikatory formatu podaje poniższa tabela.



Spec.	Typ	Opis
<b>%d, %i</b>	<b>int</b>	Liczba całkowita w formacie dziesiętnym
<b>%o</b>	<b>int</b>	Liczba całkowita w formacie ósemkowym
<b>%x, %X</b>	<b>int, char</b>	Liczba całkowita w formacie szesnastkowym
<b>%u</b>	<b>int</b>	Liczba całkowita bez znaku
<b>%c</b>	<b>char</b>	Znak
<b>%s</b>	<b>char *</b>	Łańcuch (ang. <i>String</i> )
<b>%f</b>	<b>float, double</b>	Liczba zmiennoprzecinkowa. Format [-]m.dddddd
<b>%e, %E</b>	<b>float, double</b>	Liczba zmiennoprzecinkowa w postaci wykładniczej. Format [-]m.dddddde±xx, [-]m.ddddddeE±xx
<b>%g, %G</b>	<b>float, double</b>	Liczba wypisana w formacie %e, %E jeśli wykładnik jest mniejszy niż -4 albo większy lub równy precyzji, w przeciwnym wypadku liczba wypisana w formacie %f
<b>%p</b>	<b>void *</b>	Wskaźnik

Tab. 2-2 Ważniejsze specyfikatory formatu

Poniżej podany został przykład funkcji `printf()`.



Jeszcze inny przykład funkcji `printf()` dany jest poniżej.

```
// Ilustracja działania funkcji printf
#include <stdio.h>

main () {
    printf("napis: %s, liczba int: %d, liczba float: %f, znak:
%c\n","to jest napis",123,3.14,'c');
}
```

Przykład 2-3 Działanie funkcji `printf`

To co jest polu specyfikacji formatu czyli tekst:

**"napis: %s, liczba int: %d, liczba float: %f, znak: %c\n"**

wyprowadzany jest na stdout znak po znaku aż do napotkania znaku %. Będzie to tekst:

**napis:**

To co po nim czyli `%s` oznacza że w formacie **string** (patrz tabela wyżej) zostanie wyprowadzona pierwsza zmienna czyli tekst: **"to jest napis"**

Na konsoli pojawi się:

**napis: to jest napis**

Dalej wyprowadzony zostanie dalszy ciąg specyfikacji czyli:

**napis: to jest napis liczba int:**

Dalej przeczytany zostanie specyfikator `%d` czyli specyfikacja liczby **int** (decimal).

Według tej specyfikacji wyprowadzona będzie kolejna zmienna z listy czyli **123**. Na stdout będzie teraz napis:

**napis: to jest napis liczba int: 123**

Postępowanie będzie powtarzane, aż do wyczerpania specyfikacji i w końcu pojawi się napis jak niżej.

**napis: to jest napis, liczba int: 123, liczba float: 3.140000, znak: c**

Pomiędzy znakiem procenta, a znakiem przekształcenia mogą lecz nie muszą wystąpić w następującej kolejności:

- - (minus) – Wyrównanie argumentu do lewej strony jego pola
- Liczba określająca minimalny rozmiar pola

Aby określić ilość drukowanych cyfr do kodu formatującego można dodać kody długości pól:

- `%Ld` – dla liczby `int`, gdzie `L` jest zadaną liczbą cyfr na której liczba ma być zapisana.
- `%L.Pf`, `%L.Pg` – dla liczby `float` i `double`, gdzie `L` jest zadaną liczbą cyfr na której liczba ma być zapisana a `P` liczbą miejsc po przecinku.

Dla przykładu:

- `%4d` – liczba dziesiętna na czterech pozycjach
- `%10f` – liczba rzeczywista na 10 pozycjach
- `%10.2f` – liczba rzeczywista na 10 pozycjach, 2 cyfry po przecinku
- `%.3f` – liczba rzeczywista z dokładnością do 3 cyfr po przecinku

Pomiędzy specyfikatorami formatu mogą wystąpić dowolne znaki które będą po prostu wyprowadzone na konsolę. Wśród tych znaków mogą być znaki specjalne które nie mają bezpośredniej reprezentacji znakowej. Podane zostały one poniżej.

- `'\a'` - alarm (sygnał akustyczny terminala)
- `'\b'` - backspace (usuwa poprzedzający znak)
- `'\f'` - wysunięcie strony (np. w drukarce)
- `'\r'` - powrót kursora (karetki) do początku wiersza
- `'\n'` - znak nowego wiersza
- `'\"'` - cudzysłów
- `'\''` - apostrof
- `'\\'` - ukośnik wsteczny (backslash)
- `'\t'` - tabulacja pozioma
- `'\v'` - tabulacja pionowa

Znaki specjalne służą między innymi do sterowania wydrukiem. Przykład użycia różnych specyfikatorów formatu podaje poniższy przykład.

```
// Test funkcji printf
#include <stdio.h>
int main (void){
    int kk1 = 145E5;
    double pi = 3.14159265;
    double xx = 3.1E-7;
    char *tab = "Ala ma kota";
    long int el = 10L;
    unsigned int ui = 1E4;
    unsigned long ul = 10E5L;
    int max = 6;
    printf("|%d| \t|f| \t|s|\n", kk1, pi, tab);
    printf("|%15d| \t|%15f| \t|%15s|\n", kk1, pi, tab);
    printf("|%-15d| \t|%-15.8f| \t|%-15.8s|\n", kk1, pi, tab);
    printf("|%-15.10d| \t|%-15.2f| \t|%.10s|\n", kk1, pi, tab);
    printf("%d %i %o %x\n\n", kk1, kk1, kk1, kk1);
    printf("%ld %u %lu %c\n\n", el, ui, ul, tab[0]);
    return 0;
}
```

Przykład 2-4 Użycie różnych specyfikatorów formatu

Wyniki działania programu dane są poniżej.

```
|14500000|      |3.141593|      |Ala ma kota|
|      14500000|      |      3.141593|      |      Ala ma kota|
|14500000|      |3.14159265|      |Ala ma k|
|0014500000|      |3.14|      |Ala ma kot|
14500000 14500000 67240240 dd40a0

10 10000 1000000 A
```

Dokładniejszy opis funkcji printf podany został w dokumentacji:

<https://pl.wikibooks.org/wiki/C/printf>

## 2.5 Funkcja scanf

Równie ważną funkcją jest możliwość czytania danych z klawiatury. Służy do tego między innymi funkcja `scanf`.

```
int scanf(const char* format,...)
```

Łańcuch formatujący tworzy się tak jak dla funkcji `printf`. Następnie po łańcuchu formatującym dodaje się kolejne zmienne które mają być odczytywane, poprzedzone znakiem `&` (operator adresu). Przykładowo poniższa funkcja wczytuje z konsoli liczbę `int` i umieszcza ją w zmiennej `number`.

```
scanf( "%d", &number );
```

Cały przykład dany jest poniżej.

```
// Wczytywanie liczby int z konsoli
#include <stdio.h>

int main() {
    int number;
    printf( "Wprowadz liczbe: " );
    scanf( "%d", &number );
    printf( "Liczba jest: %d", number );
    getchar();
    return 0;
}
```

Przykład 2-5 Wczytywanie liczby int

### 3. Typy danych, stałe, zmienne, wyrażenia arytmetyczne i logiczne

#### 3.1 Pole i obwód koła

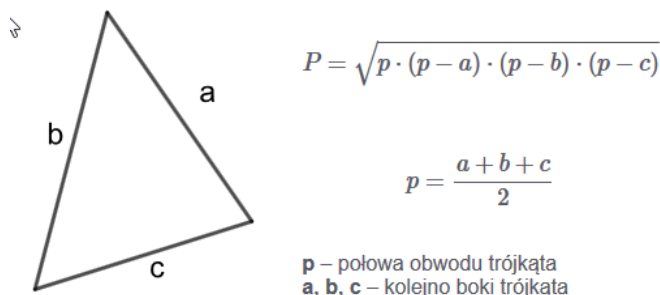
Napisz program obliczania pola i obwodu koła na podstawie wprowadzanego promienia. Promień wprowadzamy za pomocą funkcji `scanf` a wynik wyprowadzamy za pomocą funkcji `printf`.

**Pole koła** o promieniu  $r$  wynosi  $P = \pi r^2$ .

**Obwód koła** o promieniu  $r$  wynosi  $L = 2\pi r$ .

#### 3.2 Pole trójkąta

Napisz program obliczania pola trójkąta na podstawie wzoru Herona. Długości boków  $a, b, c$  są liczbami typu `float` i wprowadzamy je za pomocą funkcji `scanf`. Wynik wyprowadzamy za pomocą funkcji `printf`.



#### 3.3 Kalkulator

Napisz program pełniący funkcję kalkulatora działającego na liczbach `double`. Kalkulator ma wykonywać cztery działania `+`, `-`, `*`, `/`. Liczby wczytaj za pomocą `scanf` a działanie za pomocą funkcji `getchar()`.

#### 3.4 Kody ascii

Napisz program który wprowadza z konsoli znak `c` i wypisuje jego kod ASCII.

#### 3.5 Reprezentacja liczb jako `dec`, `hex`.

Napisz program który wprowadza z konsoli liczbę `int` i wypisuje jej reprezentację dziesiętną, ósemkową i szesnastkową.

#### 3.6 Operator `sizeof()`

Napisz program wypisujący na konsoli liczbę bajtów zajmowaną przez wszystkie typy proste języka C. Użyj funkcji `sizeof()`.

#### 3.7 Funkcje matematyczne

Napisz program obliczający wartość podanego niżej wyrażenia.

$$y = \frac{\frac{1}{2} \cdot \sin^2(0.45) + 2 \cdot \operatorname{tg}(\sqrt{2})}{\log_{10}(14) + 2 \cdot e^4}$$

### 3.8 Porównania 2

Dla jakiej wartości T program napisze woda?

```
if (T < 0)
    printf("lod\n");
else if (T < 100)
    printf("woda\n");
else    printf("para\n");
```

## 4. Binarna reprezentacja informacji

### 4.1 Rodzaje informacji reprezentowane w komputerach

W komputerach złożone informacje są reprezentowane przez ciągi bitów mających wartości tylko 0 lub 1. Najczęściej występującymi typami danych są:

- Liczby całkowite
- Liczby rzeczywiste
- Znaki z których zbudowane są teksty

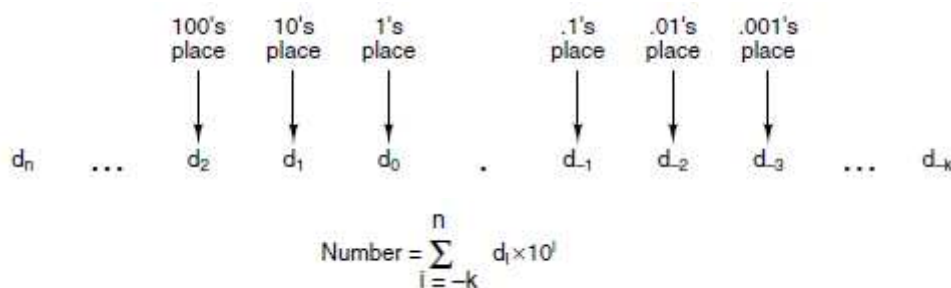
Z tych prostych typów danych tworzone są typy bardziej złożone jak tablice, teksty, obrazy, dane multimedialne.

### 4.2 Pozycyjne systemy liczbowe

Powszechnie używane liczby dziesiętne składają się z ciągu cyfr dziesiętnych (0,1,...,9) i być może przecinka. W zależności od pozycji danej cyfry w ciągu, oznacza ona wielokrotność potęgi pewnej liczby uznawanej za bazę danego systemu. Powszechnie używa się systemu dziesiętnego, w którym za bazę przyjmuje się liczbę dziesięć. Tak zapis 46 532 ma następujące znaczenie.

$$4 \times 10^4 + 6 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 = 46\,532.$$

Ogólną zasadę zapisu pozycyjnego o podstawie 10 liczby pokazuje poniższy rysunek.



Rys. 4-1 Ogólne przedstawienie liczby dziesiętnej

Liczba (Number) jest sumą iloczynów kolejnych potęg liczby 10 i cyfr od 0 do 9. Dziesięć jako podstawa systemu liczenia jest wygodna dla ludzi ale niekoniecznie dla maszyn. W większości komputerów za podstawę liczenia przyjmuje się liczbę 2 i taki system nazywamy dwójkowym lub binarnym. Używane jest także przedstawienie liczb w postaci ósemkowej i dwójkowej.

Zbiór podstawowych cech dowolnego systemu pozycyjnego o podstawie  $k$  jest następujący:

1. System pozycyjny charakteryzuje liczba zwana podstawą systemu pozycyjnego.
2. Do zapisu liczby służą cyfry.
3. Cyfr jest zawsze tyle, ile wynosi podstawa systemu:  $0, 1, 2, \dots, k-1$ .
4. Cyfry ustawiamy na kolejnych pozycjach.
5. Pozycje numerujemy od 0 poczynając od strony prawej zapisu.
6. Każda pozycja posiada swoją wagę.
7. Waga jest równa podstawie systemu podniesionej do potęgi o wartości numeru



pozycji.

8. Cyfry określają ile razy waga danej pozycji uczestniczy w wartości liczby.

9. Wartość liczby obliczamy sumując iloczyny cyfr przez wagi ich pozycji.

System liczbowy o podstawie  $k$  wymaga  $k$  różnych symboli do reprezentowania cyfr  $0, 1, 2, \dots, k-1$ .

- Liczby dziesiętne (Decimal) tworzone są z dziesięciu różnych cyfr  $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ .
- Liczby dwójkowe (Binary) zapisywane są za pomocą cyfr  $0$  i  $1$
- Liczby ósemkowe (Octal) za pomocą ośmiu  $0, 1, 2, 3, 4, 5, 6, 7$ ,
- Liczby szesnastkowe (Hexadecimal) za pomocą cyfr  $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$  i liter  $A, B, C, D, E, F$ .

Zamiast określenia cyfra binarna (ang. *Binary Digit*) używa się zamiennie słowa bit. Na poniższym rysunku pokazano przedstawienie tej samej liczby 2001 jako liczby binarnej, ósemkowej, dziesiętnej i szesnastkowej.

Binary	1	1	1	1	1	0	1	0	0	0	1												
	$1 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$																						
	1024	+	512	+	256	+	128	+	64	+	0	+	16	+	0	+	0	+	0	+	0	+	1
Octal	3	7	2	1																			
	$3 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$																						
	1536	+	448	+	16	+	1																
Decimal	2	0	0	1																			
	$2 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$																						
	2000	+	0	+	0	+	1																
Hexadecimal	7	D	1																			.	
	$7 \times 16^2 + 13 \times 16^1 + 1 \times 16^0$																						
	1792	+	208	+	1																		

Rys. 4-2 Przedstawienie liczby 2001 w różnych systemach liczenia

Liczby z zakresów  $0$  do  $15$  w systemach binarnym, dziesiętnym i szesnastkowym przedstawia poniższa tabela.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

### 4.3 Przekształcanie liczb między systemami pozycyjnymi

Przekształcenia (konwersje) pomiędzy liczbami dwójkowymi, ósemkowymi i szesnastkowymi wykonuje się bardzo łatwo. By przekształcić liczbę binarną na system ósemkowy grupujemy poszczególne bity (zaczynając od najmłodszego) w grupy trój bitowe. Następnie zastępujemy każdą grupę przez równoważną cyfrę ósemkową zgodnie z poniższą tabelą

Decimal	Binary	Octal	Hex
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Tabela 4-1 Liczby od 0 do 15 w przedstawieniu dziesiętnym, binarnym, ósemkowym i szesnastkowym

**Uwaga!**

Maksymalna liczba binarna jaka daje się przedstawić na  $n$  bitach to  $2^n - 1$ . Liczba taka posiada same jedynki na wszystkich pozycjach.

Podobnie dokonuje się konwersji pomiędzy zapisem dwójkowym a szesnastkowym z tą różnicą że bity grupuje się po cztery. Poniższe przykłady pokazują konwersję liczb binarnych na postać ósemkową i szesnastkową.

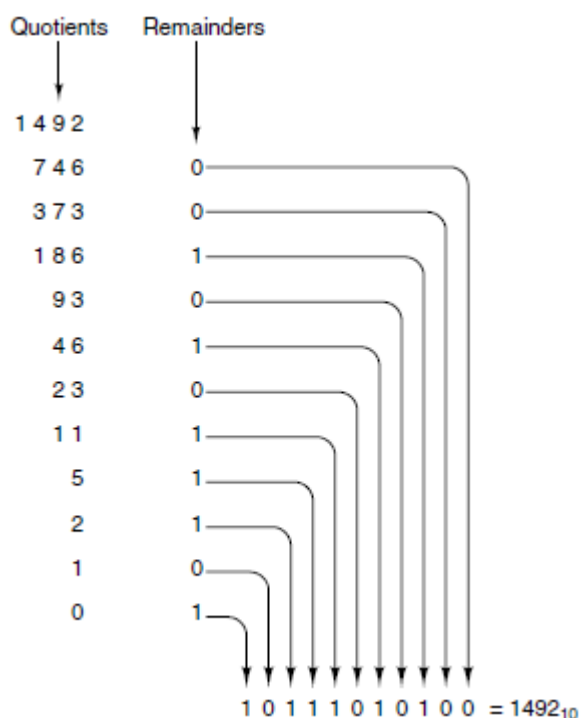
Hexadecimal	1	9	4	8	.	B	6		
Binary	0001	1001	1010	1000	.	1011	101100		
Octal	1	4	5	1	0	.	5	5	4

Rys. 4-3 Przykład 1 konwersji liczby binarnej na ósemkową i szesnastkową

Hexadecimal	7	B	A	3	.	B	C	4		
Binary	0111	1011	1010	0011	.	1011	1100	0100		
Octal	7	5	6	4	3	.	5	7	0	4

Rys. 4-4 Przykład 2 konwersji liczby binarnej na ósemkową i szesnastkową

Konwersji liczby dziesiętnej na dwójkową możemy dokonać dzieląc sukcesywnie liczbę przez 2 (połowienie liczby) i zapisując otrzymywane reszty. Kolejne reszty są coraz bardziej znaczącymi cyframi zapisu dwójkowego liczby. Pokazuje to poniższy przykład w którym zamieniamy liczbę 1492 na liczbę binarną (Quotient – wynik dzielenia, Remainder – reszta).



Rys. 4-5 Zamiana liczby 1492 na postać binarną przez kolejne dzielenia przez 2

Konwersję liczby z postaci dwójkowej na dziesiętną możemy łatwo wykonać sumując kolejne potęgi dwójki. Załóżmy że chcemy przekształcić do postaci dziesiętnej liczbę binarną 10101101. Przedstawiamy ją jako sumę kolejnych potęg liczby 2.

$$1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

Po obliczeniu potęg dwójki otrzymujemy:

$$1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$$

$$1 \cdot 128 + 1 \cdot 32 + 1 \cdot 8 + 1 \cdot 4 + 1 \cdot 1$$

$$128 + 32 + 8 + 4 + 1$$

co daje wynik 173.

#### 4.4 Kodowanie tekstu

Oprócz cyfr, komputer przetwarza także teksty. Teksty składają się z liter, cyfr, znaków interpunkcyjnych, graficznych. Aby komputer mógł zapisywać i przetwarzać teksty, ich składowe muszą być jakoś zakodowane. W informatyce powszechnie stosuje się kody ASCII. **ASCII** (ang. American Standard Code for Information Interchange) jest to siedmiobitowy system kodowania znaków, używany we współczesnych komputerach oraz sieciach komputerowych, a także innych urządzeniach wyposażonych w mikroprocesor. Przyporządkowuje liczbom z zakresu 0–127 litery alfabetu łacińskiego języka angielskiego, cyfry, znaki przestankowe i inne symbole oraz polecenia sterujące. Większość współczesnych systemów

kodowania znaków jest rozszerzeniem standardu ASCII. Standard ASCII został stworzony na podstawie kodu telegraficznego. Prace nad nim rozpoczęły się w 1960 roku, w 1963 roku, została udostępniona pierwsza wersja standardu ASCII. W porównaniu do wcześniejszych systemów kodowania znaków, ten zestaw znaków był wygodny w użyciu do sortowania alfabetycznego tekstów, zmiany wielkości znaków, a także wspierał urządzenia inne niż dalekopisy. Od czasu wprowadzenia na rynek, ASCII został czterokrotnie zaktualizowany





Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	\$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

Tabela 4-2 Kody ASCII

Ponieważ kod ASCII jest 7-bitowy, a większość komputerów operuje na 8-bitowych bajtach, dodatkowy bit można wykorzystać na powiększenie zbioru kodowanych znaków do 256 symboli. Powstało wiele różnych rozszerzeń ASCII, ponad 220 stron kodowych DOS i Windows, wykorzystujących ósmy bit (np. norma ISO 8859, rozszerzenia firm IBM lub Microsoft) nazywanych stronami kodowymi. Również kodowanie UTF-8 można uważać za rozszerzenie ASCII, tutaj jednak dodatkowe znaki są kodowane na 2 i więcej bajtach. Formalnie, mianem *rozszerzeń ASCII* można nazwać jedynie te standardy, które zachowują układ pierwszych 128 znaków i dodają nowe *na końcu tabeli*.

## 4.5 Ćwiczenia

### 4.5.1 Zamiana liczb dziesiętnych na binarne, ósemkowe i szesnastkowe

Napisz w języku C program zamieniający liczby dziesiętne na ich reprezentację binarną, ósemkową i szesnastkową. Program ma nazywać się `konwert2bin` a liczba do konwersji ma być podana jako argument

```
konwert2bin 99
```

```
bin: 1100011 oct:0143 hex: 0x63
```

lub wczytana funkcją `scanf`. Początek programu dany jest poniżej.

```
#include <stdio.h>

int main (int argc, char *argv[])
{   int liczba;
    int wynik[32]; // Tablica na wynik
    printf ("Podaj liczbe dziesietne:");
    scanf ("%d", &liczba);
    // tu ma być zamiana
    return 0;
}
```

#### 4.5.2 Zamiana liczb binarnych na dziesiętne, ósemkowe i szesnastkowe

Napisz w języku C program zamieniający liczby binarne ich reprezentację dziesiętną, ósemkową i szesnastkową. Program ma nazywać się konwert2dec a liczba do konwersji ma być podana jako argument. Np.

**konwert2dec 01100011**

dec: 99 oct:0143 hex: 0x63

lub wczytana za pomocą funkcji bibliotecznej scanf. Zwróć uwagę że wprowadzaną liczbę binarną należy potraktować jako tablicę znaków a nie liczbę całkowitą.

```
#include <stdio.h>

int main (int argc, char *argv[])
{   int wynik;
    int liczba[32]; // Tablica na wynik
    printf ("Podaj liczbe binarna:");
    scanf ("%s", &liczba);
    // tu ma być zamiana
    return 0;
}
```

### 4.5.3 Kodowanie znaków ASCII

Napisz program który wypisuje kody ASCII znaków o numerach od 0 do 127 oraz ich kody dziesiętne i szesnastkowe. Jeżeli znak jest znakiem sterującym i nie posiada reprezentacji znakowej zastąp go odpowiednim symbolem z poniższej tabeli

Opis	Opis ang.	Symbol	Numer dziesiętny
znak pusty	null	NUL	0
awaryjne przerwanie działania programu	end of text	ETX	3
koniec transmisji danych	end of transmission	EOT	4
usunięcie poprzedzającego znaku	backspace	BSP	8
znak tabulacji	horizontal tab	HT	9
koniec wiersza	line feed	LF	10
koniec strony	form feed	FF	12
powrót do początku wiersza	carriage return	CR	13
koniec danych w pliku	end of file	EOF	26
znak sterujący: początek rozkazu	escape	ESC	27

Tabela 4-3 Znaki sterujące

## 5. Instrukcje sterujące i instrukcje iteracji

### 5.1 Ciąg Fibonacciego

Napisz program obliczający kolejne liczby ciągu Fibonacciego. Ciąg został omówiony w roku 1202 przez Leonarda z Pizy, zwanego Fibonaccim, w dziele "Liber abaci" jako rozwiązanie zadania o rozmnażaniu się królików. Pierwszy wyraz ciągu jest równy 0, drugi jest równy 1, każdy następny jest sumą dwóch poprzednich.

Formalnie:

$$F_n := \begin{cases} 0 & \text{dla } n = 0, \\ 1 & \text{dla } n = 1, \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

Liczbę kroków wczytaj za pomocą scanf.

Pierwsze dwadzieścia wyrazów ciągu Fibonacciego to:



$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$	$F_{11}$	$F_{12}$	$F_{13}$	$F_{14}$	$F_{15}$	$F_{16}$	$F_{17}$	$F_{18}$	$F_{19}$
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181

### 5.2 Pętla i warunek

Napisz program który oblicza w pętli ile liczb z zakresu 1 do 100 jest większych od 3. Wykorzystaj instrukcje for(...) oraz if(...).

### 5.3 Pętla i wartość średnia

Napisz program który liczy wartość średnią z liczb 1, 4, 9, ..., 81, 100. Są to kwadraty kolejnych liczb 1,2,...,10.

### 5.4 Ile jest liczb parzystych w zakresie?

Napisz program który zlicza liczby parzyste i nieparzyste w zakresie od 1 do 10. Wykorzystaj operator % który podaje resztę z dzielenia.

### 5.5 Liczenie jedynek w reprezentacji binarnej.

Napisz program który wprowadza z konsoli znak c i wypisuje liczbę bitów ustawionych na 1. Potraktuj znak c jako zmienną typu `unsigned char`.



## 5.6 Trójkąt

Napisz program wypisujący na konsoli podany niżej trójkąt.

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
```

## 5.7 Zliczanie znaków

Napisz program zliczania w tekście wprowadzanym z konsoli małych liter, dużych liter, cyfr, spacji i znaków sterujących. Wykorzystaj poniższy kod i funkcje z pliku nagłówkowego <ctype.h>.

```
#include<stdio.h>

int main() {
    int c;
    while((c = getchar()) != EOF)
    {
        printf("%c",c);
    }
    return 0;
}
```

**EOF** (ang. *End Of File*) jest kodem końca wprowadzania.

## 5.8 Pętla while

Poniżej dany jest fragment programu wypisującego kolejne liczby przy użyciu instrukcji for.

```
for(i = 0; i < 10; i = i + 1)
    printf("i is %d\n", i);
```

Napisz program robiący to samo przy użyciu instrukcji

```
while (warunek) { ...};
do { ... } while(warunek);
```

## 5.9 Tablica sum liczb

Używając dwóch zagnieżdżonych pętli for napisz program który drukuje tablicę taką jak poniżej. Element na przecięciu kolumny  $i$  ( $i=1,2,\dots,10$ ) oraz wiersza  $j$  ( $j=1,2,\dots,10$ ) jest sumą liczb  $i$  oraz  $j$  czyli wynosi  $i + j$ .

	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11	12	13
4	5	6	7	8	9	10	11	12	13	14
5	6	7	8	9	10	11	12	13	14	15
6	7	8	9	10	11	12	13	14	15	16
7	8	9	10	11	12	13	14	15	16	17
8	9	10	11	12	13	14	15	16	17	18
9	10	11	12	13	14	15	16	17	18	19
10	11	12	13	14	15	16	17	18	19	20

## 5.10 Tabliczka mnożenia

Używając dwóch zagnieżdżonych pętli for napisz program który drukuje tabliczkę mnożenia taką jak poniżej. Element na przecięciu kolumny  $i$  ( $i=1,2,\dots,10$ ) oraz wiersza  $j$  ( $j=1,2,\dots,10$ ) jest iloczynem liczb  $i$  oraz  $j$  czyli wynosi  $i * j$ .

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

## 5.11 Szyfr Cezara

Szyfr Cezara jest prostym szyfrem podstawieniowym. Przy szyfrowaniu polega on zastąpieniu kodu danego znaku, kodem otrzymanym z kodu znaku jawnego do którego dodano klucz **key**. Operacja deszyfracji polega na odjęciu od kodu znaku zaszyfrowanego klucza **key**. Należy zabezpieczyć się przed wyjściem poza zakres kodowanych znaków. Znaczy to że jak wyjdziemy poza zakres kodów, należy dokonać korekty. Przykładowo przy przesunięciu o 1 kod litery z przechodzi w a

Przykład kodowania znaków podano poniżej.

Tekst jawny	a	b	c		y	z
Tekst zaszyfrowany	b	c	d		z	a

Przykład 5-1 Szyfr Cezara przy kluczu 1

W programie należy przewidzieć wystąpienie dużych liter, małych liter, spacji i cyfr.

W programie można użyć danego niżej przykładu.

```
#include <stdio.h>

int main() {
    char c;
    while(1)
    {
        c = getche();
        printf(" znak: %c kod: %d\n",c,c);
        if(c == 27) break;
    }
    return 0;
}
```

Kod klawisza Esc to 27 i powoduje on zakończenie programu. Napisz program szyfrowania i deszyfrowania tekstu.

## 5.12 Sklejanie tekstów

Poniżej dany jest program wczytujący dwa napisy: tekst1 i tekst2. Uzupełnij program tak aby tekst3 był sklejeniem napisów tekst1 i tekst2. Pamiętaj że napis w języku C jest trzymany w tablicy i kończy się znacznikiem 0x00.

```
// Dodawanie tekstow
#include <stdio.h>
#define TSIZE 40

int main() {
    char text1[TSIZE];
    char text2[TSIZE];
    char text3[2*TSIZE];
    int i;
    int len1, len2;
    printf("podaj tekst1: ");
    gets(text1);
    printf("%s\n", text1);
    printf("podaj tekst2: ");
    gets(text2);
    printf("%s", text2);
    // Tu ma być dodawanie tekstow
    return 0;
}
```

Przykład 5-1 Program sklej\_teksty.c

Przykład:

tekst1: "Hej do"

tekst2: "pracy rodacy"

tekst3: "Hej do pracy rodacy"

## 6. Tablice i Funkcje

### 6.1 Znajdowanie maksimum w tablicy liczb

Napisz program tworzący w pamięci komputera 10-elementową tablicę liczb typu `double` i zapełnij ją (z użyciem pętli `for`) liczbami losowymi z przedziału `[0, 1]` które generowane są funkcją `rand()`. Następnie (z użyciem drugiej pętli `for`) wypisz zawartość tablicy. Na koniec uzupełnij kod programu tak, aby program znajdował i wypisywał maksymalny element i jego indeks. Sposób uzyskania liczb losowych podany jest poniżej.

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

int main() {
    const int n = 10;
    double tab[n];
    for (int i = 0; i < n; ++i) {
        tab[i] = (rand() / (RAND_MAX + 1.0));
    }
    // ...
    return 0;
}
```

### 6.2 Znajdowanie maksimum w tablicy liczb – funkcja

Przekształć program znajdowania maksimum w tablicy w taki sposób aby szukanie wykonywała funkcja:

```
int maximum(double tab[], int size)
```

Przeszukiwana jest tablica `tab` o wymiarze `size`, a funkcja zwraca maksymalny element.

### 6.3 Sumowanie elementów tablicy - funkcja

Niech w programie zadeklarowana będzie tablica jednowymiarowa jako zmienna globalna.

```
#include <stdio.h>
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int suma_tab(int tab[], int size);
int main(void) {
    ...
}
```

Napisz funkcję `suma_tab(...)` sumującą elementy dowolnej tablicy `int tab[]` o wymiarze `size`. Wywołaj tę funkcję i wyświetl wynik.

## 6.4 Sortowanie bąbelkowe

Korzystając z danego poprzednio programu tworzącego losową tablicę typu `double`, napisz program sortowanie bąbelkowego. Algorytm polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Niech  $n$ -elementowa tablica  $a$  zawiera liczby `int` które należy posortować powiedzmy rosnąco. Sortowanie bąbelkowe polega na zastosowaniu następującego algorytmu:

### Krok 1

Zacznij od indeksu  $i=0$  i porównaj  $a[0]$  i  $a[1]$ . Gdy  $a[0] > a[1]$  zamień je miejscami. Teraz porównaj  $a[1]$  z  $a[2]$ , gdy  $a[1] > a[2]$  zamień je miejscami. Powtarzaj ten proces aż do końca tablicy. W wyniku tego postępowania największy element znajdzie się na końcu tablicy. Postępowanie to nazywa się przebiegiem, przetwarzamy elementy  $[0, n-1]$ .

### Krok 2

Powtarzamy powyższe postępowanie ale przetwarzamy elementy  $[0, n-2]$  gdyż w pozycji  $n-1$  jest największy element. Po zakończeniu tego kroku dwa ostatnie elementy  $a[n-2]$  i  $a[n-1]$  są na właściwych pozycjach.

### Krok $n-1$

Powtarzamy te postępowanie  $n-1$  razy. Tablica zostanie posortowana.

4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5
3	4	2	1	5
3	2	4	1	5
3	2	1	4	5
2	3	1	4	5
2	1	3	4	5
1	2	3	4	5

Zmodyfikuj program tak, aby dla każdej iteracji pętli `do{ ... } while(...)` wyświetlał listę par indeksów elementów ulegających zamianie.

## 6.5 Sortowanie bąbelkowe – funkcja

Przekształć poprzedni program w taki sposób aby sortowanie wykonywane było przez funkcję:

```
int bubble(int tab[], int size);
```

Funkcja ma zwracać liczbę przestawień.

## 7. Łańcuchy i napisy

### 7.1 Reprezentacja łańcuchów w języku C

Do przechowania łańcucha znaków służy tablica znakowa. W kolejnych pozycjach tej tablicy są pamiętane kolejne znaki. Ostatnim znakiem musi być znak końca, czyli znak *ASCII* o numerze 0. Poniżej pokazano przykład deklaracji tablicy znakowej, która może przechowywać łańcuch o maksymalnej długości 9 znaków. Tablica ma 10 elementów, bo musi jeszcze pomieścić znak końca.

```
char s[10];
```

Deklarując taką tablicę, można do niej od razu wpisać jakiś łańcuch, na przykład

```
char s[10]={'p','r','o','g','r','a','m','\0'};
```

Nie wolno wtedy pominąć końcowego znaku '\0'. Prostszy sposób polega na przypisaniu do deklarowanej tablicy odpowiedniej stałej łańcuchowej:

```
char s[10]= "program";
```

Końcowy znak '\0' wpisze się teraz automatycznie.

<code>char s[10];</code>	
<code>s[0]</code>	<code>p</code>
<code>s[1]</code>	<code>r</code>
<code>s[2]</code>	<code>o</code>
<code>s[3]</code>	<code>g</code>
<code>s[4]</code>	<code>r</code>
<code>s[5]</code>	<code>a</code>
<code>s[6]</code>	<code>m</code>
<code>s[7]</code>	<code>\0</code>
<code>s[8]</code>	
<code>s[9]</code>	

Rys. 7-1 Przechowywanie napisów w postaci tablicy znaków

Łańcuch można wyprowadzić na ekran za pomocą instrukcji `printf(...)`.

```
printf("Napis to: %s ",s);
```

```
Napis to: program
```

Łańcuch można odczytać z klawiatury za pomocą instrukcji `scanf` z kodem formatującym `"%s"`. Przy tym należy pamiętać że:

1. Przed nazwą tablicy, w której jest zapamiętywany łańcuch, nie ma operatora adresu, ponieważ nazwa tablicy jest jej adresem.

2. Jeżeli czytamy łańcuch zawierający spację, to zostanie zapamiętana tylko ta jego część, która poprzedza spację, ponieważ spacja zostanie potraktowana jako separator.

Przykład użycia funkcji `printf` i `scanf` do wczytania danej, która jest nazwisko osoby:

```
char name[40];
printf("Podaj nazwisko: ");
scanf("%s", name)
```

Funkcje biblioteczne `puts` i `gets` służą do wypisywania i wczytywania łańcuchów. Funkcje `puts` i `gets` nie wymagają kodów formatujących.

Funkcja `puts` wyprowadza na ekran łańcuch będący jej argumentem, następnie przesuwa kursor na początek następnej linii ekranu. Argument funkcji `puts` może być jawna stała łańcuchowa, lub nazwa tablicy, która przechowuje łańcuch.

Funkcja `gets` odczytuje łańcuch z klawiatury i zapamiętuje go w tablicy, której nazwa jest jej argumentem. Funkcja `gets` odczytuje bez problemu także łańcuchy, zawierające spację.

```
char name[40];
puts("Podaj nazwisko: ");
gets(name);
```

Do elementów łańcucha odwołujemy się tak, jak do elementów tablicy (przez indeks lub wskaźnik). Długość łańcucha pamiętanego w tablicy może być mniejsza, niż rozmiar tablicy. Do określenia długości łańcucha aktualnie pamiętanego w tablicy służy funkcja biblioteczna `strlen`, opisana w zbiorze nagłówkowym `<string.h>`.

```
#include <stdio.h>
#include <string.h>
int main () {
    char napis[20] = "To jest lancuch";
    int len, i;
    len = strlen(napis);
    char *pocz;
    pocz = napis;
    for(i = 0; i < len; i++) {
        printf("%s\n", pocz);
        pocz++;
    }
    return(0);
}
```

Przykład 7-1 Program `string-demo.c`



```

To jest lancuch
o jest lancuch
 jest lancuch
 jest lancuch
est lancuch
st lancuch
t lancuch
 lancuch
lancuch
ancuch
ncuch
cuch
uch
ch
h

```

Przykład poniżej pokazuje dwie różne realizacje funkcji, która zlicza liczbę wystąpień znaku *z* w łańcuchu *s*.

```

int ile_z1(char z, char s[]) {
    int j, ile=0;
    for(j=0;j<strlen(s);j++) {
        if(s[j]==z) ile++;
    }
    return ile;
}

```

#### Przykład 7-2 Wykorzystanie tablicy

Funkcja `ile_z1` stosuje pętlę `for` ze zmienną licznikową *j*, która służy jako indeks kolejnych znaków łańcucha. Liczba kroków jest wyznaczona przez funkcję `strlen`.

```

int ile_z2(char z, char *s) {
    int j=0, ile=0;
    while(*(s+j)) {
        if(*(s+j)==z) ile++;
        j++;
    }
    return ile;
}

```

#### Przykład 7-3 Wykorzystanie wskaźnika

Funkcja `ile_z2` wykorzystuje pętlę `while` i wskaźnikowe odwołania do elementów. Warunek działania pętli to `*(s+j)`. Gdy *j*-ty znak jest znakiem końca o wartości liczbowej zero, instrukcja `while` przestaje działać i stan licznika *ile* zostaje przekazany przez `return`.

Jeżeli łańcuch *s* ma być parametrem funkcji to deklarujemy go jako `char *s`.

W języku C++ występuje wiele funkcji bibliotecznych, zdefiniowanych w zbiorze nagłówkowym `<string.h>` które są przeznaczone do operacji na łańcuchach znakowych. Zestawiono je w poniższej tabeli.

Funkcja	Działanie funkcji
<b>char</b> <b>*p=strcat(s1,s2)</b>	Skleja łańcuch s1 z łańcuchem s2. Zwraca wskaźnik na łańcuch wynikowy
<b>strcpy(s,s1)</b>	Kopiuje łańcuch s1 do łańcucha s. Argument s1 może być także stała łańcuchowa.
<b>strncpy(s,s1,n)</b>	Kopiuje n początkowych znaków łańcucha s1 do łańcucha s.
<b>int strlen(s)</b>	Zwraca aktualna długość łańcucha s, bez uwzględnienia znaku końca.
<b>int strcmp(s1,s2)</b>	Porównuje łańcuchy s1 i s2. Zwraca wartość: <ul style="list-style-type: none"> <li>• = 0, gdy łańcuchy są jednakowe,</li> <li>• &gt; 0, gdy s1 jest alfabetycznie większy od s2,</li> <li>• &lt; 0, gdy s2 jest alfabetycznie mniejszy od s1.</li> </ul>
<b>int stricmp(s1,s2)</b>	Porównuje łańcuchy s1 i s2 bez rozróżniania dużych i małych liter.
<b>int strncmp(s1,s2,n)</b> ;	Porównuje tylko n początkowych znaków łańcuchów s1 i s2.
<b>char</b> <b>*p=strstr(s,s1)</b> ;	Zwraca wskaźnik na początek pierwszego wystąpienia podłańcucha s1 w łańcuchu s. Gdy s nie zawiera s1, zwraca wartość NULL.

Tabela 7-1 Funkcje standardowe operujące na napisach

Tablice łańcuchów

Łańcuch jest tablicą, więc tablica t, która zawiera M łańcuchów N Znakowych, jest tablica dwuwymiarową:

```
char t[M][N];
```

Tablice taka można wstępnie zapełnić przy deklarowaniu, wpisując wartości poszczególnych wierszy w cudzysłowach jako stałe łańcuchowe.

```
char t[4][8]= {"Tomek", "Jarek", "Zbyszek", "Agata"};
```

Tablica wskaźników na łańcuchy znakowe

Użycie znakowej tablicy dwuwymiarowej nie jest jedynym sposobem pamiętania łańcuchów znakowych. Można posłużyć się równie\_ tablicą wskaźników na znaki, inicjując ją tak, jak poniżej:

```
char *tab[6]=
{"ADAM", "TADEUSZ", "EWA", "GRZEGORZ", "IGNACY", "ZENON"};
```

Po takiej deklaracji, kompilator umieści inicjowane łańcuchy w pamięci bezpośrednio po sobie, niezależnie od ich długości. Po znaku końca pierwszego łańcucha znajdzie się początkowy znak drugiego łańcucha itd. Dzięki temu nie ma niewykorzystanych miejsc i znacznie zmniejsza się rozmiar pamięci, potrzebnej do zapamiętania tekstu. Jednocześnie z lokowaniem łańcuchów w pamięci, ich adresy są umieszczane w

kolejnych pozycjach tablicy `tab`. Dlatego za pomocą elementów tej tablicy można odwoływać się do kolejnych łańcuchów w pamięci.

tab[0]	Adr0		Adr0	A	D	A	M	\0											
tab[1]	Adr1		Adr1	T	A	D	E	U	S	Z	\0								
tab[2]	Adr2		Adr2	E	W	A	\0												
tab[3]	Adr3		Adr3	G	R	Z	E	G	O	R	Z	\0							
tab[4]	Adr4		Adr4	I	G	N	A	C	Y	\0									
tab[5]	Adr5		Adr5	Z	E	N	O	N	\0										

Rys. 7-2 Tablica łańcuchów deklarowana jako `char *tab[6]`

tab[0]	A	D	A	M	\0														
tab[1]	T	A	D	E	U	S	Z	\0											
tab[2]	E	W	A	\0															
tab[3]	G	R	Z	E	G	O	R	Z	\0										
tab[4]	I	G	N	A	C	Y	\0												
tab[5]	Z	E	N	O	N	\0													

Rys. 7-3 Tablica łańcuchów deklarowana jako `char tab[6][12]`

## 7.2 Obliczanie długości napisu

Napisz funkcję: `int dlug_napis(char text[])` która zwraca długość napisu `text` będącego jej argumentem. Nie używaj funkcji bibliotecznych np. `strlen()` ale napisz własną jej wersję.

## 7.3 Zliczanie słów w łańcuchu tekstowym

Zadanie zliczenia słów ( ang. *words counting* ) sprowadza się do przeglądnięcia tablicy zawierającej znaki. Na początku pracy algorytmu ustawiamy znacznik słów `t` na `false`. Wartość `true` tego znacznika oznacza przetwarzanie znaków słowa. Licznik słów `ls` zerujemy. Teraz w pętli przeglądamy kolejne znaki łańcucha `s`. Jeśli napotkanym znakiem jest znak litery lub cyfry, to sprawdzamy stan znacznika `t`. Jeśli jest on ustawiony na `false`, to znaczy, iż napotkaliśmy w tekście początek słowa. W takim przypadku ustawiamy `t` na `true` i zwiększamy o 1 licznik `ls`. Jeśli znacznik `t` jest już ustawiony na `true`, to napotkaliśmy kolejną literę już zliczonego słowa – nic nie robimy. Jeśli napotkamy inny znak, to traktujemy go jako separator i znacznik `t` zawsze zerujemy. Po przeglądnięciu wszystkich znaków łańcucha `s` w zmiennej `ls` mamy liczbę słów. Koniec napisu sygnalizowany jest znakiem zera - `\0`.

Wejście: `s` – łańcuch tekstowy tablica `char s[N]`.

Wyjście: liczba słów zawartych w łańcuchu `s`.

Zmienne pomocnicze:

`i` – indeks znaków w łańcuchu napis, `i=0,1,..., N`

`ls` – licznik słów

`t` – znacznik słowa

Do sprawdzenia czy znak jest literą czy cyfrą stosujemy funkcję:

`int isalnum(int argument)` która zwraca 1 gdy argument jest alfanumeryczny a 0 gdy nie. Można też użyć innych funkcji z pliku nagłówkowego `<ctype.h>`. Do wczytania napisu możemy użyć funkcji `gets()`.

```
// Liczenie słów w lancuchu
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define N 100

int main( ) {
    char napis[N];
    // char napis[] = "to jest napis";
    int i;           // licznik znakow
    int ls = 0;      // licznik slow
    int litcyfr;     // 1 gdy litera lub cyfra
    unsigned char c; // biezacy znak
    int slowo = 0;   // 1 gdy jesteśmy wewnątrz słowa
    printf("Podaj napis: ");
    gets(napis);
    printf("%s\n",napis);
    for(i=0;i<N;i++) {
        c = napis[i];
        // tu trzeba uzupełnić
    }
    printf("\nW tekście jest słów: %d\n",ls);
    return ls;
}
```

Przykład 7-4 Fragment programu wyszukiwania słów w tekście licz\_slowa.c

Przykładowy wynik podany jest poniżej.

```
Podaj napis: Ala ma kota mruczka
Ala ma kota mruczka

0  A  nowe słowo: 1 1
1  l
2  a
3      koniec słowa: 0 1
4  m  nowe słowo: 1 2
5  a
6      koniec słowa: 0 2
7  k  nowe słowo: 1 3
8  o
9  t
10 a
11     koniec słowa: 0 3
12 m  nowe słowo: 1 4
13 r
14 u
15 c
16 z
17 k
18 a
W tekście jest słów: 4
```

## 7.4 Liczenie podłańcuchów w łańcuchu

Do sprawdzania czy w łańcuchu występuje podłańcuch służy funkcja

```
char *strstr(const char *lancuch, const char *podl)
```

lancuch – łańcuch który przeszukujemy

podl – podłańcuch którego szukamy

Funkcja zwraca wskaźnik na pierwsze wystąpienie łańcucha lub NULL gdy podłańcucha nie znaleziono. Poniżej podany jest przykład

```
#include <stdio.h>
#include <string.h>
int main () {
    const char haystack[] = "To jest lancuch do przeszukania";
    const char needle[10] = "jest";
    char *ret;
    ret = strstr(haystack, needle);
    printf("The substring is: %s\n", ret);
    return(0);
}
```

Przykład 7-5 Program string-szukaj.c

**The substring is: jest lancuch do przeszukania**

Napisz program który policzy wystąpienia danego podłańcucha w łańcuchu wprowadzanym z konsoli za pomocą funkcji `gets()`.

(rozwiązanie substring-ile-razy.c)

## 7.5 Sprawdzanie hasła

W oparciu o poniższy wzór napisz program sprawdzający hasło.

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

#define MAX 20

int main(int argc, char *argv[]) {
    int i;
    char haslo[MAX];
    char wzor[MAX] = "tajne";
    printf("Podaj haslo: \n");
    gets(haslo);
    // ...
    return 0;
}
```

Przykład 7-6 Program haslo.c

Gdy hasło jest błędne można je poprawić tylko trzy razy. Użyj funkcji `strcmp(s1,s2)`.

## 7.6 Argumenty funkcji main - kalkulator

Poniższy program wypisuje argumenty funkcji main.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 4

int main(int argc, char *argv[]) {
    int i;
    int op1, op2, wynik;
    char *operacja[MAX] =
        {"plus", "minus", "razy", "podziel"};
    printf("argc = %d\n", argc);
    for (i = 0; i < argc; ++i) {
        printf("%s\n", argv[i]);
    }
    // ...
    return ;
}
```

Przykład 7-7 oblicz.c

W oparciu o ten przykład napisz program oblicz który pełni funkcje kalkulatora linii poleceń. Ma realizować 4 działania a operacja kodowana jest w tablicy:

```
char *operacja[MAX]=
{"plus", "minus", "razy", "podziel"};
```

Program wywołujemy podając w linii poleceń:

```
C:>oblicz argument1 operacja argument2
```

Np:

oblicz 2 razy 4 = 8

```
2 razy 4 = 8
```

Zamiana stringu na liczbę odbywa się za pomocą funkcji `itoa(string);`

## 7.7 Sortowanie tablicy łańcuchów

Napisz funkcje:

`sortuj(char[][N], int size)` – sortowanie tablicy łańcuchów

Wykorzystaj dany poniżej szkielet programu. Do sortowania użyj algorytmu sortowania bąbelkowego.

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

const int M=4; // wiersze
const int N=12; // kolumny

void pisz_tab(char tab[][N], int row);
void sortuj(char[][N], int row);

int main(){
    char t[4][12]={"Tomek", "Jarek", "Zbyszek", "Agata"};
    puts("Tablica przed sortowaniem:\n");

    pisz_tab(t, M);
    sortuj(t, 4);
    puts("\n\nTablica po sortowaniu:\n");
    pisz_tab(t, M);
    getch();
    return 0;
}

void pisz_tab(char tab[][N], int row){
    int j, k;
    for (j=0; j<row; j++) {
        printf("%s \n", tab[j]);
    }
}
```

Przykład 7-8 Sortowanie tablicy łańcuchów – program string-sort.c

### Opis funkcji `sortuj`:

Funkcja porządkuje wiersze w kolejności alfabetycznej. Zastosuj algorytm bąbelkowy znany z poprzednich ćwiczeń. W jego implementacji łańcuchowej występują dwie ważne cechy, różniące ją od postaci używanej do sortowania tablic liczbowych:

1. Do porównania wierszy nie wolno stosować operatorów relacyjnych `>` lub `<`, które nie działają na tablicach. Zamiast tego stosujemy funkcje `strcmp(s1, s2)`.
2. Do kopiowania wierszy tablicy przy ich zamianie miejscami nie można użyć operatora przypisania, bo nie działa on na tablicach. Zamiast tego stosujemy funkcję biblioteczną `strcpy`. Można kopiować całe wiersze będące łańcuchami. Funkcja `strcpy(t[i], t[j])` kopiuje wiersz macierzy `j` na miejsce wiersza `i`.

Wyniki działania programu dane są poniżej.



```

Tablica przed sortowaniem:
Tomek
Jarek
Zbyszek
Agata

Tablica po sortowaniu:
Agata
Jarek
Tomek
Zbyszek

```

## 7.8 Sortowanie 2 tablicy łańcuchów

Napisz funkcje:

`sortuj(char *[])` – sortowanie tablicy łańcuchów

Wykorzystaj dany poniżej szkielet programu. Do sortowania użyj algorytmu sortowania bąbelkowego.

```

// Sortowanie tablicy łańcuchów
#include <stdio.h>
#include <conio.h>
#include <string.h>
const int M=6;
void pisz_tab( char *[]);
void sortuj(char *[]);

int main() {
    char *tab[6]=
    {"ADAM", "TADEUSZ", "EWA", "GRZEGORZ", "IGNACY", "ZENON"};
    pisz_tab(tab);
    sortuj(tab);
    pisz_tab(tab);
    getch();
    return 0;
}

void pisz_tab( char *tab[]) {
    int j;
    for (j=0; j<M; j++)
        puts(tab[j]);
    puts("\n");
}

void sortuj(char *tab[])
{
    // ...
}

```

Przykład 7-9 Sortowanie 2 tablicy łańcuchów – program string-sort2.c

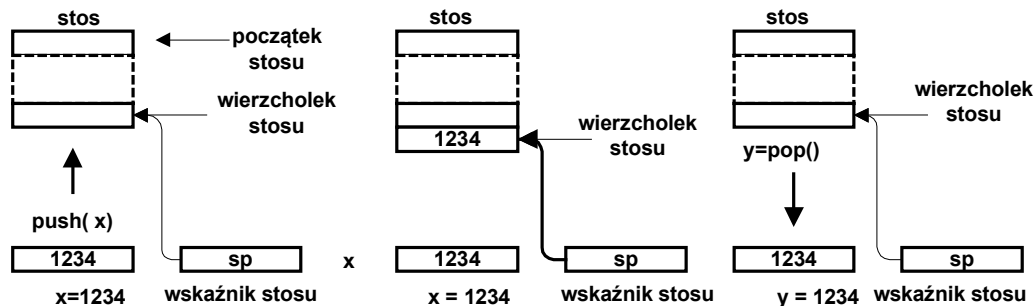
W tym zadaniu tablica do posortowania zapisana jest inaczej, jako `char *tab[6]` czyli tablica wskaźników do łańcuchów.



## 8. Tablice, wektory, macierze

### 8.1 Implementacja stosu w oparciu o tablicę

Działanie stosu i operacje pop i push pokazuje poniższy rysunek.



Rys. 8-1 Ilustracja działania instrukcji `push(x)` i `y = pop()` operujących na stosie

Poniżej podano fragment programu który ma realizować funkcję stosu w oparciu o tablicę `int stack[MAX]` składającą się z liczb `int`.

Należy uzupełnić podany poniżej program o następujące funkcje:

- `void display(int stack[])` – wyświetlanie zawartości stosu i wskaźnika stosu `top`
- `void push(int stack[],int elem)` – wstawianie na stos elementu `elem`
- `int pop(int stack[])` – usunięcie ze stosu elementu z wierzchołka
- `is_empty()` – zwraca 1 gdy stos jest pusty.

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

#define MAX 10 // Maksymalna wielkosc stosu

int stack[MAX]; // tablica stosu
int top;        // wierzcholek stosu

/* wyswietl stos */
void display(int []);

/* push (wstaw) element na stos*/
void push(int [],int);

/* pop (usun) element ze stosu */
int pop(int []);
```

```

void main() {
    int item=0;
    int choice=0;
    top = 0;
    while(1) {
        printf("Wybierz (1: pokaz (display), 2: wstaw (push),
                3: usun (pop)), 4: Koniec...");
        scanf("%d",&choice);
        switch(choice) {
            case 1: display(stack); break;
            case 2:
                printf("podaj liczbe ktora mam wstawic :");
                scanf("%d",&item);
                printf(" wstawiam %d\n",item);
                push(stack,item);
            break;
            case 3: pop(stack); break;
            case 4: exit(0);
            default:
                printf("\nZla opcja");
                break;
        }
    } // end of while(1)
}

```

Przykład 8-1 program stos\_tab1.c

```

Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...
2
podaj liczbe ktora mam wstawic :222
  wstawiam 222
Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...
2
podaj liczbe ktora mam wstawic :333
  wstawiam 333
Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...
1
333 <-- top
222

Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...
2
podaj liczbe ktora mam wstawic :444
  wstawiam 444
Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...:1
444 <-- top
333
222

Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...
3
444 usuniety
Wybierz (1: pokaz (display), 2: wstaw (push), 3: usun (pop)), 4: Koniec...:1
333 <-- top
222

```

Przykład 8-2 Ilustracja działania programu stos

## 8.2 Szukanie binarne (ang. *binary search*)

Przeszukiwanie zbiorów danych jest w informatyce częstym zadaniem. Ważne jest by odbywało się możliwie szybko. Najprostszą metodą przeszukiwania jest przegląd zupełny. Aby sprawdzić czy w tablicy 1000 elementowej znajduje się pewien element (np. 123) trzeba wykonać 1000 sprawdzeń. Procedurę można znacznie przyspieszyć gdy tablica z danymi jest posortowana (np. rosnąco). Wtedy można zastosować algorytm szukania binarnego.

Napisz funkcję `szukaj_bin`, która otrzymuje posortowaną rosnąco tablicę liczb całkowitych `tab` oraz wartość poszukiwaną `liczba` i zwraca informację czy liczba wystąpiła w podanej tablicy `tab`.

```
int szukaj_bin(int tab[], int liczba, int size)
```

Algorytm jest realizowany metodą "dziel i zwyciężaj" (ang. *divide and conquer*). Dzieli on tablicę na mniejsze podtablice do momentu wyszukania pozycji (lub nie w przypadku gdy taki element nie istnieje) elementu szukanego. Metoda może być zaimplementowana jako:

- Iteracyjna (ang. *iterative*)
- Rekurencyjna (ang. *recursive*)

Przeanalizujemy następujący zbiór 7 uporządkowanych liczb:



Założymy że szukamy elementu  $x = 4$ . Potrzebne będą nam trzy zmienne pomocnicze:

$low$  – indeks lewego końca przedziału

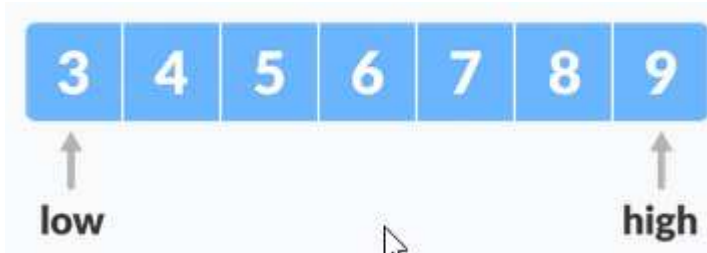
$high$  – indeks prawego końca przedziału

$mid$  – indeks środka przedziału

W pierwszym kroku algorytmu ustawiamy:

```
int low = 0; int high = 6; int mid = (low+high)/2 = 3;
```

A więc mamy:



Znajdujemy element środkowy  $tab[(low + high)/2] = 6$ .



Sprawdzamy czy  $x = tab[mid]$ . Gdy tak algorytm się kończy. Gdy nie:

Sprawdzamy czy  $x > tab[mid]$ . Gdy tak szukany element  $x$  leży na prawo od  $mid$ . Wtedy podstawiamy:

$low = mid + 1;$

Gdy  $x < tab[mid]$  Gdy tak szukany element  $x$  leży na lewo od  $mid$ . Wtedy podstawiamy:

$high = mid - 1;$



Powtarzamy algorytm tak długo aż  $low = high$ .



W końcu znajdujemy że  $x = 4$  na pozycji 1.

Algorytm iteracyjny możemy wyrazić w pseudokodzie jak poniżej.

```
Powtarzaj aż indeksy low i high się nie spotkają {
    mid = (low + high)/2
    if (x == tab[mid])
        return mid
    else if (x > tab[mid]) // x is on the right side
        low = mid + 1
    else // x is on the left side
        high = mid - 1
}
```

Algorytm rekurencyjny dany jest poniżej

```
binarySearch(tab, x, low, high)
    if low > high
        return False
    else {
        mid = (low + high) / 2
        if x == tab[mid]
            return mid
        else if x > tab[mid]
            // x is on the right side
            return binarySearch(tab, x, mid + 1, high)
        else
            // x is on the left side
            return binarySearch(tab, x, low, mid - 1)
    }
```

Napisz wersję iteracyjną i rekurencyjną tego algorytmu.

**Złożoność obliczeniowa** tego sposobu wyszukiwania jest rzędu  $O(\log_2(n))$ .

Oznacza to, że metoda jest znacznie szybsza niż algorytm przeszukiwania liniowego. Dla tablicy 1000-elementowej wystarczy wykonać w pesymistycznej sytuacji około  $\log_2(1000) \approx 10$  kroków, natomiast w przypadku przeszukiwania liniowego należy wykonać tych kroków 1000 czyli złożoność wynosi  $O(n)$ .

### 8.3 Dodawanie i mnożenie wektorów

Napisz funkcję `dodaj_wekt(int a[], int b[], int c[], int size)` która dodaje dwa wektory a i b o wymiarze size, wynik w wektorze c.

Wektor c obliczamy ze wzoru:  $c[i] = a[i] + b[i]$  gdzie  $i=0,1,\dots,size$ .

Napisz funkcję `mnoz_wekt(int a[], int b[], int c[], int size)` która oblicza iloczyn wektorów a i b. Wektor c obliczamy ze wzoru:  $c[i] = a[i] * b[i]$  gdzie  $i=0,1,\dots,size$ .

Wyświetl wektory a,b,c za pomocą funkcji `int print_wekt(int a[], int size)`.

```
#include <stdio.h>
#define N 10

int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int b[] = {11,12, 13, 14, 15, 16, 17, 18, 19, 20};
int c[10];

int dodaj_wekt(int a[], int b[], int c[], int size);
int mnoz_wekt(int a[], int b[], int c[], int size);
int print_wekt(int a[], int size);

int main(void) {
    print_wekt(a,N);
    print_wekt(b,N);
    dodaj_wekt(a,b,c,N);
    print_wekt(c,N);
    mnoz_wekt(a,b,c,N);
    print_wekt(c,N);
}
```

Przykład 8-3 Dodawanie wektorów

Działanie programu pokazane dla dodawania wektorów jest poniżej.

a	1	2	3	4	5	6	7	8	9	10
b	11	12	13	14	15	16	17	18	19	20
	12	14	16	18	20	22	24	26	28	30

### 8.4 Funkcje i tablice 2 wymiarowe

Poniżej dany jest fragment kodu w którym zadeklarowano tablicę 2 wymiarową `int tab[ROW][COL]`. Podano też funkcję `int losuj(zakres)` która zwraca liczbę pseudolosową z zakresu od 0 do zakres. Podano też funkcję `utworz_tab(int tab[ROW][COL],int row, int col)` która wypełnia tablicę wielkościami pseudolosowymi. Należy zaimplementować dwie funkcje:



- `int znajdz_max(int tab[ROW][COL],int row, int col)` – znajdowanie maksimum w tablicy
- `double znajdz_sredn(int tab[ROW][COL],int row, int col)` – znajdowanie wartości średniej z elementów tablicy (suma elementów podzielona przez ich liczbę)

```
#include <stdlib.h>
#include <stdio.h>
#define ROW 5
#define COL 10

int losuj(int zakres);
int utworz_tab(int tab[ROW][COL],int row, int col);
int znajdz_max(int tab[ROW][COL],int row, int col);
double znajdz_sredn(int tab[ROW][COL],int row, int col);

int main() {
    int i,j;
    int max;
    double sredn;
    int tab[ROW][COL];
    // Tworzymy tablice
    utworz_tab(tab,ROW,COL);
    // Maksimum
    max = znajdz_max(tab,ROW,COL);
    printf("maksimum %d\n",max);
    // Srednia
    sredn = znajdz_sredn(tab,ROW,COL);
    printf("Srednia %f\n",sredn);
    return 0;
}
```

```

int losuj(int zakres) {
    int tmp;
    tmp = (int)(rand() / (RAND_MAX + 1.0) * zakres);
    return tmp;
}

int utworz_tab(int tab[ROW][COL],int row, int col)
{
    int i,j;
    int losowa;
    for(i=0;i<row;i++) {
        for(j=0;j<col;j++) {
            losowa = losuj(100);
            // printf("Liczba: %d\n",losowa);
            tab[i][j] =losowa;
            printf("%2d ",losowa);
        }
        printf("\n");
    }
    return 0;
}

```

Przykład 8-4 Tablice 2 wymiarowe maksimum\_tab2.c

## 8.5 Działania na macierzach

Napisz program w którym użyte będą funkcje dodawania i mnożenia macierzy.

$c[i][j] = a[i][j] + b[i][j]$ ,  $C = A + B$

$c[i][j]$  = suma iloczynów wiersza i macierzy a i kolumny j macierzy b

Mnożenie macierzy  $A \cdot B$  jest możliwe jeśli macierz A ma tyle samo kolumn co macierz B ma wierszy. Iloczynem macierzy  $A=[a_{ij}]_{n \times p}$  przez macierz  $B=[b_{ij}]_{p \times m}$  nazywamy macierz  $C=[c_{ij}]_{n \times m}$

taką, że:

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+...+a_{ip}b_{pj}=\sum_{k=1}^p a_{ik}b_{kj}$$

Patrz: [https://pl.wikipedia.org/wiki/Mno%C5%BCenie\\_macierzy](https://pl.wikipedia.org/wiki/Mno%C5%BCenie_macierzy)

## 8.6 Zarządzanie bazą danych opartą na tablicy

Napisz program zarządzania bazą danych opartą na tablicy. Tablica ma zawierać dane osobowe. Są to dane w postaci struktury:

```
typedef struct {
    char imie[MAXLEN];
    char nazwisko[MAXLEN];
    int  pesel;
} osoba_t;
```

W bazie danych wykonujemy typowo cztery operacje:

- Wstawianie rekordu – INSERT
- Kasowanie rekordu – DELETE
- Aktualizacja rekordu – UPDATE
- Uzyskanie dostępu do informacji – SELECT

Np. polecenia języka SQL:

INSERT („Jan”, „Kowalski”, 2);

SELECT \* FROM osoby WHERE pesel = 123;

```
// Baza danych osob oparta na tablicy
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX 50

typedef struct {
    char imie[MAXLEN];
    char nazwisko[MAXLEN];
    int  pesel;
} osoba_t;

int indeks[MAX];

// Tablica zawierajaca struktury typu osoba_t
osoba_t tab[MAX];

int lelem = 0;    // aktualna liczba elementow tablicy
// pozycja na ktorej wstawic nastepny element
int koniec = 0;

// wczytuje dane osoby do struktury osoba
int  wczytaj_osobe(osoba_t *osoba);
// Dodaje osobe do tablicy
int  dodaj_osobe(osoba_t *osoba);
// Szuka osoby o danym pesel i kopiuje do &osoba)
int  szukaj_osobe(int pesel, osoba_t *osoba);
void wyswietl_osobe(osoba_t *osoba);
void wyswietl_liste(osoba_t tab[], int ile);
int  usun_osobe(int pesel);
int  menu(void);
```

```

int main() {
    koniec = 0;
    lelem = 0;
    int pesel;
    int wybor;
    osoba_t osoba;
    printf("Baza danych osob\n");
    do {
        menu();
        wybor = getch();
        switch(wybor) {
            case '1' : wczytaj_osobe(&osoba);
                       dodaj_osobe(&osoba);
                       break;
            case '2' : wyswietl_liste(tab);
                       break;
            case '3' : printf("Szukaj - podaj pesel ");
                       scanf("%d", &pesel);
                       szukaj_osobe(pesel, &osoba);
                       break;
            case '4' : printf("Usun - podaj pesel ");
                       scanf("%d", &pesel);
                       usun_osobe(pesel);
                       break;
            case '5' : cout << "Koniec" << endl;
                       return 0;
        }
    } while(1);
    printf("Koniec\n");
}

int menu(void) {
    printf("1 dodaj, 2 lista, 3 szukaj, 4 usun, 5
koniec\n");
}

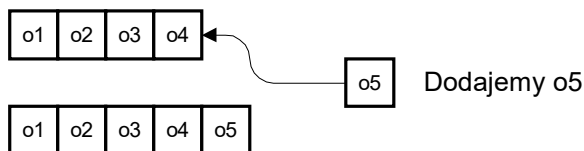
```

Przykład 8-5 Szkielet programu baza-tablica-osoba.c

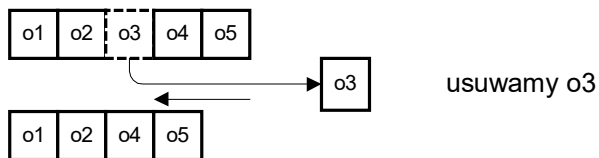
Baza danych ma zapewniać dane niżej funkcje:

<code>int wczytaj_osobe(osoba_t *osoba);</code>	Wczytuje dane osoby ze <code>cin</code> i umieszcza je w strukturze <code>osoba</code> .
<code>void dodaj_osobe(osoba_t *osoba);</code>	Dodaje nowy rekord <code>osoba</code> do tablicy <code>tab</code> na jej końcu na pozycji <code>koniec</code> .
<code>void usun_osobe(int pesel);</code>	Usuwa z tablicy rekord osoby o podanym <code>pesel</code> . Rekordy znajdujące się na prawo od usuniętego elementu muszą zostać dosunięte w lewo.
<code>int szukaj_osobe(int pesel, osoba_t *osoba);</code>	Szukaj osoby o danym <code>pesel</code> i wpisz je do zmiennej <code>&amp;osoba</code> . Funkcja zwraca pozycję w tablicy na której znajduje się szukana osoba lub <code>-1</code> gdy nie znaleziono.
<code>void wyswietl_osobe(osoba_t *osoba)</code>	Wyświetl dane struktury <code>osoba</code> .
<code>void wyswietl_liste(osoba_t tab[], int ile);</code>	Wyświetl całą bazę danych.

Napisz program implementujący powyższe funkcjonalności. W przypadku braku w powyższym opisie pewnych danych przyjmij własne rozwiązania.



Rys. 8-1 Dodanie nowego elementu o5 na końcu tablicy



Rys. 8-2 Usunięcie elementu o3 ze środka tablicy, trzeba przesunąć o4 i o5 w lewo o jedną pozycję

## 9. Operacje na plikach – funkcje niskiego poziomu

### 9.1 Odczyt pliku

Napisz program o nazwie `cat.c` który pobiera nazwę pliku z linii poleceń (`argv[1]` w funkcji `main(int argc, char * argv[])`) i wypisuje jego zawartość na `stdout`. Np.

```
C:>cat cat.c
```

Wypisuje plik `cat.c` na standardowym wyjściu. Wykorzystaj funkcje niskiego poziomu: `open()`, `read()` i `close()`. Nazwa pliku to `argv[1]`.

### 9.2 Kopiowanie plików

Napisz program o nazwie `copy.c` który pobiera z linii poleceń dwie nazwy plików: nazwę pliku źródłowego (`argv[1]` w funkcji `main(int argc, char * argv[])`) i docelowego (`argv[2]`). Np.

```
C:>copy zrodlowy.txt docelowy.c
```

```
$cp zrodlowy.txt docelowy.c
```

Program ma kopiować plik źródłowy na docelowy. Poniżej dano pseudokod

```
Nazwa pliku źródłowego - argv[1]
Nazwa pliku docelowego - argv[2]
char buf[SIZE]
Otwórz plik źródłowy
Utwórz plik docelowy
do {
    czytaj z pliku źródłowego SIZE bajtów do bufora buf
    zapisz z bufora buf bajty do pliku wynikowego (tyle ile
    odczytano)
} while (da się coś odczytać z pliku źródłowego)
zamknij plik źródłowy
zamknij plik docelowy
```

### 9.3 Baza danych osobowych oparta na tablicy, rozszerzenie o zapis do pliku

Do poprzedniego programu zarządzającego bazą danych osobowych (zad 8.6) dopisz dwie funkcje zapisu bazy na dysk i odczytu bazy z dysku.

```
int zapisz() - funkcja zapisuje tablicę tab[MAX] na dysk
int odczyt() - funkcja odczytuje tablicę tab[MAX] z dysku
```

Użyj biblioteki niskiego poziomu dostępu do pliku, funkcje: `open`, `close`, `read`, `write`. Przykład odczytu z pliku podano poniżej.

```
// Przyklad zapisu i odczytu z pliku
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#define MAXLEN 80

typedef struct {
    char imie[MAXLEN];
    char nazwisko[MAXLEN];
    int pesel;
} osoba_t;

int main(int argc, char *argv[]) {
    int wr,rd,res;
    osoba_t zelement;
    osoba_t oelement;
    int i;

    // Wpisywanie danych -----
    strcpy(zelement.imie, "Jan");
    strcpy(zelement.nazwisko, "Kowalski");
    zelement.pesel = 123;

    printf("Zapis \n");

    // Zapis do pliku -----
    wr = open("plik.bin",O_WRONLY | O_CREAT,0666);
    if(wr < 0) {
        perror("open - zapis");
        return 0;
    }
    // Zapis do pliku
    res = write(wr,&zelement,sizeof(zelement));
    if(res < 0) perror("zapis");
    else { printf("Zapisano: %d bajtow\n",res);
           printf("%s %s %d\n",
                 zelement.imie,zelement.nazwisko,zelement.pesel);
    }
    close(wr);

    // Odczyt z pliku -----
    rd = open("plik.bin",O_RDONLY);
    if(rd < 0) {
        perror("open - odczyt");
        return 0;
    }
    do {
        res = read(rd,&oelement,sizeof(oelement));
        if(res < 0) perror("odczyt");
        else {
            printf("Odczytano: %d bajtow\n",res);
            printf("%s %s %d\n",
                  oelement.imie,oelement.nazwisko,oelement.pesel);
        } while(res != 0)
    } while(res != 0)
    close(rd);
    return rd;
}
```

**Przykład 9-1** Zapisu i odczytu struktury z pliku file\_rd\_wr.c

## 9.4 Przykład użycia plików – baza danych Hotel

Program realizuje funkcję bazy danych dla hotelu. Po uruchomieniu wprowadzane jest menu jak niżej.

- 1 – melduj
- 2 – wymelduj
- 3 – kto w pokoju
- 4 – lista
- 5 – szukaj nazwisko
- 6 – szukaj pesel
- 7 - koniec

Program główny wyświetla menu i wczytuje wybór operatora `opt` zwracany przez funkcję `menu()`. Następnie w zależności od wartości zmiennej `opt` wykonywana jest odpowiednia funkcja.

```
do {
    opt = menu();
    switch(opt) {
        case MELD      : melduj(bazaf);          break;
        case WYMELD    : wymeld(bazaf);          break;
        case KTO       : info(bazaf);             break;
        case LISTA     : lista(bazaf);            break;
        case ZNAJDZ_N   : znajdz_nazw(bazaf);     break;
        case ZNAJDZ_P   : znajdz_pesel(bazaf);    break;
        case WYJSCIE    : _exit(0);
        default        : ;
    }
} while(stop != 1);
```

Przykład 9-2 Wyświetlenie listy wyboru i główna pętla programu.

Poniżej zostaną opisane funkcje składające się na akcje wykonywane przez program.

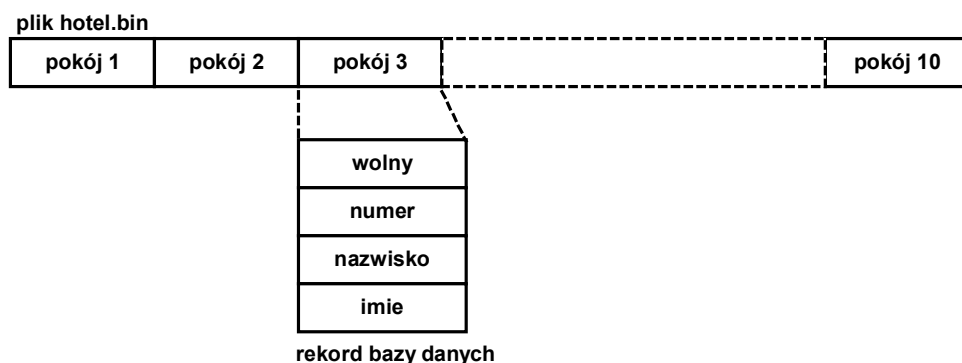
### Funkcja `init(bazaf,NPOKOJOW)`

Podstawą działania programu jest baza danych zapamiętana w pliku o nazwie "hotel.bin". Na początku program sprawdza czy taki plik istnieje. Gdy nie tworzy plik `bazaf` i pustą bazę danych składającą się z rekordów w postaci struktur:

```
typedef struct {
    int wolny;
    int numer;
    char nazwisko[NSIZE];
    char imie[NSIZE];
    int pesel;
} pokoj_t;
```



Rekordów będzie NPOKOJOW w tym przykładzie 10. Pola numer przyjmują wartości 1,2,...,10. Pola wolny ustawiane są na 1. Pola Imie i Nazwisko są pustymi napisami.



Rys. 9-1 Struktura bazy danych

#### Funkcja lista(char \* plik)

Funkcja otwiera plik do odczytu, a następnie odczytuje kolejne rekordy bazy danych i je wypisuje. Odczyt jest w pętli i kończy się gdy funkcja read(...) zwraca 0.

#### Funkcja znajdz\_nazw(char \* plik)

Funkcja prosi o podanie szukanego nazwiska a następnie otwiera plik do odczytu. Dalej odczytuje kolejne rekordy pokój i porównuje szukane nazwisko z polem pokoj.nazwisko. Gdy są takie same funkcja się kończy.

#### Funkcja znajdz\_pesel(int pesel)

Funkcja prosi o podanie peselu szukanej osoby a następnie otwiera plik do odczytu. Dalej odczytuje kolejne rekordy pokój i porównuje szukany pesel z polem pokoj.pesel. Gdy są takie same funkcja się kończy.

#### Funkcja melduj(char \* plik)

Funkcja odczytuje kolejne rekordy pokoi by znaleźć wolny pokój. Gdy znajdzie pokój o numerze numer, tworzy rekord pokoju o nazwie pokój. Następnie ustawia wskaźnik bieżącej pozycji pliku na ten rekord wykonując funkcję:

```
Iseek(fh,numer*sizeof(pokoj),SEEK_SET)
```

Dalej wpisuje rekord pokój w bieżącej pozycji pliku aktualizując tym samym poprzednią jego zawartość.

#### Funkcja wymeld(char \* plik)

Po podaniu numeru pokoju funkcja ustawia wskaźnik bieżącej pozycji pliku na ten pokój wykonując funkcję:

```
Iseek(fh,(numer - 1)*sizeof(pokoj),SEEK_SET)
```

Dalej tworzy rekord pokój dla pustego pokoju i zapisuje go do pliku.

#### Funkcja info(char \* plik)

Funkcja wczytuje numer pokoju a następnie ustawia wskaźnik bieżącej pozycji pliku na ten pokój wykonując funkcję:

```
Iseek(fh,(numer - 1)*sizeof(pokoj),SEEK_SET)
```

Dalej odczytuje rekord pokoju wykonując funkcję: `read(fh,&pokoj,sizeof(pokoj))` i wyprowadza dane pokoju na ekran.

## 10. Operacje na plikach – biblioteka standardowa

### 10.1 Zapisywanie linii tekstu do pliku

Napisz program który zapisuje linie tekstu wprowadzane ze `stdin` do pliku o nazwie `plik1.txt`. Program powinien:

- Utworzyć plik o nazwie `plik1.txt` w trybie zapisu i sprawdzić czy się udało (funkcja `fopen()`).
- W pętli wczytywać kolejne linie wprowadzane przez operatora z klawiatury funkcja `char *gets(char *str);`
- Zapisywać wprowadzane linie do pliku funkcja `int fputs(const char *str, FILE *stream)`
- Sprawdzić czy wprowadzono napis "koniec". Gdy tak zamknąć plik i wypisać liczbę wprowadzonych linii.

Przykład dany poniżej.

```
wprowadz kolejne linie
linia "koniec" konczy program
linia1
linia dwa
linia trzy i pol
koniec

zapisano 4 linii
```

### 10.2 Odczyt z pliku tekstowego i liczenie linii

Napisz program `listuj.c` który odczytuje linie z pliku tekstowego, wyprowadza linie na `stdout` i liczy te linie. Nazwa pliku do wylistowania powinna być parametrem podanym po nazwie programu jako `argv[1]`. Na przykład:

```
listuj plik.txt
cat nazwa_pliku
```

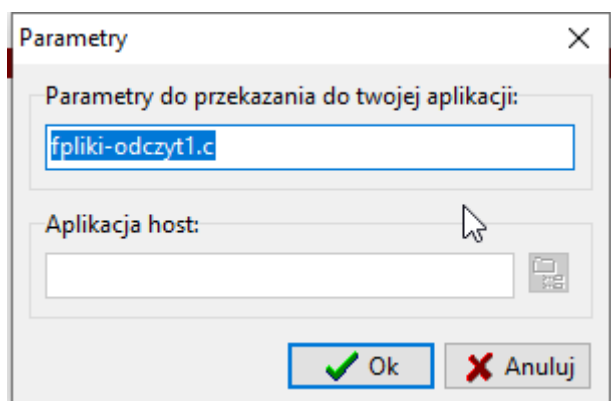
Nazwę pliku można uzyskać z parametrów funkcji `main` jako `argv[1]`.

```
int main(int argc, char *argv[]) {
    printf("Parametr %s\n", argv[1]);
    ...
}
```

Aby napisać program należy:

- Otworzyć plik o nazwie podanej jako `argv[1]` w trybie odczytu i sprawdzić czy się udało (funkcja `fopen`).
- W pętli wczytywać kolejne linie pliku funkcja `char * fgets( char * str, int num, FILE * stream );`
- Wypisać odczytane linie na `stdout`.
- Sprawdzić czy osiągnięto koniec pliku funkcja `feof()`. Gdy tak zamknąć plik i wypisać liczbę odczytanych linii.

Parametry programu można podać z menu Uruchom / Parametry lub linii polecenia jak pokazano poniżej.



```
d:\WSH\Wstep-do-progr\lab\Programy>fpliki-odczyt1 hello.c
// Pierwszy program
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello Word\n");
    return 0;
}

odczytano 9 linii
```

### 10.3 Baza danych - tablica, rozszerzenie o zapis do pliku bibl. standard.

Do poprzedniego programu zarządzającego bazą danych osobowych (zad 8.6) dopisz dwie funkcje zapisu bazy na dysk i odczytu bazy z dysku.

int zapisz() - funkcja zapisuje tablicę tab[MAX] na dysk  
 int odczyt() - funkcja odczytuje tablicę tab[MAX] z dysku

Użyj biblioteki standardowej dostępu do pliku, funkcje: fopen, fclose, fread, fwrite. Przykład odczytu z pliku podano poniżej.

W programie tym ważne jest to że zapisywane i odczytywane rekordy są stałej długości.

```

// Przyklad zapisu i odczytu z pliku
// uzyta biblioteka standardowa.
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#define MAXLEN 80

typedef struct {
    char imie[MAXLEN];
    char nazwisko[MAXLEN];
    int pesel;
} osoba_t;

int main(int argc, char *argv[]) {
    int res;
    FILE * wr, *rd;
    osoba_t zelement;
    osoba_t oelement;
    // wpisujemy dane -----
    strcpy(zelement.imie, "Jan");
    strcpy(zelement.nazwisko, "Kowalski");
    zelement.pesel = 123;

    printf("Zapis \n");

    // Zapis do pliku -----
    wr = fopen("plik.bin","w");
    if(wr == NULL) {
        perror("open - zapis");
        return 0;
    }
    // Zapis do pliku
    res = fwrite(&zelement,sizeof(zelement),1,wr);
    if(res < 0) perror("zapis");
    else { printf("Zapisano: %d bajtow\n",res);
           printf("%s %s %d\n",
                 zelement.imie,zelement.nazwisko,zelement.pesel);
    }
    fclose(wr);

    // Odczyt z pliku -----
    rd = fopen("plik.bin","r");
    if(wr == NULL) {
        perror("fopen - odczyt");
        return 0;
    }
    res = fread(&oelement,sizeof(oelement),1,rd);
    if(res < 0) perror("odczyt");
    else {
        printf("Odczytano: %d bajtow\n",res);
        printf("%s %s %d\n",
              oelement.imie,oelement.nazwisko,oelement.pesel);
    }
    fclose(rd);
    return 0;
}

```

Przykład 10-1 Zapisu i odczytu struktury z pliku write\_read\_std.c

## 10.4 Odczyt z pliku tekstowego i liczenie słów kluczowych

Napisz program `szukaj.c` który odczytuje linie z pliku tekstowego, wyprowadza na `stdout` te linie które zawierają podane jako parametr słowo kluczowe. Program odpowiada linuxowemu filtrowi `grep`. Nazwa pliku do wylistowania i słowo kluczowe powinno być parametrem podanym po nazwie programu. Na przykład:

```
szukaj plik.txt printf
(analogicznie jak grep plik wzor)
```

Nazwę pliku i słowo kluczowe można uzyskać z parametrów funkcji `main` jako `argv[1]` i `argv[2]`.

```
int main(int argc, char *argv[]) {
    printf("Parametry %s %s\n",argv[1],argv[2]);
    ...
}
```

Konstrukcja programu podobna jest do poprzedniego. Do przeszukiwania wczytywanych linii wykorzystaj funkcję biblioteczną `strstr(char * linia, char *wzor)`

Funkcja zwraca wskaźnik na znaleziony wzór lub `NULL` co pokazuje poniższy przykład

```
#include <stdio.h>
#include <string.h>
int main () {
    char string[55] = "This is a test string for testing";
    char *p;
    p = strstr(string, "test");
    if(p != NULL) {
        printf("string %s found\n",p);
    }
    else printf("string not found\n" );
    return 0;
}
```

Przykład 10-2 Szukanie wzorca w łańcuchu

## 10.5 Baza danych osobowych w pliku

Baza danych osobowych zawiera następujące informacje:

```
typedef struct { // Rekord bazy danych ----
    char imie[MAXLEN]; // Imie
    char nazwisko[MAXLEN]; // Nazwisko
    int pesel; // Pesel
    int zajety; // 1 gdy rekord zajety, 0 gdy nie
} osoba_t;
```

Dane o osobach pamiętane są w pliku. Plik składa się z rekordów zawierających struktury `osoba_t`. Operacje które powinny być możliwe to;

- Dodawanie nowego rekordu
- Aktualizacja istniejącego rekordu
- Kasowanie rekordu
- Poszukiwanie rekordu o danym peselu

W bazie o tej strukturze powstaje problem kasowania rekordu. Nie jest poprawne przepisywanie wszystkich rekordów leżących na prawo od skasowanego rekordu w celu likwidacji "dziury", po rekordzie skasowanym. Zamiast tego oznaczamy taki rekord jako wolny ustawiając pole `zajety = 0`.

Rekordy oznaczone jako wolne będą pomijane przy listowaniu bazy danych.

Zapis nowego rekordu polega na odnalezieniu rekordu oznaczonego jako wolny, a gdy takiego nie ma, wpisanie nowego rekordu na końcu.



Rys. 10-1 Struktura bazy danych osobowych

Należy zaimplementować funkcje:

- Aktualizacja pola `imie` dla rekordu o podanym peselu
- Poszukiwanie rekordu na podstawie nazwiska

Kod programu z bazą danych dany jest poniżej.

```
// Baza danych osob oparta pliku
// Zapis/odczyt bazy danych do/z pliku
// Napisał: Jędrzej Ulasiewicz
#include <stdio.h>
#include <conio.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

#define BNAME "baza2.dat"
```

```

#define MAXLEN 40
#define MAX 50

typedef struct { // Rekord bazy danych ----
    char imie[MAXLEN]; // Imie
    char nazwisko[MAXLEN]; // Nazwisko
    int pesel; // Pesel
    int zajety; // 1 gdy rekord zajety, 0 gdy nie
} osoba_t;

int lelem = 0; // Ile elementow w bazie

int wczytaj_osobe(osoba_t *osoba); // Wczytanie danych osoby
int dodaj_osobe(osoba_t *osoba); // Dodanie osoby do bazy
// Szukanie osoby na podst. peselu
int szukaj_osobe(int pesel, osoba_t *osoba);
void wyswietl_osobe(osoba_t *osoba); // Wyszwietlanie danych osoby
int usun_osobe(int pesel); // Usuwanie osoby na podstawie peselu
int odczyt(void); // Odczyt bazy danych
int popraw(); // Poprawienie Imienia - do zrobienia
int menu(void); // Wypisanie menu

int main() {
    lelem = 0;
    int pesel;
    int wybor, res;
    osoba_t osoba;
    printf("Baza danych osob\n");
    res = odczyt();

    do { // petla glowna -----
        menu();
        wybor = getch();
        switch(wybor) {
            case '1' : wczytaj_osobe(&osoba);
                       dodaj_osobe(&osoba);
                       break;
            case '2' : odczyt();
                       break;
            case '3' : printf("Szukaj - podaj pesel: ");
                       scanf("%d", &pesel);
                       szukaj_p(pesel);
                       break;
            case '4' : printf("usun - podaj pesel: ");
                       scanf("%d", &pesel);
                       usun_osobe(pesel);
                       break;
            case '5' : popraw();
                       break;
            case '6' : printf("Koniec\n");
                       return 0;
        }
    } while(1);
    printf("Koniec\n");
}

```



```

int dodaj_osobe(osoba_t *osoba) {
    int i = 0;
    int rd, res, poz;
    FILE *fd;
    osoba_t os;
    fd = fopen(BNAME, "r+");
    if(fd == NULL) {
        perror("Dodaj");
        return -1;
    }
    // Szukanie wolnej pozycji ----
    printf("Dodaj\n");
    do {
        rd = fread(&os, sizeof(osoba_t), 1, fd);
        printf("fread %d, i= %d \n", rd, i);
        if(rd == 0) break;
        // printf("Odczyt: %s %s %d
%d\n", os.nazwisko, os.imie, os.pesel, os.zajety);
        if(os.zajety == 1) {
            printf("rekord %d zajety\n", i);
        } else {
            printf("Znaleziono wolny rekord poz: %d\n", i);
            break;
        }
        i++;
    } while(1);
    printf("indeks %d\n", i);
    // Zaznaczamy ze rekord zajety
    osoba->zajety = 1;
    poz = fseek(fd, i*sizeof(osoba_t), SEEK_SET);
    res = fwrite(osoba, sizeof(osoba_t), 1, fd);
    printf("dodano na pozycji: %d wynik: %d %s\n", i, res, osoba->
nazwisko);
    fclose(fd);
    lelem++;
    return res;
}

int wczytaj_osobe(osoba_t *osoba) {
    int cnt;
    printf("Podaj imie: ");
    scanf("%s", &osoba->imie);
    cnt++;
    printf("Podaj nazwisko: ");
    scanf("%s", &osoba->nazwisko);
    cnt++;
    printf("Podaj pesel: ");
    scanf("%d", &osoba->pesel);
    cnt++;
    osoba->zajety = 1;
    return cnt;
}

void wyswietl_osobe(osoba_t *osoba) {
    int cnt;
    printf("%s %s  %d \n", osoba->imie, osoba->nazwisko, osoba->pesel);
    return ;
}

```

```

int szukaj_p(int pesel) {
    int res;
    FILE * fh;
    int i=0;
    osoba_t os;
    printf("Szukaj pesel: %d\n",pesel);
    fh = fopen(BNAME,"r" );
    if(fh < 0) {
        perror("open: ");
        return 0;
    }
    do {
        res = fread(&os,sizeof(osoba_t),1,fh);
        // printf("odczyt el: %d wynik: %d \n",i,res);
        if(res != 1) break;
        if(os.zajety == 0) {
            i++;
            continue;
        };
        printf(" %s %s %d \n",os.imie,os.nazwisko,os.pesel);
        if(os.pesel == pesel) {
            printf("Znaleziono poz: %d %s %s %d \n",i,os.imie,
os.nazwisko, os.pesel);
            fclose(fh);
            return i;
        }
        i++;
    } while(1);
    fclose(fh);
    return -1;
}

int usun_osobe(int pesel) {
    int res,poz,rd,i = 0;
    int pozycja = 0;
    FILE * fd;
    osoba_t os;
    printf("Usun: %d \n",pesel);
    // Najpierw szukamy osoby
    pozycja = szukaj_p(pesel);
    if(pozycja < 0) {
        printf("nie znaleziono\n");
        return 0;
    }
    fd = fopen(BNAME,"r+");
    if(fd == NULL) {
        perror("Usun");
        return -1;
    }
    // Szukanie wolnej pozycji ----
    printf("Usun pozycja: %d\n",pozycja);
    // Zaznaczamy ze rekord zajety
    os.zajety = 0;
    strcpy(os.imie,"?");
    strcpy(os.nazwisko,"?");
    os.pesel = 0;
    poz = fseek(fd,pozycja*sizeof(osoba_t),SEEK_SET);
    res = fwrite(&os,sizeof(osoba_t),1,fd);
    printf("skasowano na pozycji: %d wynik: %d %s\n",pozycja,
res,os.nazwisko);
}

```

```

        fclose(fd);
        return 1;
    }

int menu(void) {
    printf("1 dodaj, 2 lista, 3 szukaj, 4 usun, 5 popraw, 6 koniec\n");
}

int odczyt() {
    int res;
    FILE * fh;
    int i=0;
    osoba_t os;
    printf("Odczyt\n");
    fh = fopen(BNAME,"r" );
    if(fh < 0) {
        perror("open: ");
        return 0;
    }
    do {
        res = fread(&os,sizeof(osoba_t),1,fh);
        // printf("odczyt el: %d wynik: %d \n",i,res);
        if(res != 1) break;
        if(os.zajety == 1) {
            printf(" %s  %s %d \n",os.imie,os.nazwisko,os.pesel);
            i++;
        }
    } while(1 );
    fclose(fh);
    return i;
}

int popraw() {
    int pesel;
    printf("Popraw - podaj pesel: %d \n",pesel);
    printf("Do zrobienia\n");
    // Tu nalezy wpisac funkcje
    // Odszukac rekord z danym peselem, znalezc pozycje
    // Odczytac rekord
    // Zmienic pole
    // Ustawic wskaznik biezacej pozycji w pliku
    // Zapisac
}

```

**Przykład 10-3** Program baza-osoba-file-c.c

## 10.6 Baza danych hotel

Większość baz danych operuje na rekordach o stałej strukturze zawierających pewne informacje. Zwykle zapewniane są następujące funkcje bazy danych:

- Dodawanie rekordu - insert
- Aktualizacja rekordu - update
- Kasowanie rekordu - delete
- Poszukiwanie rekordu - select

Korzystając ze wzorca podanego na wykładzie (przykład hotel) napisz program realizujący funkcję bazy danych o obywatelach. Pojedynczy rekord będzie strukturą zawierającą następujące pola:

```
#define NSIZE 40
#define ASIZE 80

typedef struct {          // Rekord opisu osoby
    int  wolny;           // 1 gdy rekord wolny 0 gdy zajety
    char nazwisko[NSIZE];
    char imie[NSIZE];
    char adres[ASIZE];
    int  pesel;           // Numer pesel
} osoba_t;
```

Baza powinna realizować następujące funkcje:

- Dodawanie nowej osoby
- Zmiana adresu
- Listowanie bazy
- Przeszukiwanie bazy na podstawie numeru PESEL
- Przeszukiwanie bazy na podstawie nazwiska

Baza powinna uniemożliwiać wprowadzenie dwóch osób o tym samym numerze pesel. Wykorzystaj funkcje biblioteki standardowej: `fopen`, `fwread`, `fwrite`, `fseek`, `fclose`, `feof`. Wykorzystaj poniższy kod.

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>

#define SIZE 80
#define POKOJOW 10
#define NSIZE 30
#define MELD 1
#define WYMELD 2
#define KTO 3
#define LISTA 4
#define ZNAJDZ 5
#define WYJSCIE 6

typedef struct {
    int wolny;
```

```

        int numer;
        char nazwisko[NSIZE];
        char imie[NSIZE];
    } pokoj_t;

char bazaf[] = "hotel_f.bin";
pokoj_t pokoj;

int menu(void) {
    int opcja;
    printf("\n1 - meldowanie\n");
    printf("2 - wymeldowanie\n");
    printf("3 - kto w pokoju\n");
    printf("4 - lista\n");    printf("5 - znajdz\n");
    printf("6 - wyjscie\n"); scanf("%d",&opcja);
    return opcja;
}

int main(int argc, char*argv[]) {
    int opt,res,stop=0;
    struct stat buf;
    res = stat(bazaf,&buf);
    if(res != 0) {
        // Plik nie istnieje - inicjacja bazy
        printf("Brak pliku bazy - tworzenie\n");
        init(bazaf,POKOJOW);
    }
    do {
        opt = menu();
        switch(opt) {
            case MELD      : melduj(bazaf); break;
            case WYMELD    : wymeld(bazaf); break;
            case KTO       : info(bazaf);    break;
            case LISTA     : lista(bazaf);   break;
            case ZNAJDZ    : znajdz(bazaf);  break;
            case WYJSCIE   : _exit(0);
            default        : printf("zly wybor");
        }
    } while(stop != 1);
    return 0;
}

```

```
1 - meldowanie
2 - wymeldowanie
3 - kto w pokoju
4 - lista
5 - znajdz
6 - wyjscie
4
1 Ulasiewicz Jędrzej
2 Kowalski Jan
3 wolny
4 Słodowy Adam
5 wolny
6 wolny
7 wolny
8 wolny
9 wolny
10 wolny
```

## 11. Wskaźniki

### 11.1 Dostęp wskaźnikowy do elementów tablicy

Poniżej dany jest program znajdowania maksimum w tablicy. Dostęp do elementów tablicy jest poprzez jej indeks.

```
// Obliczanie maksimum z tablicy
#include <stdlib.h>
#define DIM 8

int tab[DIM] = {4,1,6,89,12,6,33,7};

int main() {
    int i;
    int max = 0;
    for(i=0;i<DIM;i++) {
        if(max < tab[i]) max = tab[i];
    }
    return max;
}
```

Przykład 11-1 Przykład użycia tablicy w języku C – program maximum.c

Zmodyfikuj ten program tak, by przeszukiwanie tablicy odbywało się za pomocą wskaźników.

## 11.2 Tablica dynamiczna

Poniżej dany jest program demonstrujący działanie tablicy dynamicznej.

```
// Tablica dynamiczna
#include <stdio.h>
#include <stdlib.h>

int main ( ) {
    int i;

    // ilosc elementow tablicy
    int n;
    // wczytanie ilosci elementow tablicy
    printf("Podaj ilosc elementow tablicy: \n" ) ;
    scanf("%d",&n);

    // wskaznik na zmienna typu int
    int * wski = NULL;
    // przydzielenie pam na n elemenotwa tablice int
    wski = malloc(n * sizeof(int));

    // wpis do tablicy
    for ( i=0; i<n; i++)
        wski[i] = i ;

    // wyswietlenie zawartości tablicy
    for(i=0; i<n ; i++)
        printf("%d ",wski[i]);
    printf("\n");

    // zwolnienie pamieci przydzielonej na tablice
    free(wski);
    wski = NULL ;
    return 0;
}
```

**Przykład 11-1** Program tablica-dynamiczna1.c

Wyniki działania programu:

Podaj ilosc elementow tablicy:

6

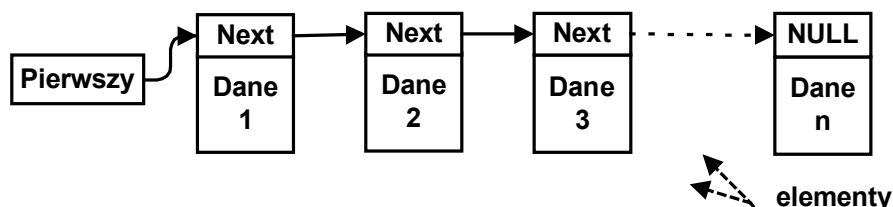
0 1 2 3 4 5

Przekształć ten program tak żeby tablica zawierała elementy double i dostęp do niej był nie za pomocą indeksu ale wskaźników.



### 11.3 Lista jednokierunkowa

Jedną z często spotykanych struktur danych są listy. Lista składa się z elementów (zwykle struktur) zawierających różnego typu dane. Elementy te tworzą połączenie jak pokazano na poniższym rysunku.



Rys. 11-1 Lista jednokierunkowa

Każdy z elementów listy zawiera jakieś dane oraz wskaźnik Next na następny element listy. Listy często są sortowane względem jakiegoś pola.

Podstawowe operacje zdefiniowane na liście:

- Inicjacja pustej listy
- Dodanie elementu do listy (ang. *insert*) - na początku, na końcu, po określonym elemencie
- Usunięcie elementu z listy (ang. *remove*) – z początku, z końca, ze środka po wartości jakiegoś pola
- Przeglądanie elementów listy (iterowanie)
- Szukanie na liście pewnego elementu

Prostą listę jednokierunkową można zbudować definiując jej element `node_t` jak niżej.

```
typedef struct node {
    int val;
    struct node * next;
} node_t;
```

Lista ta w polu `val` przechowuje wartość `int`, ale może przechowywać cokolwiek innego.

Poniżej podano program przykładowy, uruchom poniższy program.

```
// Ilustracja operacji na liscie
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int val;
    struct node * next;
} node_t;

int  print_list(node_t * head);
void push(node_t * head, int val);
int  pop(node_t ** head);

int main(void ) {
    int i;
    node_t * head = NULL;
    // Tworzymy pierwszy element
    head = (node_t *) malloc(sizeof(node_t));
    if (head == NULL) {
        return 1;
    }
    head->val = 1;
    head->next = NULL;
    // Dodajemy 4 elementy
    for(i=2;i<=5;i++) {
        push(head,i);
    }
    // Drukujemy liste
    print_list(head);
    // usuwamy pierwszy element
    pop(&head);
    // Drukujemy liste
    print_list(head);
    return 0;
}
```

```

int print_list(node_t * head)
// Wypisanie całej listy
// head - poczatek
{
    int cnt = 0;
    printf("Lista -----\\n");
    node_t * current = head;
    while (current != NULL) {
        printf(" pozycja: %d val = %d\\n", cnt, current->val);
        current = current->next;
        cnt++;
    }
    printf("-----\\n");
    return cnt;
}

void push(node_t * head, int val)
// Dodanie elementu na koncu listy
{
    node_t * current = head;
    printf("Push: %d\\n", val);
    while (current->next != NULL) {
        current = current->next;
    }

    /* now we can add a new variable */
    current->next = (node_t *) malloc(sizeof(node_t));
    current->next->val = val;
    current->next->next = NULL;
}

int pop(node_t ** head)
// Usuniecie elementu z poczatku listy
// head - poczatek listy
{
    int retval = -1;
    node_t * next_node = NULL;
    printf("pop - usuwam pierwszy element\\n");
    if (*head == NULL) {
        return -1;
    }

    next_node = (*head)->next;
    retval = (*head)->val;
    printf("pop: %d\\n", retval);
    free(*head);
    *head = next_node;
    return retval;
}

```

Przykład 11-2 Program lista1.c – operacje na liście

**Wyniki działania:**

```

Push: 2
Push: 3
Push: 4
Push: 5
Lista -----
  pozycja: 0 val = 1
  pozycja: 1 val = 2
  pozycja: 2 val = 3
  pozycja: 3 val = 4
  pozycja: 4 val = 5
-----
pop - usuwam pierwszy element
pop: 1
Lista -----
  pozycja: 0 val = 2
  pozycja: 1 val = 3
  pozycja: 2 val = 4
  pozycja: 3 val = 5
-----

```

Dopisz do programu następujące funkcje:

**A) Funkcję szukającą na liście rekordu w którym pole val == xval**

```

int szukaj(node_t *head, int xval)
// szukanie na liście elementu val
// Funkcja zwraca numer pozycji lub -1 gdy nie znaleziono

```

**B) Funkcję wstawiającą nowy element na początku listy**

```

void ppush(node_t ** head, int val)
// Dodanie elementu val na początku listy

```

**C) Funkcję usuwającą z niej element o danym kluczu klucz.**

```

int lremove(node_t ** head, int klucz)

```

Kod do testowania funkcji może być jak poniżej.

```

// Ilustracja operacji na liście
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int val;
    struct node * next;
} node_t;

int print_list(node_t * head);
void push(node_t * head, int val);
int lpop(node_t ** head);
int lremove(node_t ** head, int klucz);
void ppush(node_t ** head, int val);
int szukaj(node_t *head, int xval);

```

```

int main(void ) {
    int i;
    node_t * head = NULL;
    // Tworzymy pierwszy element
    head = (node_t *) malloc(sizeof(node_t));
    if (head == NULL) {
        return 1;
    }
    head->val = 1;
    head->next = NULL;
    // Dodajemy 4 elementy
    for(i=2;i<=5;i++) {
        push(head,i);
    }
    // Drukujemy liste
    print_list(head);
    // usuwamy pierwszy element
    lpop(&head);
    // Drukujemy liste
    print_list(head);
    lremove(&head,3);
    print_list(head);
    ppush(&head,77);
    print_list(head);
    szukaj(head,77);
    return 0;
}

```

### Przykład 11-3 Program lista1-rozw.c

Wyniki testu mogą być jak poniżej.

```

Push: 2
Push: 3
Push: 4
Push: 5
Lista -----
pozycja: 0 val = 1
pozycja: 1 val = 2
pozycja: 2 val = 3
pozycja: 3 val = 4
pozycja: 4 val = 5
-----
pop: 1
Lista -----
pozycja: 0 val = 2
pozycja: 1 val = 3
pozycja: 2 val = 4
pozycja: 3 val = 5
-----
Usun: 3
Znaleziono 3 na pozycji: 0
Lista -----
pozycja: 0 val = 2
pozycja: 1 val = 4
pozycja: 2 val = 5
-----

```

```
PPush: 77
Lista -----
  pozycja: 0 val = 77
  pozycja: 1 val = 2
  pozycja: 2 val = 4
  pozycja: 3 val = 5
-----
Szukaj: 77
Znaleziono na pozycji: 0
```

## 12. Źródła

- 1) Artur Pyszczyk, Programowanie w języku C dla początkujących oraz średnio zaawansowanych programistów. <https://www.arturpyszczyk.pl/files/c/pwc.pdf>
- 2) TutorialsPoint, Programming C tutorial. <https://www.tutorialspoint.com/cprogramming/index.htm>
- 3) Wikibooks, kurs programowania w C, <https://pl.wikibooks.org/wiki/C>
- 4) Kompilator gcc online, [https://www.onlinegdb.com/online\\_c\\_compiler](https://www.onlinegdb.com/online_c_compiler)
- 5) Instalacja bibliotek w Dev-C++, <http://cpp0x.pl/artykuly/?id=49>
- 6) Pakiety oprogramowania <https://sourceforge.net/projects/devpaks/>
- 7) Programming with the Dev C++ IDE, <http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf>
- 7) Jerzy Wałaszek, Algorytmy i struktury danych [https://eduinf.waw.pl/inf/alg/001\\_search/index.php](https://eduinf.waw.pl/inf/alg/001_search/index.php)