

Ćwiczenie nr 2

Programowanie imperatywne

Środowisko uruchomieniowe

1. Pobrać plik **TC.zip** (*Turbo C Compiler*)
(<http://www.cs.put.poznan.pl/awidelska/TC.zip>).
2. Rozpakować powyższe archiwum w katalogu roboczym (na zajęciach najprawdopodobniej C:\Temp).
3. Ważne pliki i katalogi wykorzystywane podczas kompilacji:
 - a. katalog ...TC\Projects – zawiera przykładowy projekt *myMain* oraz plik *blank.c*; służy jako repozytorium rozwiązań poniższych zadań,
 - b. plik ...TC\Projects\blank.c – jest uogólnionym wzorcem prostego programu napisanego w języku C; jego zawartość będzie się powtarzać we wszystkich rozwiązaniach poniższych zadań,


```
#include <stdio.h>
#include <stdlib.h>    // standardowe biblioteki

void main() {          // główna funkcja programu
    system("PAUSE");  // program przed zakończeniem oczekuje
}                      // na naciśnięcie dowolnego klawisza
```
 - c. ...TC\clean.bat – służy do usuwania plików z rozszerzeniami **.bak*, **.obj*, **.exe* z wszystkich podkatalogów katalogu *Project*,
 - d. ...TC\make.bat – służy do kompilowania przygotowanego programu; jako parametr należy podać nazwę projektu reprezentującego przygotowany program.
4. Tworzenie nowego projektu (np.: *myMain*)
 - a. utworzenie nowego podkatalogu o nazwie reprezentowanej przez nazwę projektu (np.: *myMain*) w katalogu ...TC\Projects,
 - b. przekopiowanie do nowo utworzonego katalogu pliku *blank.c* i zmienienie jego nazwy na *nazwa_projektu.c* (np.: *myMain.c*); jeżeli nazwy katalogu i pliku źródłowego będą różne wówczas kompilacja z użyciem *make.bat* zakończy się niepowodzeniem.
 - c. w celu skompilowania napisanego programu należy uruchomić konsolę systemową w katalogu ...TC\; następnie należy wpisać polecenie kompilacji w postaci:
make.bat myMain

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Wersja 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\TC>make.bat myMain

C:\TC>copy Projects\myMain\myMain.c . /y
Liczba skopiowanych plików:      1.

C:\TC>tcc myMain.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
mymain.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

        Available memory 384456

C:\TC>move .\myMain.* .\Projects\myMain\
C:\TC>myMain.c
C:\TC>MYMAIN.EXE
C:\TC>MYMAIN.OBJ

C:\TC>
```

- d. w rezultacie w katalogu projektu pojawiają się pliki z rozszerzeniami **.obj* i **.exe*.

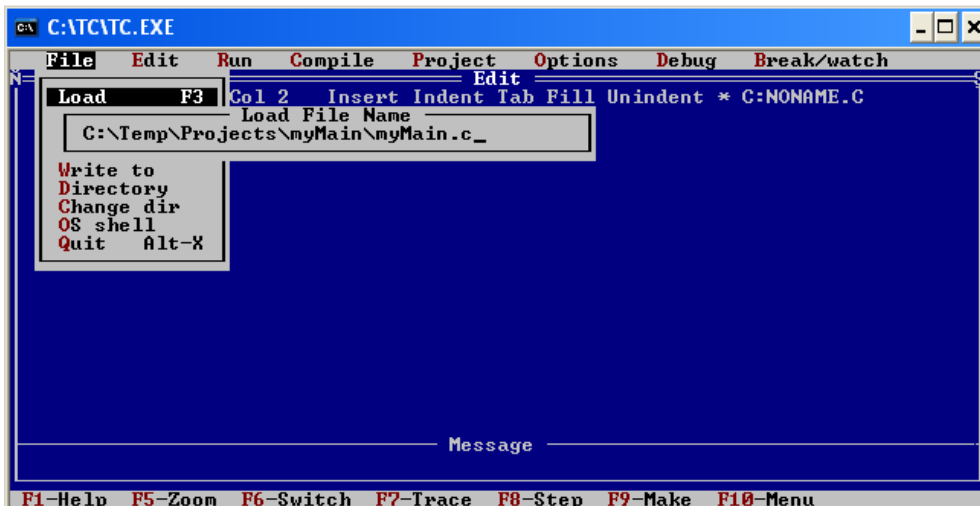
(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

5. Edytor Turbo C

- a. w katalogu ...\\TC znajduje się plik TC.exe, który jest przykładem bardzo prostego edytora pozwalającego na implementowanie programów w języku C. Aby uruchomić program wystarczy dwukrotnie nacisnąć lewym przyciskiem myszy na powyżej przytoczoną nazwę pliku. Po uruchomieniu edytora domyślnie pojawia się okno pozwalające implementować kod w pliku o nazwie NONAME.C. W celu przejścia w tryb edycji pliku należy wykorzystać przycisk *Enter*.
- b. w celu poprawnego kompilowania i uruchamiania programów należy po uruchomieniu edytora ustawić podstawowe opcje środowiska:
 1. ścieżkę dostępu do katalogu zawierającego interfejsy bibliotek (*Include directories*),
 2. ścieżkę dostępu do katalogu zawierającego biblioteki (*Library directories*),
 3. ścieżkę dostępu do katalogu głównego środowiska (*Turbo C directory*).



- c. aby otworzyć nowy (pusty) plik źródłowy należy skorzystać z menu *File->New*.
- d. aby wczytać plik z kodem źródłowym programu należy skorzystać z menu *File->Load* (skrót klawiszowy: F3).



- e. w celu zapisania dokonanych modyfikacji aktualnie otwartego kodu źródłowego programu należy skorzystać z menu *File->Save* (skrót klawiszowy: F2).
- f. aby zbudować plik wykonywalny tworzonego programu należy skorzystać z menu *Compile->Build all*.
- g. w celu uruchomienia pliku wykonywalnego programu należy skorzystać z menu *Run->Run* (skrót klawiszowy: *Ctrl+F9*).
- h. aby zamknąć edytor należy skorzystać z menu *File->Quit* (skrót klawiszowy: *Alt+X*).

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

6. Uwagi

- nazwa projektu nie powinna być dłuższa niż 8 znaków, ponieważ kompilator będzie sygnalizował problem ze znalezieniem właściwego pliku,
- nie wykorzystywać komentarzy postaci „//” w kodzie programu,
- unikąć nazywania projektu i wykorzystywanych funkcji w ciele programu w ten sam sposób,
- nie nazywać własnych funkcji nazwami podstawowych funkcji bibliotecznych,
- w kilku programach należy wykorzystać bibliotekę *math* (poprzez dodanie w kodzie źródłowym deklaracji `#include <math.h>`,
- pamiętać o deklarowaniu wszystkich niestandardowych bibliotek przed ich użyciem w programie.

Wprowadzenie do języka C

1. Podstawowa struktura prostego programu

```
#include <stdio.h>
/* biblioteka standardowa */
main()
/* definicja funkcji main */
{
/* blok instrukcji */
printf("ahoj, przygodo\n");
/* wywołanie funkcji printf */
return 0;
}
Wyjście:
ahoj, przygodo
```

2. Deklarowanie zmiennych

definicje zmiennych (deklaracja + inicjalizacja),

```
typ ident = wartość [, ident = wartość]*;
```

```
int licznik=125, suma=0;
float dokladnosc=0.001, blad=0.1;
double moc=15e6, straty=1500;
```

3. Ważniejsze operatory

+	dodawanie
-	odejmowanie
*	mnożenie, operator dostępu pośredniego (wyluskanie)
/	dzielenie, dzielenie całkowite
%	dzielenie modulo
++	inkrementacja (zwiększenie o 1)
--	dekrementacja (zmniejszenie o 1)
=	przypisanie
==	równe
!=	różne
<	mniejsze
<=	mniejsze lub równe
>	większe
>=	większe lub równe

```
int a, b, c;
a = 1;
b = 2;
c = a+b;
a = b = c = 0; /* wielokrotne podstawienie */
a = b/c; /* część całkowita z dzielenia b przez c */
a = b%c /* reszta z dzielenia b przez c */
a++; /* powiększ a o 1 */
++a; /* powiększ a o 1 */
a=a+1; /* powiększ a o 1 */
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

4. Podstawowe pętle

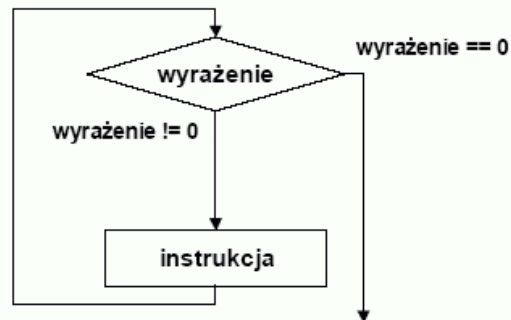
➤ pętla while

```
while (wyrażenie)
    instrukcja;
```

albo:

```
while (wyrażenie)
{
    instrukcja1;
    instrukcja2;
    instrukcja3;
}
```

- warunek jest testowany przed wejściem do pętli
- ciało pętli może nie zostać ani razu wykonane



przykład

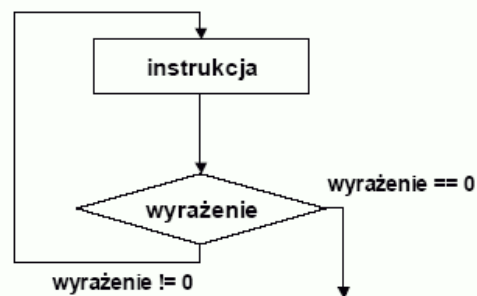
```
int n=10;
while (n>0)
{
    printf ("%d\t", n);
    n--;
}
```

wyjście:

10	9	8	7	6	5	4	3	2	1
----	---	---	---	---	---	---	---	---	---

➤ pętla do-while

- warunek jest testowany po wykonaniu ciała pętli
- ciało pętli musi być choć raz wykonane



przykład:

```
int n=10;
do
{
    printf ("%d\t", n);
    n--;
} while (n>0);
```

wyjście:

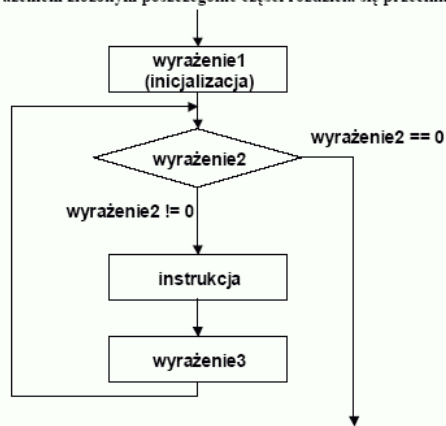
10	9	8	7	6	5	4	3	2	1
----	---	---	---	---	---	---	---	---	---

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

➤ pętla for

for (wyrażenie1;wyrażenie2;wyrażenie3) instrukcja;

- dowolne (wszystkie) wyrażenia można pominąć,
- wyrażenie1 jest częścią inicjującą pętli, jest obliczane tylko raz; jeśli jest wyrażeniem złożonym poszczególne wyrażenia składowe rozdziela się przecinkami,
- wyrażenie2 to warunek kontynuacji pętli, pominięcie go jest równoważne z zastąpieniem stałą różną od zera (warunek zawsze prawdziwy),
- wyrażenie3 jest obliczane po każdym przebiegu pętli i określa zmianę stanu pętli; jeśli jest wyrażeniem złożonym poszczególne części rozdziela się przecinkami.



przykład:

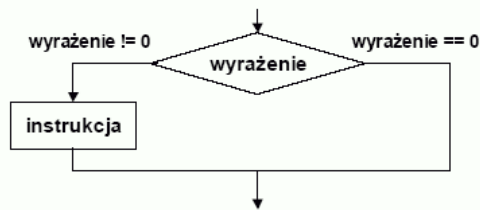
```
int i, j;
for (i=1, j=2;
     j<1000;
     i++, j*=i) printf ("%d,%d\t", i, j);
return 0;
```

wyjście:

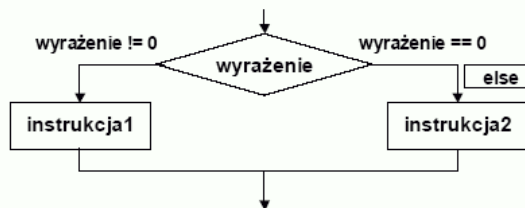
```
[1,2] [2,4] [3,12] [4,48] [5,240]
```

➤ instrukcja warunkowa

if (wyrażenie) instrukcja;



if (wyrażenie) instrukcja;
else instrukcja;



```
if ((n % 2) == 0) printf ("parzysta");
else printf ("nieparzysta");
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

5. Tablice

Tablice

tablica jednowymiarowa:

```
typ id [ rozmiar ];
```

z inicjalizacją

```
typ id [ opcjonalny_rozmiar ] = { lista_stałych };
```

Przykład:

- wektor (tablica jednowymiarowa) liczb całkowitych zawierający 5 elementów

```
int tab[5];  
/* elementy tab[0], tab[1], ..., tab[4] */
```

Przykład:

- 4-elementowy wektor liczb całkowitych, rozmiar określony na podstawie listy inicjalizacji

```
int tab[] = { 1, 2, 3, 5 };  
/* tab[0] = 1, tab[1] = 2, tab[2] = 3, tab[3]=5 */
```

Przykład:

- wektor częściowo zainicjalizowany (list inicjalizacji krótsza niż zadeklarowana liczba elementów)

```
int tab[4] = { 1, 2 };  
/* tab[0] = 1, tab[1] = 2,  
   pozostałe elementy są inicjalizowane zerami*/
```

Odwołania do elementów tablic
nazwa_tablicy[element]

Przykład:

- suma elementów w tablicy
- automatyczne określenie rozmiaru tablicy

```
unsigned u;          /* w języku C indeksy tablic  
                    są zawsze nieujemne */  
int suma;  
int tab[5] = { 1, 2, 2, 3, 4 };  
  
for(u=0, suma=0;  
   u < (sizeof(tab)/sizeof(tab[0]));  
   u++)  
   suma += tab[u];    /* suma=suma+tab[u]; */  
printf("Suma elementow w tablicy = %d", suma);
```

tablice znakowe są specjalnym przypadkiem

reprezentacja łańcuchów

```
"\n\taHello\tworld\n"
```

wewnątrz jest reprezentowany jako ciąg znaków zakończony
wartownikiem ('\0')

```
'\n' '\a' | H | e | l | l | o | '\t' | w | o | r | l | d | '\n' '\0'
```

widok pod debuggerem (Turbo C++ 1.01)

[0]	'\n'	10	(0x0A)
[1]	'\a'	7	(0x07)
[2]	'H'	72	(0x48)
[3]	'e'	101	(0x65)
[4]	'l'	108	(0x6C)
[5]	'l'	108	(0x6C)
[6]	'o'	111	(0x6F)
[7]	'\t'	9	(0x09)
[8]	'w'	119	(0x77)
[9]	'o'	111	(0x6F)
[10]	'r'	114	(0x72)
[11]	'l'	108	(0x6C)
[12]	'd'	100	(0x64)
[13]	'\n'	10	(0x0A)
[14]	'\x0'	0	(0x00)

Przykład:

- 5-elementowa tablica znakowa

```
char tab[5]; /* tab[0], ..., tab[4],  
             brak inicjalizacji */  
char tab[] = "tekst"; /* tab[0]='t', tab[1]='e',  
                      ..., tab[5]='\0' */  
char tab[] = {'t', 'e', 'k', 's', 't', '\0'};  
/* alternatywny sposób inicjalizacji */
```

typowe błędy

```
char tab[5] = "tekst"; /* raczej nie o to chodziło  
                      - brak wartownika */  
char tab[4] = "tekst"; /* błąd kompilacji, za mała  
                      tablica */  
char tab[] = {'t', 'e', 'k', 's', 't'}; /* brak  
                      inicjalizacji elementu tab[5] */
```

Tablice wielowymiarowe

Przykład:

```
int tab2d[4][2]; /* 4 wiersze po 2 elementy */
```

rozmieszczane w pamięci wierszami,

tab2d[0][0]	tab2d[0][1]
tab2d[1][0]	tab2d[1][1]
tab2d[2][0]	tab2d[2][1]
tab2d[3][0]	tab2d[3][1]

inicjalizacja

```
int tab2d[4][2]={{0,1},{2,3},{4,5},{6,7}};  
int tab2d[4][2]={0,1,2,3,4,5,6,7}; /*niezalecane*/
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

6. Wskaźniki

Wskaźniki

wskaźnik to zmienna, która zawiera adres innej zmiennej

Przykłady:

```
int *ptr_2_int; /* wskaźnik na zmienną typu int */
double *ptr_2_double; /* wskaźnik na zmienną typu double */
int *pa, *pb, a, b; /* 2 wskaźniki na zmienną
                    typu int, 2 zmienne typu int */
void *ptr1; /* "wskaźnik do niczego", nie ma
            zmiennych typu void */
```

- pobranie adresu zmiennej – operator &
- uzyskanie dostępu do zmiennej – operator *

```
int i = 5, j = 7, res;
int *pi = &i; /* pi wskazuje na i */
*pi = 3; /* równoważne i=3; */
pi = &j; /* pi wskazuje na j */
*pi = 4; /* równoważne j=4; */
{ /* nowy blok instrukcji */
    int *p1=&i, /* p1 wskazuje na i */
        *p2=&j, /* p2 wskazuje na j */
        *p3=&res; /* p3 wskazuje na res */
    /* zmienne p1, p2, p3 istnieją
       tylko w tym bloku instrukcji */
    *p3 = *p2**p1; /* res = j * i; */
    *p2**p1; /* j *= i; */
} /* zmienne p1, p2, p3 już nie istnieją
   ale wartości w res, j pozostają */
```

- uwaga na klasę register

```
register int b;
int a, *pa;

pa = &a;
/* pa = &b; nie można -> klasa register */
```

- wskaźniki i liczby całkowite nie są ze sobą wymienne (zła praktyka programistyczna)
- jedynym wyjątkiem jest zero (0), stałą zero można przypisać wskaźnikowi, wskaźnik można porównać ze stałą 0, zazwyczaj zamiast zera (0) używa się nazwy symbolicznej NULL (0 w kontekście wskaźnika), implementacja gwarantuje, że żaden wskaźnik wskazujący na zmienną nie może przyjąć wartości NULL

Przykład:

- kopiowanie wskaźników, zgodność typów

```
int aa, *pa=NULL, *pb=NULL;
double *pd;

pa = &aa; /* pa wskazuje na aa */
pb = pa; /* poprawnie, pb i pa wskazują na aa */
pd = pa; /* potencjalny błąd !!!,
         double - 8B w formacie fp,
         int - 4B, liczba całkowita */
*pd = 2.0; /* zamaże 8B zamiast 4B */
pd = &aa; /* potencjalny błąd !!! */
```

Przykład:

- wskaźnik do stałej

```
const int aa;
int *pa;

pa = &aa; /* niebezpieczna konwersja */
*pa = 1; /* błąd !!!,
         przypisanie zostanie wykonane */
```

powinno być:

```
const int aa;
const int *pa;
int b;

pa = &aa; /* prawidłowa konwersja */
/* *pa = 1; <- kompilator zgłosi błąd */
b = *pa; /* poprawne pobranie wartości */
```

7. Tablice a wskaźniki

Tablice a wskaźniki

w języku C występuje ścisła zależność między wskaźnikami i tablicami każdą operację, którą można wykonać indeksując tablicę można wykonać również za pomocą wskaźników

Przykład:

- suma elementów tablicy (za pomocą indeksowania)

```
unsigned u;
int suma, tab[5] = { 1, 2, 2, 3, 4 };
for(u=0, suma=0;
    u < (sizeof(tab)/sizeof(tab[0]));
    u++)
    suma += tab[u];
```

- suma elementów tablicy (z przesuwaniem wskaźnika)

```
unsigned u;
int suma, tab[5] = { 1, 2, 2, 3, 4 }, *pa;
for(u=0, suma=0, pa=tab; /* pa=&tab[0]; */
    u < (sizeof(tab)/sizeof(tab[0]));
    u++)
{
    suma += *pa;
    pa++;
}
```

- suma elementów tablicy (dodawanie do wskaźnika)

```
unsigned u;
int suma, tab[5] = { 1, 2, 2, 3, 4 }, *pa;
for(u=0, suma=0, pa=tab; /* pa=&tab[0]; */
    u < (sizeof(tab)/sizeof(tab[0]));
    u++)
    suma += *(pa+u);
```

- różne operacje na wskaźnikach

```
y = *ip + 1; /* wartość obiektu pod ip, powiększamy
             o 1 i podstawiamy pod y */
*ip+=1; /* zwiększenie obiektu wskazywanego
        przez ip */
(*ip)++; /* j.w. - nawiasy są niezbędne */
```

- jeżeli wskaźniki wskazują na elementy tej samej tablicy to można ze sobą porównywać (==, <, <=, != ...)
- skutek operacji arytmetycznych lub porównań dla wskaźników odwołujących się do elementów różnych tablic jest nieokreślony
- wyjątek to użycie adresu pierwszego miejsca po ostatnim elemencie tablicy

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

8. Formatowane wejście/wyjście

Formatowane wejście

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

- Funkcja czyta z wejścia ciąg znaków i pod kontrolą argumentu format przypisuje przekształcone wartości kolejnym argumentom. Każdy z tych argumentów musi być wskaźnikiem.
- Funkcja kończy działanie, gdy zinterpretuje cały łańcuch sterujący.
- Funkcja zwraca EOF, jeśli przed jakimkolwiek przekształceniem napotka koniec pliku lub nastąpi jakiś błąd, w przeciwnym przypadku zwraca liczbę przekształconych i przypisanych danych wejściowych.
- Format zwykle zawiera specyfikacje przekształceń, które sterują interpretacją wejściowego ciągu znaków. W formacie mogą wystąpić:
 - Odstępy i tabulacje, które są ignorowane
 - Zwykle znaki (nie %) spodziewane jako następne "czarne" znaki w strumieniu wejściowym,
 - Specyfikacje przekształceń złożone ze znaku %, opcjonalnego znaku * wstrzymującego przypisanie, opcjonalnej liczby określającej maksymalny rozmiar pola, opcjonalnej litery h, l lub L określającej rozmiar odpowiedniego argumentu oraz ze znaku przekształcenia.
- Znaki przekształcenia d, i, n, o, u i x można poprzedzić literą h, jeśli argument jest wskaźnikiem do obiektów typu short, a nie int, lub literą l, jeśli argument jest wskaźnikiem do obiektów typu long.
- Znaki przekształcenia e, f i g mogą być poprzedzone literą l, jeśli na liście argumentów znajduje się wskaźnik do obiektów typu double, a nie float, lub literą L dla typu long double.

Znak	Dana wejściowa	Typ argumentu
d	Liczba całkowita dziesiętna	int *
i	Liczba całkowita; albo w postaci ósemkowej (wiodące zero) lub szesnastkowej (wiodące 0x lub 0X)	int *
o	Liczba całkowita ósemkowa (z wiodącym zerem lub bez)	int *
u	Liczba całkowita dziesiętna bez znaku	unsigned int *
x	Liczba całkowita szesnastkowa (z wiodącym 0x lub 0X lub bez)	int *

Formatowane wyjście

```
int printf(const char *format, ...);
```

- Wartość zwracana przez funkcję równa się liczbie wypisanych znaków lub jest liczbą ujemną w przypadku błędu.
- Liczba argumentów musi odpowiadać liczbie w formacie, jeżeli jest mniejsza rezultaty są nieprzewidywalne.
- argument format zawiera znaki dwojakiego rodzaju:
 - zwykle znaki, które są bezpośrednio kopiowane do strumienia wyjściowego *stdout*
 - specyfikacje przekształcenia, które wskazują sposób przekształcenia i wypisania kolejnych argumentów funkcji
- modyfikatory (w dowolnej kolejności), które wpływają na postać wyniku:
 - : dosunięcie przekształconego argumentu do lewego krańca jego pola
 - + : wypisanie liczby zawsze ze znakiem
 - odstęp : poprzedzenie wyniku znakiem odstępu, jeśli jego pierwszym znakiem nie jest plus lub minus
 - 0 - dla przekształceń liczbowych, uzupełnienie liczby wiodącymi zerami do pełnego rozmiaru pola
 - # - alternatywna postać wyniku: dla formatu o pierwszą wypisaną cyfrą będzie 0; dla formatu x lub X wynik różny od 0 poprzedza się 0x lub 0X; dla formatów e, E, f, g i G wypisana liczba zawsze będzie miała kropkę dziesiętną (dla g i G nieznaczące zera nie zostaną usunięte)
- Liczba określająca minimalny rozmiar pola. Jeśli przekształcony argument jest krótszy od rozmiaru pola, to zostanie dopelniony z lewej strony (lub z prawej, jeśli zlecono dosunięcie w lewo) do pełnego rozmiaru pola. Znakiem wypełniającym jest zwykle odstęp, chyba że polecono dopelnianie zerami - wówczas jest nim znak 0.
- Kropka oddzielająca rozmiar pola od precyzji
- Liczba (precyzja) określająca: maksymalną liczbę wypisywanych znaków tekstu; liczbę cyfr wypisywanych po kropce dziesiętnej w specyfikacjach e, E lub f; liczbę cyfr znaczących w specyfikacjach g lub G; minimalną liczbę cyfr wypisywanych dla argumentu całkowitego (dopisanie wiodących zer).
- Modyfikator długości h, l lub L. Litera h wskazuje, że odpowiedni argument jest typu short lub unsigned short; liter l wskazuje, że argument jest typu long lub unsigned long; litera L wskazuje, że argument jest typu long double.

Zamiast liczby określającej rozmiar pola lub precyzję (lub obie) można podać znak *. W tym przypadku żądaną wartość oblicza się na podstawie następnego argumentu (lub następnych argumentów) funkcji. Argument ten musi mieć typ int.

Znak	Dana wejściowa	Typ argumentu
c	Znaki; kolejne znaki zostaną wstawione do wskazanej tablicy. Liczba wpisanych znaków określona jest rozmiarem pola (domyślnie 1). Nie dodaje się znaku '\0'. Nie obowiązuje zwykle pomijanie białych znaków.	char *
s	Ciąg czarnych znaków (bez cudzysłowów). Tablica podana jako argument musi być dostatecznie duża, aby pomieścić cały ciąg razem z dodanym na końcu znakiem '\0'.	char *
e f g	Liczba zmiennopozycyjna. Wejściowy format dla liczb zmiennopozycyjnych zawiera opcjonalny znak, ciąg cyfr ewentualnie rozdzielony kropką dziesiętną oraz opcjonalny wykładnik potęgi, składający się z litery E lub e i liczby całkowitej ewentualnie ze znakiem.	float *
p	Wartość wskaźnika w postaci zależnej od implementacji.	void *
n	Wpisuje do argumentu liczbę dotychczas przeczytanych znaków w tym wywołaniu funkcji. Nie czyta żadnych wejściowych znaków. Nie zwiększa licznika przeczytanych pól.	int *
[...]	Odpowiada najdłuższemu niepustemu ciągowi wejściowych znaków ze zbioru znaków zawartych między nawiasami. Na końcu jest dopisywany znak '\0'. Napis [...] włącza znak] do zbioru.	char *
[^...]	Odpowiada najdłuższemu niepustemu ciągowi wejściowych znaków nie należących do zbioru znaków zawartych między nawiasami. Na końcu jest dopisywany znak '\0'. Napis [^...] włącza znak] do zbioru.	char *
%	Znak %. Nie ma żadnego przypisania.	

Przykład:

```
char tekst[10];
double zmienna;
scanf("%9s", tekst);
/* łańcuch "abrakadabra" zostanie wczytany jako
"abrakadab", łańcuch "Ala ma kota" zostanie
wczytany jako "Ala" (do spacji) */

scanf("%9c", tekst); /* zostanie wczytanych 9 znaków
włącznie ze spacjami */

tekst[9]='\0'; /* ale wartownika trzeba wpisać
samemu */

scanf("%lg", &zmienna);
scanf("%c", &tekst[4]); /* czytamy jeden znak do
tablicy */
```

Znak	Typ argumentu	Przekształcanie do postaci
d, i	int	liczby dziesiętnej ze znakiem
o	int	liczby ósemkowej bez znaku (bez wiodącego zera)
x, X	int	liczby szesnastkowej bez znaku (bez wiodącego 0x lub 0X) z użyciem liter abcdef dla 0x lub ABCDEF dla 0X
u	int	liczby dziesiętnej bez znaku
c	int	pojedynczego znaku po przekształceniu do typu <i>unsigned char</i>
s	char *	Znaki tekstu są wypisywane aż do napotkania znaku '\0' lub liczba wypisanych znaków osiągnie wskazaną precyzję
f	double	liczby dziesiętnej [-]m.dxxxxxx lub [-]m.dxxxxE±xx, gdzie liczba cyfr d zależy od zadanej precyzji. Domyślną precyzją jest 6; przy precyzji 0 opuszcza się kropkę dziesiętną
g, G	double	Jeśli wykładnik potęgi jest mniejszy niż -4 lub większy lub równy precyzji, to stosuje się specyfikację %e lub %E; w przeciwnym przypadku %f. Nie wypisuje się nie znaczących zer i zbędnej kropki dziesiętnej
p	void *	wskaźnika (reprezentacja zależy od implementacji)
n	int *	Liczbę dotychczas wypisanych znaków w tym wywołaniu funkcji printf zapisuje się do odpowiedniego argumentu. Nie ma żadnego przekształcenia argumentu
%		Nie ma żadnego przekształcenia argumentu - zostanie wypisany znak %.

Przykład:

```
char tekst[]="Hello world";
double zmienna = 1.0/3.0;
printf("%+20s\t% f\t%010.3f",
tekst, zmienna, zmienna);
Hello world 0.333333 000000.333
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

9. Podstawowe funkcje matematyczne (biblioteka math.h)

- `double sqrt(double x)` - The *sqrt()* function computes the square root of x , \sqrt{x} .
- `double ceil(double x)` - The *ceil()* function computes the smallest integral value not less than x .
- `double floor(double x)` - The *floor()* function computes the largest integral value not greater than x .
- `double pow(double x, double y)` - The *pow()* function computes the value of x raised to the power y , x^y . If x is negative, y must be an integer value.

10. Powyższe i dodatkowe materiały znajdują się na stronie dr. inż. W. Complaka

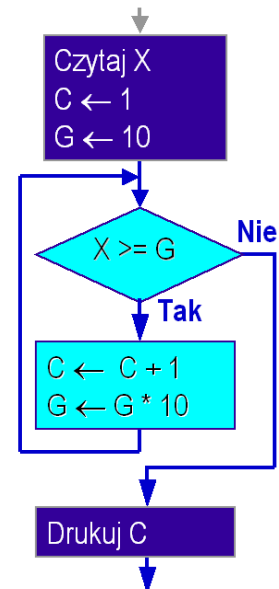
- http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_0.PDF
- http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_1.PDF
- http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_2.PDF
- http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_3.PDF
- http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_4.PDF
- http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_5.PDF

Zadania

Zad. 1(*). Skompiluj i przetestuj program obliczania liczby cyfr dziesiętnych podanej liczby naturalnej n 0, 1, 2, ... (program ten był prezentowany na wykładzie).

We wszystkich poniższych zadaniach:

1. (*) Opracuj zbiór przypadków testowych, które doprecyzują interfejs użytkownika.
2. (*) Narysuj schemat blokowy dla proponowanego rozwiązania.
3. (*) Przelicz (metodą ręcznej symulacji, która była prezentowana na wykładzie) wybrane przypadki testowe na schemacie, by sprawdzić jego poprawność.
4. Napisz program w C.
5. (*) Przelicz na nim wybrane przypadki testowe metodą ręcznej symulacji.



Zad. 2. Napisz program czytania trzech liczb naturalnych i drukowania ich średniej arytmetycznej.

Zad. 3. Napisz program czytania trzech liczb i sprawdzania, czy mogą być one długościami boków jakiegoś trójkąta. **Wskazówka:** $|b-c| < a < b+c$.

Zad. 4. Napisz program obliczania wartości funkcji **$n!$** (silnia) dla n będących liczbami naturalnymi (0, 1, 2, .. 8).

Zad. 5. Napisz program sprawdzania, czy podana liczba jest liczbą automorficzną. **Wskazówka:** Liczba automorficzna to liczba, która występuje na końcu swojego kwadratu np.: 5 bo $5^2 = 25$ lub 25 bo $25^2 = 625$.

Zad. 6(*). Napisz program znajdowania wszystkich liczb automorficznych w podanym przedziale domkniętym $\langle A, B \rangle$.

Zad. 7. Napisz program obliczania sumy cyfr dziesiętnych podanej liczby naturalnej n (np.: suma cyfr liczby 0 jest równa 0, a suma cyfr dziesiętnych liczby 99 jest równa 18).

Zad. 8. Napisz program obliczania wartość funkcji potęgowej a^b dla a, b będących liczbami naturalnymi (0,1, 2,...).

Zad. 9. Napisz program czytania skończonego ciągu liczb całkowitych i drukowania tego ciągu w odwrotnej kolejności (np.: 10 liczb).

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Zad. 10. Napisz program znajdowania **najmniejszej liczby** w skończonym ciągu liczb naturalnych (np.: 10 liczb).

Zad. 11. Napisz program obliczania **sumy liczb** podanego, skończonego ciągu liczb całkowitych ((*rzeczywistych) (np.: 10 liczb).

Zad. 12(*). Napisz program obliczania **wartości średniej** podanego, skończonego ciągu liczb naturalnych (rzeczywistych) (np.: 10 liczb). Należy zmodyfikować odpowiednio rozwiązanie, które zostało zaproponowane dla zadania 11.

Zad. 13. Napisz program czytania rozmiaru n (skończona liczba naturalna) i rysowania macierzy o rozmiarach $n \times n$ typu *lewa-dolna*. Macierz *lewa-dolna* zawiera na głównej przekątnej i pod tą przekątną znaki X, natomiast powyżej głównej przekątnej są znaki spacji. Oto przykład macierzy lewa-dolna dla $n=4$:

```
X
XX
XXX
XXXX
```

Zad. 14(*). Napisz program czytania rozmiaru n i rysowania macierzy o rozmiarach $n \times n$ typu *prawa-górna*. Macierz *prawa-górna* zawiera na głównej przekątnej i nad tą przekątną znaki X, natomiast poniżej głównej przekątnej są znaki spacji. Oto przykład takiej macierzy dla $n=4$:

```
XXXX
 XXX
  XX
   X
```

Zad. 15(*). Napisz program rozwiązywania **równania kwadratowego** $ax^2 + bx + c = 0$. Uwzględnij wszystkie możliwe przypadki wartości współczynników a, b, c .

Zad. 16(*). Napisz program sprawdzania, czy podana liczba naturalna jest **liczbą pierwszą**. **Wskazówka:** wykorzystaj metodę sita Eratostenesa.

Zad. 17(*). Napisz program wyszukiwania **wszystkich liczb pierwszych** w podanym przedziale $\langle A, B \rangle$. Zastosuj funkcję sprawdzającą, czy dana liczba jest liczbą pierwszą z poprzedniego zadania.

Zad. 18. Napisz program obliczania **największego wspólnego podzielnika** (NWP) dwóch liczb naturalnych.

Zad. 19(*). Napisz program obliczania **NWP dla skończonego zbioru liczb** naturalnych (np.: 10 liczb).

Zad. 20. Napisz program obliczania **najmniejszej wspólnej wielokrotności** (NWW) dwóch liczb naturalnych.

Zad. 21(*). Napisz program obliczania **NWW dla skończonego zbioru liczb** naturalnych (np.: 10 liczb).

Zad. 22(*). Napisz program wyszukiwania w podanym przedziale $\langle D, G \rangle$ wszystkich liczb naturalnych a, b , dla których istnieje liczba naturalna c spełniająca warunek $a^2 + b^2 = c^2$ (np.: $3^2 + 4^2 = 5^2$).

Zad. 23. Napisz program wyznaczania n -tego wyrazu **ciągu Fibonacciego**.

Zad. 24(*). Napisz program sortujący ciąg liczb naturalnych metodą przez **wymianę/wybór** (selectionsort).

Zad. 25(*). Napisz program **scalania ciągów**, który polega na łączeniu kilku (co najmniej dwóch) posortowanych ciągów w jeden ciąg posortowany.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Zad. 26(*). Napisz program **mnożenia macierzy kwadratowych**.

Zad. 27(*). Napisz program sprawdzający **współliniowość trzech punktów**.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.