Programowanie strukturalne – język C

Dr inż. Sławomir Samolej
D102 C, tel: 865 1766,
email: ssamolej@prz-rzeszow.pl
WWW: ssamolej.prz-rzeszow.pl

Cechy programowania strukturalnego

- Możliwość wydzielenia w programie bloków, wydzielających grupę instrukcji,
- Możliwość wydzielenia w programie osobnych modułów (funkcji/procedur), które mogą być wielokrotnie wywoływane,
- W języku zdefiniowane są pętle (co eliminuje konieczność definiowania skoków – goto),
- W odróżnieniu od programowania obiektowego – struktury danych są definiowane niezależnie od instrukcji na nich operujących.

Podstawowe elementy języka C

- **Zestaw znaków**
- **Nazwy i słowa zastrzeżone**
- **Typy danych**
- **Stałe**
- **Zmienne i tablice**
- **Deklaracje**
- **Wyrażenia**
- **Instrukcje**

Zestaw znaków języka C

- **Duże litery alfabetu łacińskiego [A..Z]**
- **Małe litery alfabetu łacińskiego [a..z]**
- **Cyfry [0..9]**
- **Znaki specjalne:**
  ! * + \ ” < # (= | { > %) ~ ; } / ^ - [: , ? & _] ‘ oraz znak odstępu (spacja)
- **UWAGA:**
  Nowe narzędzia do tworzenia oprogramowania zezwalają nawet na tworzenie nazw zmiennych i funkcji z zastosowaniem narodowych znaków diakrytycznych. Kod nie będzie wtedy zgodny ze starszymi wersjami standardu ANSI C, co zmniejszy jego przenoszalność.

Nazwy i słowa zastrzeżone

- **NAZWA służy do identyfikowania elementów programu (stałych, zmiennych funkcji, typów danych)**
- **Nazwa składa się z z ciągu liter i cyfr, z zastrzeżeniem, że pierwszym znakiem nazwy musi być litera. Znak podkreślenia _ traktowany jest jako litera**
- **Język C rozróżnia duże i małe litery!**
- **W języku C zdefiniowano tzw. SŁOWA ZASTRZEŻONE (KLUCZOWE), posiadające szczególne znaczenie dla języka. Tych słów nie wolno użyć programiście jako nazw (np. zmiennych, stałych, funkcji itd.)**

Słowa kluczowe języka C

C89:

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

C99:

_Bool	**_Imaginary**	**restrict**
_Complex	**inline**	

Podstawowe typy danych

- **int – reprezentuje liczbę całkowitą**
- **char – reprezentuje małą liczbę całkowitą o rozmiarze wystarczającym do przechowania pojedynczego znaku**
- **float – reprezentuje liczbę rzeczywistą (reprezentowaną w kodzie koprocesora)**
- **double – reprezentuje liczbę rzeczywistą o podwójnej precyzji**

- **Długość danych zależy od implementacji**
- **Wprowadzono również tzw. modyfikatory typów danych:**
  - **short – liczba krótka**
  - **long – liczba długa**
  - **signed – liczba ze znakiem**
  - **unsigned – liczba bez znaku**

  Np.: unsigned int, long int (long), long double…

Stałe (1)

- **Stałe całkowitoliczbowe:**
  - **Stałe dziesiętne (dozwolony zestaw znaków: 0 1 2 3 4 5 6 7 8 9 + -), np.: 0 1 897 -234 +665**
  - **Stałe ósemkowe (dozwolony zestaw znaków: 0 1 2 3 4 5 6 7 + -), Uwaga: pierwszą cyfrą musi być 0, np.: 0 0122 -0777 +0234**
  - **Stałe szesnastkowe (dozwolony zestaw znaków: 0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F + -), Uwaga: pierwszymi znakami muszą być 0x lub 0X, np.: 0x 0xad3 0X233F**
- **Stałe rzeczywiste (dozwolony zestaw znaków: 0 1 2 3 4 5 6 7 8 9 . + - E e), litera E lub e reprezentuje bazę systemu, tj. 10, Uwaga: 1.2×10^{-3} można zapisać 1.2e-3 lub 1.2E-3, np.: 0. 0.2 1.123 13.13E2**
- **Stałe znakowe – pojedyncze znaki „zamknięte” pomiędzy apostrofami: ‘ ‘ Uwaga: Stałe znakowe są w istocie kodami liter i innych znaków zgodnymi z ASCII lub UNICODE np.: ‘A’ ‘#’ ‘ ’(spacja)**

Stałe (2)

- **Escape-sekwencje – kody znaków niedrukowanlych służących do podstawowego formatowania wyjścia znakowego programu lub plików tekstowych,**
  Uwaga: znak sekwencji rozpoznawany jest po tym, że składa się z 2 znaków, w tym pierwszy jest zawsze backslash (\),
  np.: \n \t \" \' \? \\ \0
- **Łańcuchy znaków – stała łańcuchowa (tekstowa) składa się z ciągu o dowolnej liczbie znaków. Ciąg ten mus być ograniczony znakami udzysłowu. Łańcuchy mogą zawierać escape-sekwencje.**
  Np.: "Wynik =" "To jest element \n tekstu"
- **Stałe symboliczne – nazwa zastępująca łańcuch znaków. Do definicji służy pseudoinstrukcja #define**
  np.:
  #define NAZWA text
  #define ROZMIAR_PAMIECI 1024

Zmienne

- **Zmienna to nazwa (identyfikator) reprezentującą określony typ danych.**
- **Deklaracja zmiennej:**
  int a;
  float x1, x2, x3;
  char c = ‘A’;

Instrukcje

- **Instrukcje to te fragmenty programu, które powodują jakąś czynność (akcję) komputera w trakcie wykonywania programu.**
- **Instrukcje można podzielić na cztery grupy:**
  - **Instrukcje obliczające wartość wyrażeń**
    np.: a = 3 + b;
  - **Instrukcje grupujące**
    np.:

```
{ a = 5;
  b = 8;
  pole = a * b;
  printf(”pole=%d”, pole);
}
```

  - **Instrukcje sterujące, np.: while, if…**
  - **Instrukcje wywołania funkcji.**

Funkcje obsługujące wyjście programu (konsola)

- putchar(); - wypisanie pojedynczego znaku
- puts(); - wypisanie pojedynczego tekstu
- printf(); - formatowane wypisywanie tekstów i danych

Przykładowy program: _1_wyjscie.sln

Funkcje obsługujące wejście programu (konsola)

- znak=_getche() – pobranie 1 znaku z konsoli bez potwierdzania „Enter”
- znak=_getchar_nolock(); - pobranie 1 znaku z konsoli, wymagane jest potwierdzenie „Enter” (dawniej: getchar())
- gets(); - pobranie 1 linijki tekstu
- scanf(); - sformatowane pobieranie

Przykładowy program: _2_wejscie.sln

Operator przypisania

W języku C przypisanie wartości zmiennej odbywa się przy pomocy operatora „nadaj wartość”, np.:

```
{       int a;
        a=34; // a nadaj wartość 34
}
```

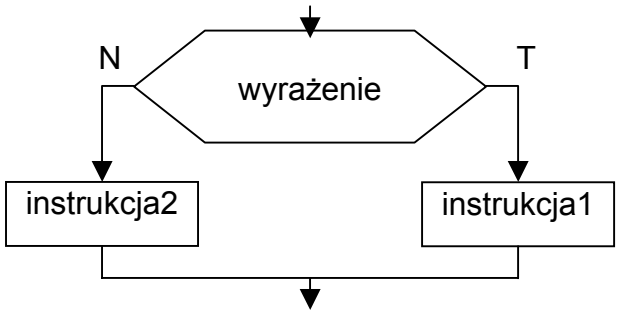
Operatory arytmetyczne

Operator	Operacja
-	odejmowanie
+	dodawanie
*	mnożenie
/	dzielenie
%	reszta z dzielenia

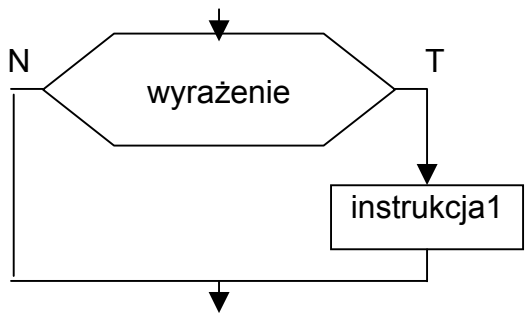
```
// wybrane operacje arytmetyczne:
{
int a, b, c;
      a=5; b=6;
c=a+b; printf("%d\n",c);
c=5%6; printf("%d\n",c);  //reszta z dzielenia
                          //5 przez 6
}
```

Przykładowy program: _3_operat_arytm.sln

Instrukcja warunkowa **if**




```
if (wyrażenie)
instrukcja1
else
instrukcja2
```




```
if (wyrażenie)
instrukcja1
```

```
// Przykład instrukcji if:
{       int a=4, b=5;
        if (a > b)              //jeśli a > b
                printf (“A>B”);       // to pisz: A>B
        else
                printf (“A<=B”);      // w przeciwnym wypadku pisz A<=B
}
```

Operatory porównania

Operator	Operacja
a == b	Czy a jest równe b?
a != b	Czy a jest różne od b?
a > b	Czy a jest większe od b?
a < b	Czy a jest mniejsze od b?
a >= b	Czy a jest większe lub równe od b?
a <=b	Czy a jest mniejsze lub równe od b?

Przykładowy program:

_4_oper_por.sln

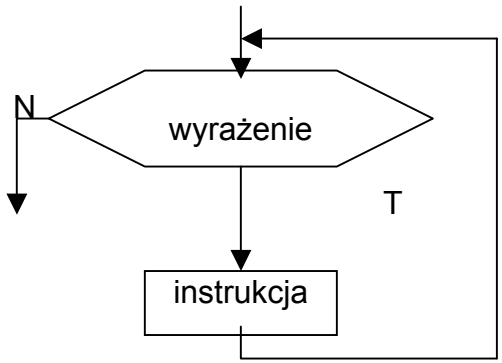
```
{ // Sprawdzenie, czy liczba jest parzysta
        int a,b;
        scanf("%d",&a);
        b=a%2;
        if(b==0) printf("liczba %d jest parzysta",a);
        else     printf("liczba %d jest nieparzysta",a);
}
```

Operatory logiczne

Operator	Operacja
&&	Logiczne „i”
||	Logiczne „lub”
!	Logiczne „nieprawda,że”

```
{ // sprawdzenie czy liczba należy do przedziału:
  float a=28.4;
  if (a>28.0 && a <= 30.0)
            printf("liczba %f nalezy do przedziału (28.0 30.0]\n",a);
  else
            printf("liczba %f NIE nalezy do przedziału (28.0 30.0]\n",a);
}
```

Instrukcja cyklu **while**




```
while(wyrażenie)
instrukcja
```

```
{
 // Wypisanie ciągu: 0,1,..,9
 int i;
 i=0;
 while(i<10)
 {
            printf("%d\n",i);
            i=i+1;
 }
}
```

Przykładowy program:

_5_petla_while.sln

Operatory unarne

++	a++	a = a + 1;	a+=1;
	++a	a = a + 1;	a+=1;
--	a--	a = a - 1;	a-=1;
	--a	a = a - 1;	a-=1;
-	-a	a = -a;	// zmiana znaku znmiennej

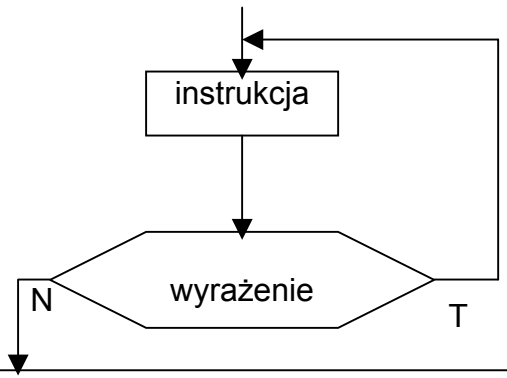
```
{ // Uwaga: operatory ++ i – inaczej pracują przy
  // operatorze przypisania:
  int a = 10, b = 0;
  b = a++;
  printf("a=%d, b=%d\n",a,b);                    // a==11, b==10

  b = 0;
  a = 10;
  b = ++a;
  printf("a=%d, b=%d\n",a,b);}                   // a==11, b==11
```

Priorytety operatorów

1. () [] -> .
2. ! ~ ++ -- -(typ) *(wskaźnikiwy) & (wskaźnikowy) sizeof
3. * / %
4. + -
5. << >>
6. < <= >= >
7. == !=
8. & (bitowe)
9. ^
10. |
11. &&
12. ||
13. ?:
14. = += -= *= /= ...
15. ,

Instrukcja cyklu ***do..while***



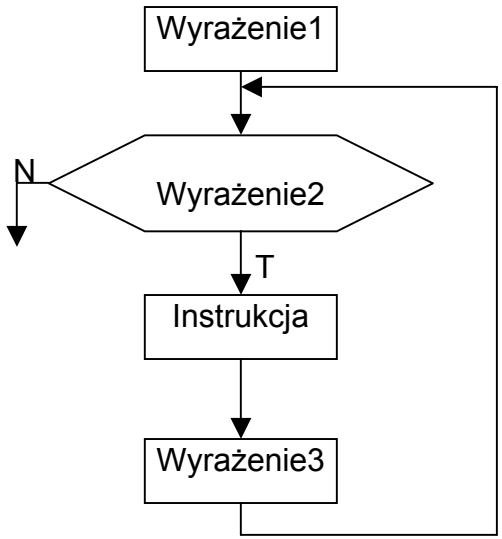

```
do
instrukcja
while(wyrażenie);
```

```
// Wypisanie liczb nieparzystych „normalnie”,
// a liczb parzystych jako pomnożonych przez 2.
#include <stdio.h>
void main(void)
{     char a=10;
      do
      {         a=a-1;
                if(a%2==1)
                          printf("a=%d\t",a);
                else
                          printf("a=%d\t",2*a);
      }
      while(a>0);
}
```

Przykładowy program:

7_petla_do_while.sln

Instrukcja cyklu ***for***




```
Wyrażenie1
while (Wyrażenie2)
{
        Instrukcja
        Wyrażenie3
}
```

for(Wyrażenie1;Wyrażenie2;Wyrażenie3)
Instrukcja

```
{ int i;
  for(i=0;i<10;i++)
 { printf("%d ",i);  }
}
```

0 1 2 3 4 5 6 7 8 9

```
{ int i,j;
  for(i=0,j=10;i<10;i++,j--)
  { printf("%d %d | ",i,j); }
}
```

0 10 | 1 9 | 2 8 | 3 7 | 4 6 | 5 5 | 6 4 | 7 3 | 8 2 | 9 1 |

Instrukcja ***break*** | instrukcja ***continue***

```
// Przerwanie odczytu ciągu
// znaków, jeśli napotka się
// spację lub tabulację:
{ char c;
  do
  { c=getchar();
    if(c==' '||c=='\t') break;
    putchar(c);
  }
  while(c!='\n');
}
```

```
// sumowanie tylko dodatnich
// elementów ciągu:
{ int a, suma=0;
  do
  { scanf("%d",&a);
    if(a<0) continue;
    suma+=a;
  }
  while(a!=0);
  printf("suma=%d\n",suma);
}
```

Instrukcja ***switch***

```
// Zliczenie ilości wystąpień a, b, c
// w tekście:
{ char c;
   int la, lb, lc;
   la=lb=lc=0;
   do
   { c=getchar();
     if(c=='a') la++;
     else
            if(c=='b') lb++;
            else
                        if(c=='c') lc++;
    }while(c!='\n');
   printf("la=%d, lb=%d, lc=%d",la,lb,lc);
}
```

```
// Zliczenie ilości wystąpień a, b, c
// w tekście:
{ char c;
   int la, lb, lc;
   la=lb=lc=0;
   do
   { c=getchar();
     switch(c)
     { case 'a': la++; break;
       case 'b': lb++; break;
       case 'c': lc++; break;
       default : break;
      }
    }while(c!='\n');
  printf("la=%d, lb=%d, lc=%d",la,lb,lc);
}
```

Funkcje - wprowadzenie

- Pojęcie funkcji wprowadzono w języku C w celu umożliwienia tworzenia podprogramów – fragmentów programów, które mają zdefiniowany interfejs z otoczeniem i mogą być wykorzystywane wielokrotnie w obrębie danego programu lub stosowane w wielu pisanych programach.
- Typowy program w języku C jest zestawem definicji funkcji oraz sposobu ich wywoływania.
- Programy w języku C tworzy się przez włączenie potrzebnych bibliotek, a następnie uzupełnienie brakującej funkcjonalności.
- Biblioteki funkcji są też często sprzedawane lub udostępniane w celu możliwości rozwijania danej klasy programów.
- Podstawową umiejętnością projektanta oprogramowania jest wydzielenie zadań, które mają być wykonywane przez osobne podprogramy. Umożliwia to rozdzielenie prac na zespół programistów.
- Często zadania wyznaczone dla pojedynczej osoby dzieli się na jeszcze mniejsze podprogramy. Zapewnia to większą czytelność.

Funkcje - definiowanie

```
#include <stdio.h>

//Lista prototypów funkcji:
int kukulka(int ile);

//Główna funakcja programu: main:
void main(void)
{           int m = 20;
            printf("Zaczynamy\n");
            m=kukulka(5);
            printf("\n Na koniec m = %d\n",m);
}
int kukulka(int ile)
{           int i;
            for(i=0;i<ile;i++)
            { printf("Ku-ku !"); }
            return 77;
}
```

- Program w języku C zawsze rozpoczyna wykonywanie od funkcji main

- Funkcja identyfikowana jest przez nazwę, wywołanie funkcji, to napisanie nazwy.
- Wywołanie funkcji powoduje rozpoczęci jej wykonywania – przeskok do fragmentu programu zapisanego w funkcji.

- Funkcja może przyjąć listę argumentów

- Funkcja może zwrócić 1 wartość

Funkcje - uwagi

- Argument funkcji (parametr wywołania funkcji) jest standardowo przekazywany przez wartość, czyli do funkcji trafia LICZBA, a nie miejsce w pamięci, gdzie się ona znajdowała.
- Argumenty funkcji przekazywane przez wartość mogą być wewnątrz niej traktowane jak zmienne lokalne. Zmiana takiej zmiennej nie spowoduje zmiany wartości „zewnętrznej” komórki pamięci.
- Istnieje możliwość innego przekazywania parametrów do funkcji – przez wartość; zostanie omówiona ona dalej.
- Obliczona wartość zwracana przez funkcję jest przekazywana do wyznaczonej zmiennej, która będzie ją przechwytywać.
- Wnętrze funkcji jest zwyczajnym blokiem programu {…}, w którym można definiować zmienne, tworzyć wyrażenia, wywoływać inne funkcje.
- Aby funkcja mogła być wywołana, jej nazwa przed wywołaniem musi być programowi znana:
  - Albo jest gdzieś podany prototyp funkcji;
  - Albo prototyp funkcji jest w pliku nagłówkowym (xxx.h).

- Przykładowy program: 9_funkcje2.sln

Tablice jednowymiarowe

- Tablica jest ciągiem identycznych elementów znajdujących się jeden za drugim w pamięci. Podstawowe informacje o tablicy to jej długość i typ danych jej elementów, np.:

  int tab1[30]; // 30-elementowa tablica elementów typu int

  char tekst[10]; // 10-elementowa tablica elementów typu char (tablica //tekstowa)

- Odwołać się do elementu tablicy można przez indeks do tego elementu np.:

  int tab2[4]={2,5,6,7};

  int a, b;

  a=tab2[0]; // zmiennej a przypisz zawartość pierwszego elementu tablicy tab2

  b=tab2[3]; // zmiennej b przypisz zawartość ostatniego elementu tablicy tab2

  tab2[2]=45; // elementowi tablicy o indeksie 2 nadaj wartość 45

- Uwaga: **Indeksowanie tablicy odbywa się zawsze od 0!** Ostatni element ma indeks równy (rozmiar tablicy) –1!
- Przykładowy program: _10_tab1.sln

Tablice tekstowe

- Tablice o elementach typu char mogą przechowywać teksty. Tekst w języku C jest ciągiem znaków (typu char) zakończonych liczbą 0. Liczbę 0 w tablicach tekstowych często koduje się przy pomocy specjalnego znaku: '\0'. Jeśli chcemy zachować w tablicy o elementach typu char pewien tekst, możemy zainicjalizować tablicę w pamięci, a następnie przypisać jej stałą tekstową, np.:

  char tekst1[100]="To jest tekst";

- W pamięci w poszczególnych elementach tablicy zapisane zostaną kolejne znaki tekstu, a na koniec '\0':

  tekst1[0]=='T', tekst1[1]=='o', tekst1[2]==' ',..., tekst1[12]='t', tekst1[13]=='\0',

- Przy inicjalizacji tablicy należy zwrócić uwagę na rozmiar tablicy. Rozmiar powinien być wystarczający do przechowania tekstu. W przykładowej tablicy tekst1 o rozmiarze 100 elementów, zapamiętany został tekst o długości 13 znaków (w długości tekstu nie uwzględnia się znaku '\0' na końcu każdego tekstu). Podczas przetwarzania tekstów z reguły nie uwzględnia się faktycznej długości tablicy, a raczej długość tekstu w niej zawartego.
- Przykładowy program: _11_tab_txt.sln

Tablice wielowymiarowe

- Tablicę 2-wymiarową można przedstawić jako pewien zestaw tablic 1-wymiarowych np.:

  float tab1[3][4];

  tab1 można interpretować jako tablicę 3-elementową złożoną z tablic 4-elementowych.
- Tablicę 3-wymiarową można traktować jako zestaw tablic 2-wymiarowych itd.
- W przypadku tablicy 2-wymiarowej indeksy można traktować jako numer wiersza i numer kolumny.
- Dostęp do elementów tablicy uzyskuje się najprościej przez podanie odpowiedniego zestawu indeksów np.:

```
int a[3][2]=    {       {2,3},
                        {4,8},
                        {1,4}
                };
int c;
a[0][0]=0;      // elementowi o indeksie (0,0) nadaj wartość 0;
c=a[0][1];      // zmiennej c przypisz zawartość elementu tablicy a o indeksach
                // (0,1)
```

- Przykładowy program: _12_tab_2n.sln

Funkcje i tablice liczbowe

```
#include <stdio.h>
int max_el(int t[], int roz_tab); // Jesli przekazyje sie do funkcji
                                  // tablice, to trzeba podac jej rozmiar osobno.

void main(void)
{       int tab[3]={12,445,23};
        int max;
        max=max_el(tab,3);
}

int max_el(int t[],int roz_tab)
{       int m_el=t[0],i;
        for(i=1;i<roz_tab;i++)
                if(t[i]>m_el) m_el=t[i];
        return m_el;
}
```

Funkcje i tablice tekstowe

```
#include <stdio.h>
int str_len(char text[]);

void main(void)
{           char t1[]="to jest tekst";
            int len;
            len=str_len(t1);
            puts("Tekst:");
            puts(t1);
            printf("ma dlugosc: %d\n",len);
}

int  str_len(char text[])
{           int i=0;
            while(text[i]!=0)
            { i++;
            }
            return i;
}
```

```
Tekst:
to jest tekst
ma dlugosc: 13
```

Wskaźniki - wprowadzenie

- Wskaźniki to zmienne, które zawierają adres innej zmiennej. Do utworzenia zmiennej wskaźnikowej służy operator „*”, np.:

```
char *wsk1;          // wskaźnik na zmienną typu char
int *ptr1;           // wskaźnik na zmienną typu int
float *f_ptr;        // wskaźnik na zmienną typu float
```

- Do przekazania wskaźnikowi adresu pewnej zmiennej służy operator „&”, np.:

```
int* i_ptr;
int a;
a=10;
i_ptr=&a;            //wskaźnikowi i_ptr przypisz adres zmiennej a
```

- Aby uzyskać wartość wskazywanej przez wskaźnik zmiennej należy posłużyć się operatorem „*”, np.:

```
char c1='a',c2;
char *c_ptr;
c_ptr=&c1;           //wskaźnikowi c_ptr przypisz adres zmiennej c1
c2=*c_ptr;           //do zmiennej c2 wpisz wartość pokazywaną przez wskaźnik c_ptr
```

- Podobnie, mając wskaźnik na pewną zmienną można dokonać zmiany zawartości tej zmiennej, np.:

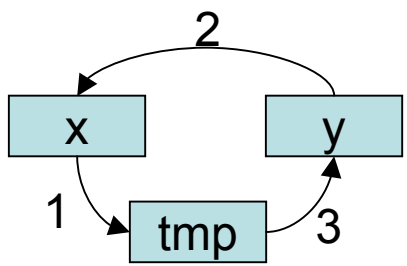
```
float dana1=2.3;
float *f_ptr=&dana1;  //wskaźnikowi f_ptr przypisz adres zmiennej dana1
*f_ptr=3.8;           //w miejsce wskazywane przez wskaźnik wpisz 3.8 (oznacza to
                      //, że zmodyfikowano stan zmiennej dana1)
```

- Przykładowy program: _13_wsk1.sln

Wskaźniki jako argumenty funkcji

- W języku C można przekazywać parametry do funkcji na dwa sposoby: przez wartość i przez wskazanie na zmienną. Gdy parametry są przekazywane przez wartość, to po zakończeniu działania funkcji stan zmiennych, które posłużyły do przekazania informacji o swoim stanie nigdy nie ulega zmianie. Przekazanie parametru przez wskazanie na zmienną (wskaźnik) umożliwia dokonania modyfikacji tej zmiennej wewnątrz funkcji.

```
#include <stdio.h>
int a=4,b=8;
int c=3,d=7;
void swap1(int x, int y);
void swap2(int *x, int *y);
void main(void)
{printf("Zmienne a, b przed wykonaniem funkcji swap1: %d, %d\n",a,b);
 printf("Zmienne c, d przed wykonaniem funkcji swap2: %d, %d\n",c,d);
 swap1(a,b);          // przekazanie parametrow przez wartosc
 swap2(&c,&d);        // przekazanie parametrow przez wskazanie
                      // na zmienne
printf("Zmienne a, b po wykonaniu funkcji swap1: %d, %d\n",a,b);
printf("Zmienne c, d po wykonaniu funkcji swap2: %d, %d\n",c,d);
}
```




```
void
swap1(int x, int y)
{ int tmp;
 tmp=x;
 x=y;
 y=tmp;}
void
swap2(int *x, int *y)
{ int tmp;
  tmp=*x;
  *x=*y;
  *y=tmp;}
```

Struktury I

- Struktura jest obiektem złożonym z jednej lub kilku zmiennych, być może różnych typów.
- Struktury można deklarować w następujący sposób:

a) przez zastosowanie słowa kluczowego „struct”:

```
struct point                    // deklaracja struktury
{
        int x;
        int y;
};
struct point pt;                // utworzenie obiektu typu struct point
```

b) przez zastosowanie słowa „struct” i metody definiowania nowych typów danych przy pomocy słowa kluczowego „typedef”:

```
typedef struct point                    // deklaracja struktury
{
        int x;
        int y;
} PUNKT;
PUNKT pt;                       // utworzenie obiektu typu PUNKT
```

Struktury II

- Elementom struktury można przypisać początkowe wartości w momencie inicjalizacji:

```
struct point                                    // deklaracja struktury
{       int x;
int y;
};
struct point pt={10,20};  // utworzenie i inicjalizacja obiektu typu struct point
```

- Poza inicjalizacją do pól struktury można odwołać się przy pomocy operatora „.”:

```
struct point                                    // deklaracja struktury
{       int x;
int y;
};
struct point pt;                                // utworzenie obiektu typu struct point
pt.x=10;                                        // nadanie wartości 10 polu x struktury
pt.
pt.y=20;
```

- Dopuszczalne jest zagnieżdżanie struktur, np.:

```
struct point {int x; int y};
struct rect{struct point pt1; struct point pt2};
```

Struktury III

- Aby uzyskać dostęp do odpowiednich pól danych należy posłużyć się kilkakrotnie operatorem ".":

```
struct point {int x; int y};
struct rect{struct point pt1; struct point pt2};
struct rect screen;
screen.pt1.x=5;
```

- Można zdefiniować wskaźnik na strukturę i przy jego pomocy odwoływać się do elementów struktury:

```
struct point {int x; int y}; // definicja struktury
struct point *point_ptr;   // definicja wskaźnika
struct point p1;                    // utworzenie obiektu typu "struct point"
int a,b;
point_ptr=&p1;                      // przypisanie wskaźnikowi "point_ptr" adresu struktury „p1"
point_ptr->x=12;                    // zastosowanie operatora "->" do uzyskania dostępu do pola
                                    // struktury
b= p1.y;
a= point_ptr->y;
```

Struktury IV

- W języku C dopuszczalne jest tworzenie funkcji działających w następujący sposób na strukturach:

a) funkcja może zwracać strukturę w całości np.:

```
struct point makepoint(int x, int y);
```

b) struktura może być argumentem wywołania funkcji:

```
struct point addpoint(struct point p1, struct point p2);
```

c) argumentem wywołania funkcji może być wskaźnik na strukturę:

```
struct point addpoint(struct point *p1_ptr, struct point *p2_ptr);
```

- Możliwe jest tworzenie tablic struktur:

```
struct klucz    {       char litera;
                        int licznik;
};
struct klucz statystyka[]=
{       {'a',0},
        {'b',0},
        {'c',0}
};
```

- Przykładowy program: _16_struct.sln

Podstawowe typy zmiennych

- W języku C są 2 podstawowe typy zmiennych:
  - Zmienne automatyczne – definiowane na początku bloku – są lokalne w danym bloku i są niszczone po wyjściu z bloku.
  - Zmienne zewnętrzne/globalne – definiowane poza blokami i funkcjami – są one widoczne dla wszystkich bloków, mogą służyć do komunikacji pomiędzy funkcjami.

```
#include <stdio.h>
int globalna = 12;
void zmien_globalna(void);

void main(void)
{int lokalna =3;
 globalna=lokalna;
  printf("%d\n",globalna);
  zmien_globalna();
  printf("%d\n",globalna);
}
void zmien_globalna(void)
{               globalna = 5;}
```

```
3
5
```

Operacje na plikach I

- Przetwarzanie plików w języku C odbywa się zawsze w trzech etapach:
  - otwarcie pliku (funkcja fopen())
  - zapis lub odczyt do/z pliku(funkcje fgetc(), fputc(), fscanf(), fprintf())
  - zamknięcie pliku (funkcja fclose()).
- Komunikacja z plikiem odbywa się przez wskaźnik do "uchwytu do pliku":
  FILE* plik;
- Pełna postać deklaracji funkcji fopen ma postać:
  FILE* fopen(char* name, char *mode);
  Parametr name powinien zawierać ścieżkę dostępu do pliku, który chcemy otworzyć. Parametr mode powinien zawierać tryb, w jakim plik ma być otwarty. Wybrane tryby otwarcia pliku mają postać:
  "r" - plik będzie otwarty do czytania;
  "w" - plik będzie otwarty do zapisu, jeśli plik o podanej nazwie istnieje, to zostanie usunięty z dysku, a następnie ponownie utworzony;
  "a" - plik jest otwarty do dopisywania na końcu.
- Funkcja fopen w przypadku pomyślnego nawiązania komunikacji z plikiem zwraca wskaźnik na obiekt typu FILE. W przypadku braku pliku funkcja zwraca NULL (0).

Operacje na plikach II

- Funkcje umożliwiające zapis lub odczyt zawartości pliku korzystają z uzyskanego wskaźnika na obiekt typu FILE.
- Przykładowo odczyt pojedynczego znaku z pliku może się odbywać następująco:

```
FILE *plik;
char c;
plik=fopen("\\autoexec.bat","r");
c=fgetc(plik);
```

- Zapis pojedynczego znaku do pliku może odbywać się następująco:

```
FILE *plik;
plik=fopen("\\tekst1.txt","w");
fputc('a',plik);
```

Operacje na plikach III

- Możliwy jest również zapis i odczyt danych tekstowych w sposób sformatowany:

```
// odczyt z pliku:
FILE *plik;
float dana;
plik=fopen("\\dane.txt","r");
fscanf(plik,"%f",&dana);
//zapis do pliku:
FILE *plik;
float dana=3.4;
plik=fopen("\\dane.txt","w");
fprintf(plik,"%f",dana);
```

- Przed zakończeniem działania programu, w którym odbywała się komunikacja z plikami dyskowymi należy zamknąć otwarte uprzednio pliki. Służy do tego funkcja:

```
fclose(plik);.
```