struktura

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

struct complex {double real; double img;
} alfa;
struct complex beta={2.0,2.0}, delta;
struct complex multi (struct complex x, struct complex y);

int  main()
{
alfa.real=0.5; alfa.img=0.5;
beta.real=0.5; beta.img=1.0;
delta=multi(alfa,beta);
```

struktura

```
printf("\n gamma=%6.3f,%6.3f\
\n alfa=%6.3f,%6.3f  beta=%6.3f,%6.3f\n",
  delta.real, delta.img, alfa.real, alfa.img, beta.real,
beta.img);
exit(0);
} /*  koniec funkcji main  */
```

struktura

```
struct complex multi( struct complex a, struct complex b)
{
 struct complex c;
 c.real=a.real*b.real -a.img*b.img;
 c.img=a.real*b.img+a.img*b.real;
 a.real=4.0; a.img=4.0;
 b.real=5.0; b.img=5.0;
 return(c);
}/*  koniec funkcji multi */
   /*  wynik programu to
    gamma=-0.250, 0.750
    alfa= 0.500, 0.500  beta= 0.500, 1.000
   */
```

dyrektywy preprocesora

#define identyfikator znacznik

(definiowanie stałych symbolicznych)

#define MAXL 512

```
for( i=1; i<MAXL; i++)
```

#define identyfikator (identyfikator…)

(ta postać dla definiowania makrodefinicji)

dyrektywy preprocesora

#ifdefined identyfikator

#ifdefined (identyfikator)

#ifdef identyfikator

(te postacie są równoważne i dają w wyniku wartość niezerową, jeśli identyfikator był poprzednio zdefiniowany przez #define)

dyrektywy preprocesora

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

#ifdef ALFA
#define MAXL 100
#else
#define MAXL 200
#endif

int  main()
{
  printf("\n MAXL=%d\n",MAXL);
}  /* koniec funkcji main  */
```

dyrektywy preprocesora

gcc –DALFA program.c

dyrektywy preprocesora

#undef identyfikator

Ta dyrektywa powoduje, że identyfikator określony wcześniej za pomocą dyrektywy #define jest w dalszej części tekstu programu niezdefiniowany

dyrektywy preprocesora

#else

#endif

#ifdef ….

…

#else

…

#endif

dyrektywy preprocesora

#ifndef identyfikator

Powyższe wyrażenie jest prawdziwe, jeśli identyfikator nie jest zdefiniowany w pliku źródłowym

```
#ifndef DEBUG
printf_table();
#endif
```

dyrektywy preprocesora

```
#ifndef BETA

…

#else

…

#endif
```

dyrektywy preprocesora

#if wyrazenie_calkowitoliczbowe

jesli jest niezerowe, to jest prawdziwe

#if TRUE

dyrektywy preprocesora

#include “nazwa_pliku”

(z katalogu bieżącego)

#include <nazwa_pliku>

(z katalogu standartowego, np /usr/include)

dyrektywy preprocesora

#line **stała** nazwa_pliku

Ta dyrektywa powoduje, ze kompilator przechodzi do wiersza o numerze określonym przez **stałą** w pliku o podanej nazwie.

#line 20 ”nowyplik.c”

dyrektywy preprocesora

Definicja makra może zawierać specjalny operator #; powoduje on zamianę parametru makra w łańcuch znakowy (czyli string); przykład jest poniżej.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  float x=6.,y=2.;
  #define  Wydruk(z)  printf("    " #z " = %f\n",z);
  Wydruk(x/y);
  exit(0);
} /*  koniec  funkcji  main  */
/* wydrukuje:    x/y = 3.000000 */
```

dyrektywy preprocesora

Definicja makra może zawierać specjalny operator ##; służący do „sklejania”; przykład umieszczono poniżej.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
#define  Makro(n)   f_##n
  float  Makro(x); /*  float f_x  */
  float  Makro(y); /*  float f_y  */
  f_x=7.0 ;    f_y=9.0 ;
  printf("%f %f\n",  f_x,  f_y);
  exit(0);
} /*  koniec  funkcji  main  */
```

pliki źródłowe

Program w C przygotowuje się w postaci pliku (zbioru) tekstowego, który jest następnie kompilowany do postaci pośredniej. Pliki pośrednie podlegają konsolidacji (linkowaniu) z innymi plikami tego samego typu i z bibliotekami → program wykonywalny.

kompilacja	+	konsolidacja
(kompilator)		(linker,konsolidator)

Przechowywanie programu podzielonego na niewielkie pliki ogranicza czas kompilacji, pliki nie zmienione nie musza być powtórnie kompilowane, małe pliki łatwiej redagować.

adres zmiennej

Do pobrania adresu zmiennej używa się jednoargumentowego operatora & (uwaga & może mieć także znaczenie dwuargumentowego operatora bitowego iloczynu logicznego)

Jednoargumentowy operator * jest "używany do wskazywania", tzn. jego argument jest adresem zmiennej.

(jednoargumentowy operator * nazywa się również operatorem dereferencji)

adres zmiennej - przykład

```
#include <stdio.h>
#include <stdlib.h>
int  main()
{
        int k;
        int n;
        int *palec;/* ta zmienna jest wskaznikiem */
        palec=&k;
        k=10;
        n=(*palec)*15;
        printf("\n\nk=%5d  n=%5d  *palec=%5d\n",k,n,*palec);
        exit(0);
}  /* koniec funkcji main  */
```

```
              k=   10  n=  150  *palec=   10
```

operatory bitowe

umożliwiają korzystanie, wykonywanie działań na pojedynczych bitach; zaś np. operator dodawania działa na całej zmiennej!

& koniunkcja bitowa

| alternatywa bitowa

^ różnica symetryczna

~ negacja bitowa

<< operator przesunięcia w lewo

>> operator przesunięcia w prawo

operatory bitowe - &

bit 1	bit 2	bit1 & bit2
0	0	0
1	0	0
0	1	0
1	1	1

```
int c1,c2,c3;
c1=0x45; c2=0x71;
c3=c1&c2;  /*  c3 jest równe  0x41 = 65 dziesiętnie */
```

operatory bitowe - |

bit 1	bit 2	bit1 | bit2
0	0	0
1	0	1
0	1	1
1	1	1

```
int c1,c2,c3;
c1=0x47; c2=0x53;
c3=c1|c2;  /*  c3 jest równe  0x57= 87 dziesiętnie */
```

operatory bitowe - ^

bit 1	bit 2	bit1 ^ bit2
0	0	0
1	0	1
0	1	1
1	1	0

```
int c1,c2,c3;
c1=0x47; c2=0x53;
c3=c1 ^ c2;  /*  c3 jest równe  0x14= 20 dziesiętnie */
```

operatory bitowe - ~

bit	~bit
0	1
1	0

```
int c1,c3;
c1=0x45;
c3=~c1;  /* c3 jest równe  0xFFFFFFBA = -70 dziesiętnie
*/
```

operatory bitowe - <<

przesuwa w lewo o określona ilość bitów

uwaga – przesunięte bity znikają, a nie pojawiają się z prawej strony

c	0x1C	00011100
c<<1	0x38	00111000
c>>2	0x07	00000111

operatory bitowe - >>

jest w nim pewna pułapka, mianowicie miejsce wolne jest zastępowane bitem znaku!

np. int k=-1;

```
int l;

l=k>>1;    /*   l jest równe minus jeden !   */
```

Jak bit ustawic?

na przykład chcemy ustawić trzeci bit na “jeden”

```
int k=1<<3;       /*  k = 8 dziesiętnie  */
int n;
n|=k;                          n=n|k;
```

Jak bit testowac?

na przykład chcemy sprawdzić trzeci bit

```
int k=1<<3;       /*  k = 8 dziesiętnie  */
int n;
  if( (n&k)!=0 )   printf(“ trzeci bit == 1 ”);
  else
  printf(“ trzeci bit == 0 “);
```

Jak bit usunac?

na przykład chcemy ustawić trzeci bit na “zero”

```
int k=1<<3;       /*  k = 8 dziesiętnie  */
int n;
n&=~k;                          n=n & (~k);
```

pola bitowe

```
int  main()
{
        struct aka{
                        int k:3;
                        int n:3;
                    } beta;
        beta.n=1;
        while(1)
        {       sleep(1);
                printf("\nbeta.n=%d",beta.n);
                beta.n+=1;   /* beta.n=beta.n+1; */
        }
        exit(0);
}  /* koniec funkcji main  */
```

wynik programu

beta.n = 1

beta.n = 2

beta.n = 3

beta.n = -4

beta.n = -3

beta.n = -2

beta.n = -1

beta.n = 0

beta.n = 1

extern

```
#include <stdio.h>
#include <stdlib.h>
extern float alfa;
                                                    /*     plik extern1.c     */
int  main()
{
printf("\n alfa=%10.4f\n",alfa);
return(0);
}  /* koniec funkcji main  */
```

```
                    float alfa=-66;                 /*     plik extern1a.c     */
```

```
        /*    gcc  -Wall  -ansi  -pedantic  extern1.c extern1a.c   */
```

extern

```
#include <stdio.h>
#include <stdlib.h>
extern float alfa;
float alfa=4;                    /*   jeden plik zrodlowy   */

int  main()
{
printf("\n alfa=%10.4f\n",alfa);
return(0);
}  /* koniec funkcji main  */
```