STRUKTURY

Przykład informacji związanych z typowym adresem pocztowym:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
struct ABONENT {
   char nazwisko [25];
   char imie [15];
   char tytul [25];
   char ulica [60];
   char miasto [30];
   char wojewodztwo [20];
   char kod [6];
   float oplata;
   int termin;
};
struct ABONENT przesylka ;
void main()
{  void odcz_inf_podst();
   odcz_inf_podst();
   printf("Nazwisko = %s %s\n",
           przesylka.imie,przesylka.nazwisko);
   printf("Oplata = %05.1f\n",przesylka.oplata);
   printf("Termin realizacji= %d\n",przesylka.termin);
}
void odcz_inf_podst()
{  strcpy (przesylka.nazwisko , "Stanislawski" );
   strcpy (przesylka.imie , "Piotr" );
   strcpy (przesylka.tytul , "mgr" );
   strcpy (przesylka.ulica , "Krucza" );
   strcpy (przesylka.miasto , "Strzelin" );
   strcpy (przesylka.wojewodztwo , "wroclawskie" );
   strcpy (przesylka.kod , "57-100" );
   przesylka.oplata = 10000;
   przesylka.termin = 2;
}
```

Deklaracja struktury danych

```
  struct etykieta-struktury {
  {
    deklaracje skladowych
  } nazwa-struktury;
```

Przykład (współrzędne punktu na płaszczyźnie):

```
struct punkt {
  int x;
  int y;
} ;
```

Powyższa deklaracja nie rezerwuje żadnej pamięci, opisuje tylko wzorzec struktury.

Przykład (użycie wzorca struktury):

```
struct punkt pkt;
```

Równoważna deklaracja struktury danych we wzorcu struktury:

```
struct punkt {
  int x;
  int y;
} pkt;
```

Inicjalizacja struktury:

```
struct punkt pkt = {10, 20};
```

Strukturę automatyczną można także zainicjować za pomocą przypisania lub wywołania funkcji, która zwraca strukturę właściwego typu.

Odwołania do składowych struktury:

```
nazwa-struktury.skladowa

Np.      pkt.x
   printf("%d, %d", pkt.x, pkt.y);
```

Zagnieżdżenie struktur (przykład zapisu trójkąta):

```
struct triangle {
   struct pkt1;
   struct pkt2;
   struct pkt3;
};
```

W obecnym standardzie ANSI C jest dopuszczalne:

- operacja przypisania dla struktur (można przypisywać jedną do drugiej, kopiować jedną na drugą),
- przekazywanie do funkcji
- zwracanie jako wartości funkcji
- pobranie jej adresu za pomocą operatora `&`
- odwołania do jej składowych

Adresy i wskaźniki struktury danych

```
#include <stdio.h>
#include <string.h>
#include <math.h>
struct ABONENT {
   char nazwisko [25];
   char imie [15];
   char tytul [25];
   char ulica [60];
   char miasto [30];
   char wojewodztwo [20];
   char kod [6];
   float oplata;
   int termin;
};

/*  struct ABONENT przesylka;
struct ABONENT *przesylkaptr = &przesylka; */
struct ABONENT przesylka, *przesylkaptr = &przesylka;

void main()
{
 void odcz_inf_podst();
 odcz_inf_podst();
 printf("Nazwisko = %s %s \n",przesylkaptr->imie,
                              przesylkaptr->nazwisko);
 printf("Oplata = %05.1f\n",przesylkaptr->oplata);
 printf("Termin realizacji= %d\n",
         przesylkaptr->termin);
}
void odcz_inf_podst()
{
 strcpy (przesylkaptr->nazwisko , "Stanislawski" );
 strcpy (przesylkaptr->imie , "Piotr" );
 strcpy (przesylkaptr->tytul , "mgr" );
 strcpy (przesylkaptr->ulica , "Krucza" );
 strcpy (przesylkaptr->miasto , "Strzelin" );
```

```
 strcpy (przesylkaptr->wojewodztwo , "wroclawskie" );
 strcpy (przesylkaptr->kod , "57-100" );
 przesylkaptr->oplata = 10000;
 przesylkaptr->termin = 2;
}
```

Uwaga: Deklaracja wskaźnika nie powoduje deklaracji pamięci dla danych wskazywanych przez ten wskaźnik. Przed przypisaniem wskaźnikowi danej porcji pamięci, musi być wcześniej wykonana odpowiednia operacja przydziału pamięci.

```
Odwolanie          przesylkaptr->nazwisko
jest rownowazne    przesylka.nazwisko
albo               (*przesylkaptr).nazwisko
```

Operatory `. i ->` są lewostronnie łączne, zatem po definicji:

```
struct triangle tr, *trp=&tr;
```

następujące cztery wyrażenia są równoważne:

```
tr.pkt1.x
trp->pkt1.x
(tr.pkt1).x
(trp->pkt1).x
```

Operatory `. -> () []` znajdują się na szczycie hierarchii priorytetów. Zatem np. operacja

```
++trp->pkt1.x
```

zwiększa zmienną x, a nie wskaźnik *trp*. Na tej samej zasadzie po deklaracji

```
struct {
  int len;
  char *str;
} *p;
```

```
*p->str      udostepnia obiekt wskazywany przez str
*p->str++    zwieksza  str po udostepnieniu obiektu
             wskazywanego przez  str
(*p->str)++  zwieksza to cos, na co wskazuje  str
*p++->str    zwieksza  p  po udostepnieniu obiektu
             wskazywanego przez  str
```

Tablice struktur

Na przykładzie programu zliczającego liczbę wystąpień słów kluczowych:

```
I - uzycie dwoch rownoleglych tablic
   char *keyword[NKEYS];  /*  slowa kluczowe */
   int keycount[NKEYS];   /* liczniki tych slow */

II - uzycie struktur
   struct key {
     char *word;
     int count;
   } keytab[NKEYS];

albo

   struct key {
     char *word;
     int count;
   };
   struct key keytab[NKEYS];
```

Inicjalizacja tablicy struktur

```
struct key {
  char *word;
  int count;
} keytab[NKEYS] = {
     "auto", 0,
     "break", 0,
     "case", 0,
     /* ...  */
     "while", 0
   };
```

Przykład programu zliczającego wystąpienia słów kluczowych języka C:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define EOF 26
#define MAXWORD 100
#define NKEYS 4
struct key {
  char *word;
  int count;
} keytab[NKEYS] = {
     "auto", 0,
     "break", 0,
     "case", 0,
     /* ...  */
     "while", 0
   };

int getword(char *,int);
int binsearch(char *, struct key *, int);
/* zlicz slowa kluczowe C */
void main()
{  int n;
   char word[MAXWORD]; /* slowo */
   while (getword(word,MAXWORD) != EOF)
     if (isalpha(word[0]))
       if ((n = binsearch(word, keytab, NKEYS)) == 0)
         keytab[n].count++;
   for (n=0; n<NKEYS; n++)
     if (keytab[n].count > 0)
       printf("%4d %s", keytab[n].count,
                        keytab[n].word);
}
/*  binsearch: szukaj slowa w tab[0], ...,tab[n-1] */
int binsearch(char *word, struct key tab[], int n)
{  int cond,  low=0, high, mid;
   high=n-1;
   while (low <= high) {
      mid=(low+high)/2;
      if ((cond = strcmp(word,tab[mid].word)) < 0)
        high=mid-1;
      else if (cond > 0)
        low=mid+1;
      else
        return mid;
   }
   return -1;
}
/* getword: wczytaj nastepne slowo lub znak */
int getword(char *word, int lim)
{  int c, getch(void);
   void ungetch(int);
   char *w = word;
   while (isspace(c=getch())) ;
   if (c != EOF)
     *w++=c;
   else
     return EOF;
   if (!isalpha(c)) {
     *w='\0';
```

```
    return c;
  }
  for ( ; --lim > 0; w++)
    if (! isalnum(*w = getch())) {
      ungetch(*w);
      break;
    }
  *w='\0';
  return word[0];
}
```

Wskaźnikowa wersja powyższego programu:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#undef EOF
#define EOF 26
#define MAXWORD 100
#define NKEYS (sizeof keytab/sizeof(struct key))
struct key {
  char *word;
  int count;
} keytab[] = {
     "auto", 0,
     "break", 0,
     "case", 0,
     /* ...  */
     "while", 0
   };

int getword(char *,int);
struct key *binsearch(char *, struct key *, int);
/* zlicz slowa kluczowe C */
void main()
{  int n;
  char word[MAXWORD]; /* slowo */
  struct key *p;
  while (getword(word,MAXWORD) != EOF)
    if (isalpha(word[0]))
      if ((p=binsearch(word, keytab, NKEYS)) != NULL)
        p->count++;
  for (p=keytab; p<keytab+NKEYS; p++)
    if (p->count > 0)
      printf("%4d %s", p->count, p->word);
}
/*  binsearch: szukaj slowa w tab[0], ...,tab[n-1] */
struct key *
    binsearch(char *word, struct key tab[], int n)
{  int cond;
   struct key *low=&tab[0], *high=&tab[n], *mid;
   while (low < high) {
      mid=low+(high-low)/2;
      if ((cond = strcmp(word,mid->word)) < 0)
        high=mid;
      else if (cond > 0)
        low=mid+1;
      else
        return mid;
   }
   return NULL;
}
```

Uwaga: W arytmetyce języka zagwarantowano, że arytmetyka na wskaźnikach zadziała poprawnie dla pierwszego elementu poza końcem tablicy, mimo że pośrednie odwołanie do n-tego elementu jest błędem.

Unie

Cel unii – udostępnić jedną zmienną do przechowywania wartości kilku różnych typów.

Przykład:

```
union u-tag{
  int ival;
  float fval;
  char *sval;
} u;
```

Odwołania do składowych unii:

```
   nazwa-unii.skladowa
albo
   wskaznik-do-unii->skladowa
```

Unie mogą występować w strukturach i tablicach i vice-versa. Notacja w odwołaniach jest identyczna, jak dla struktur, np.

```
struct {
  char *name;  /* nazwa symbolu  */
  int flags;   /* znaczniki stanu */
  int utype;   /* typ wartosci */
  union {
    int ival;
    float fval;
    char *sval;
  } u;
} symtab[NSYM];  /* tablica symboli  */

Odwolanie do skladowej ival ma postac:
  symtab[i].u.ival

a odwolanie do pierwszego znaku tekstu wskazywanego
przez sval mozna zapisac na dwa sposoby:

  *symtab[i].u.sval
  symtab[i].u.sval[0]
```

Uwaga: Unię można zainicjować tylko wartością o typie jej pierwszej składowej.

Pola bitowe

Zdefiniowanie zbioru masek

```
#define KEYWORD  01  /*  slowo kluczowe     */
#define external 02  /*  obiekt zewnetrzny  */
#define STATIC   04  /*  obiekt statyczny   */

albo
enum { KEYWORD = 01, EXTERNAL = 02, STATIC = 04 };
```

Często pojawiające się zwroty:

```
flags |= EXTERNAL | STATIC;    /* ustawia bity */
flags &= ~(EXTERNAL & STATIC); /* kasuje bity */
if ((flags & (EXTERNAL | STATIC)) == 0) ...
/* warunek prawdziwy, gdy oba bity sa skasowane */
```

Pola bitowe – alternatywny mechanizm w języku C.
Polem bitowym jest zbiór przylegających do siebie bitów znajdujących się w jednej jednostce pamięci zwanej "słowem" (wielkość zależy od implementacji).

Powyższy zestaw symboli można zastąpić definicją trzech pól:

```
struct {
   unsigned int is_keyword : 1;  /*  1 - rozmiar */
   unsigned int is_extern  : 1;  /*      pola    */
   unsigned int is_static  : 1;  /*      w bitach */
} flags;
```

która definiuje zmienną *flags* o trzech jednobitowych polach. Odwołujemy się do nich, jak do zwykłych zmiennych. Zachowują się jak małe zmienne całkowitoliczbowe i mogą występować w wyrażeniach arytmetycznych na równi z innymi obiektami całkowitymi.

Inny zapis powyższych przykładów:

```
flags.is_extern = flags.is_static = 1;
flags.is_extern = flags.is_static = 0;
if (flags.is_extern == 0 && flags.is_static == 0) ...
```

Ograniczenia:

- pola na jednych maszynach od prawej do lewej, na drugich odwrotnie,
- pola są wielkościami całkowitymi bez znaku,
- na wielu maszynach mogą być umieszczane wyłącznie w słowach,
- nie są tablicami,
- nie mają adresu (nie można stosować operatora `&`).