Lecture 7 - structures and unions

STRUCTURES
Example - informations connected with a typical post address:

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

struct ABONENT {
    char lastname[25];
    char firstname[15];
    char title[25];
    char street[60];
    char city[30];
    char county[20];
    char code[6];
    float payment;
    int termin;
};

struct ABONENT mail;

void main()
{
    void odcz_inf_podst();
    odcz_inf_podst();
    printf("Name = %s %s
",
            mail.firstname, mail.lastname);
    printf("Payment = %05.1f
",
            mail.payment);
    printf("Termin of realization = %d
",
            mail.termin);
}

void odcz_inf_podst()
{
    strcpy(mail.lastname, "Stanislawski");
    strcpy(mail.firstname, "Piotr");
    strcpy(mail.title, "M.Sc.");
    strcpy(mail.street, "Krucza");
    strcpy(mail.city, "Strzelin");
    strcpy(mail.county, "wroclawskie");
    strcpy(mail.code, "57-100");
    mail.payment = 10000;
    mail.termin = 2;
}
```

Initialization of structure:

```c
struct point pkt = {10, 20};
```

Automatic structures may be also initialized by means of the assignment or call of a function, that returns structure of an appropriate type.

Accessing the structure members:

```c
name-of-structure.member-name
```

E.g. pkt.x

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

Nested structures (example of the triangle record):

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

In the present ANSI C standard it is allowed:

- assignment operation for structures (one may assign one to another, copy one onto other).
- pass to the function,
- return as the function value,
- take its address by means of the operator &
- recall its members.

Addresses and pointers to the data structures

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

struct ABONENT {
    char lastname[25];
    char firstname[15];
    char title[25];
    char street[60];
    char city[30];
    char county[20];
    char code[6];
    float payment;
    int termin;
};

/* struct ABONENT mail;*/
struct ABONENT mail, *mailptr = &mail;

void main()
{
    void odcz_inf_podst();
    odcz_inf_podst();
    printf("Name = %s %s 
",
            mailptr->firstname,
            mailptr->lastname);
    printf("Payment = %05.1f
",
            mailptr->payment);
    printf("Termin of realization = %d
",
            mailptr->termin);
}

void odcz_inf_podst()
{
    strcpy(mail.lastname, "Stanislawski");
    strcpy(mail.firstname, "Piotr");
    strcpy(mail.title, "M.Sc.");
    strcpy(mail.street, "Krucza");
    strcpy(mail.city, "Strzelin");
    strcpy(mail.county, "wroclawskie");
    strcpy(mail.code, "57-100");
    mail.payment = 10000;
    mail.termin = 2;
    mailptr = &mail;
}
```
strcpy (mailptr->lastname, "Stanislawski");
strcpy (mailptr->firstname, "Piotr");
strcpy (mailptr->title, "M.Sc.");
strcpy (mailptr->street, "Krucza");
strcpy (mailptr->city, "Strzelin");
strcpy (mailptr->county, "wroclawskie");
strcpy (mailptr->code, "57-100");
mailptr->payment = 10000;
mailptr->termin = 2;
}

Attention: Pointer declaration does not cause the memory declaration for data pointed by that pointer. Before an assignment to the pointer given portion of memory, an appropriate operation of the memory allocation must be carried out.

Recall mailptr->nazwisko is equivalent mail.nazwisko or (*mailptr).nazwisko

Operators . i- > are left-to-right associative, therefore after the following definition:

struct triangle tr, *trp=&tr;

four expressions below are equivalent:

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

Operators .- > ( ) [ ] are on the top of the priorities hierarchy. Therefore for instance the following operation:

++trp->pkt1.x
increases value of variable x, not the pointer trp. Due to the same rule after declaration.

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

*p->str gives rise to object indicated by str
*p->str++ increases str after taking the value of the object pointed by str
(p->str)++ increases that what is pointed by str
*p++->str increases p after accessing the object pointed by str

Arrays of structures

On an example of program counting number of appearance of some key words:

I - use of two parallel arrays
char *keyword[NKEYS]; /* keywords */
int keycount[NKEYS]; /* counters of keywords */

II - use of structures
struct key {
    char *word;
    int count;
}
struct key keytab[NKEYS];

Initialization of arrays of structures

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

realization of the sample program counting appearances of key words of C-language:

#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);
/* count keywords of C */
void main() {
    int n;
    char word[MAXWORD]; /* words */
    /* words */
    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: look for words in
    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 (strcmp(word, keytab[mid].word) < 0)
                for (n=0; n<NKEYS; n++)
                    if (keytab[n].count > 0)
                        printf("%4d %s", keytab[n].count,
                        keytab[n].word);
            if ((cond = strcmp(word, keytab[mid].word)) < 0)
                high=mid-1;
            else if (cond > 0)
                low=mid+1;
            else
                return mid;
        }
        return -1;
    }

/* count keywords of C */
void main() { int n;
    char word[MAXWORD]; /* words */
    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: look for words in
    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 (strcmp(word, keytab[mid].word) < 0)
                for (n=0; n<NKEYS; n++)
                    if (keytab[n].count > 0)
                        printf("%4d %s", keytab[n].count,
                        keytab[n].word);
            if ((cond = strcmp(word, keytab[mid].word)) < 0)
                high=mid-1;
            else if (cond > 0)
                low=mid+1;
            else
                return mid;
        }
        return -1;
    }
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];
}

The pointer version of the above program:

```
#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);
/* count keywords of C */
void main()
{
    char word[MAXWORD]; /* words */
    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: look for the word in 
       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;
    }

    int n;
    char word[MAXWORD]; /* words */
    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);
        )
```

Unions

Purpose of unions – make one variable accessible to store variables of various types.

Example:

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

Accessing the union members:

```
name-of-union.member
or
pointer-to-union->member
```

Unions may appear within structures and arrays and vice-versa. Notation in recalls is identical as for structures, for example:

```
struct {
    char *name; /* symbol name */
    int flags; /* state flags */
    int utype; /* type of value */
    union {
        int ival;
        float fval;
        char *sval;
    } u;
} symtab[NSYM]; /* array of symbols */
```

Recall of member ival has the following form

```
symtab[i].u.ival
```

and recall to the first character of text pointed by sval may be written in two ways:

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

Attention: Union may be initialized only with a value of type of its first member.

```
Attention: In the language arithmetic is guaranteed, that arith- 
metric on pointers works correctly for the first element after the 
end of the array, although the indirect recall of the n-th element 
is an error.
```