CODE ACADEMY

ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

Обединения, Изброим тип, Битови полета,

CODE ACADEMY

Обединения (union)

- обединенията, подобно на структурите, групират членове от различен тип

- предаването на обединения като параметър към функция е подобно на предаване на структура

- връщането на обединение като резултат от функция е подобно на връщането на структура

- основната разлика е, че обединението заделя памет, колкото да се събере **най-големият му елемент**.

- използва се ключовата дума union
- възможно е влагане на обединения едно в друго

CODE ACADEMY

Unions

Структурата се използва за да дефинира тип данни, състоящ се от няколко полета. Всяко поле заема отделно място в паметта.

Например, се появява и в паметта

```
struct rectangle {
int width;
int height;
};
```

Обединението е подобно на структурата, но то дефинира единична локация, на която могат да бъдат дадени различни имена но полето.

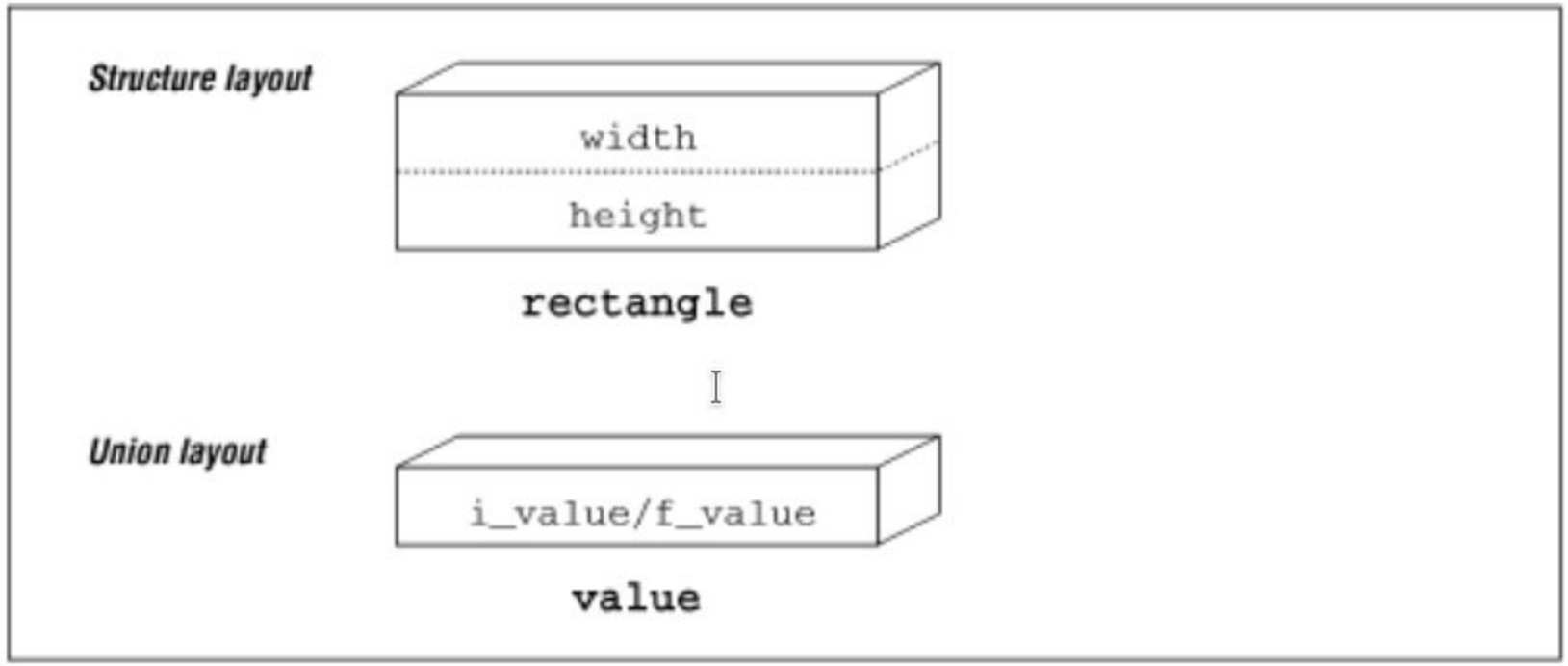
```
union value {
        long int i_value;  /* целочислена версия на стойността*/
        float f_value;      /* float версия на стойността */
};
```

Полетата **i_value** и **f_value** споделят едно и също място в паметта.

CODE ACADEMY

Структурата може да се мисли като голяма кутия, разделена на няколко различни отделения, всяко със свое собствено име.

Обединението е кутия, която не е разделена, но която има няколко различни етикети, поместени вътре в едно общо пространство.



CODE ACADEMY

Използване на обединения

```
/* Дефинираме променлива, която да съдържа цяла стойност или реална стойност, но не и двете */

union value {
        long int i_value;    /* The real number */
        float f_value;       /* The floating-point number */
} data;

int i;       /* Random integer */
float f;    /* Random floating-point number */

Int main()
{
        data.f_value = 5.0;
        data.i_value = 3;

        i = data.i_value;
        f = data.f_value;
        data.f_value; = 5.5;
        i = data.i_value;

        return(0);
}
```

```c
#include<stdio.h>
union value {
        long int i_value;     /* The real number */
        float f_value;         /* The floating-point number */
} data;

int i;        /* Random integer */
float f;     /* Random floating-point number */

int main()
{
    data.f_value = 5.0;
    data.i_value = 3;

printf("%d\t%f\n", data.i_value, data.f_value);

    i = data.i_value;
    f = data.f_value;

    printf("%d\t%f\n", i, f);

    data.f_value= 5.5;

    printf("%d\t%f\n", data.i_value, data.f_value);

    i = data.i_value;

    printf("%d\t%f\n", i, f);
    return(0);
```

```
/home/ana/CODE_ACADEMY/CODE/union1
3       0.000000
3       0.000000
1085276160      5.500000
1085276160      0.000000

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

CODE ACADEMY

Union често се използват в областта на комуникациите.

Например, ако отдалечено искате да изпратите 4 съобщения: open, close, read и write.

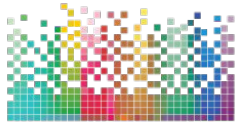
Данните, съдържащи се в тези четири съобщения се различават много в зависимост от съобщението.

Отвореното съобщение трябва да съдържа името на източника.

Написаното съобщение трябва да съдържа записани данни.

Съобщението за четене трябва да съдържа максимален брой символи, които да се прочетат.

Затвореното съобщение трябва да не съдържа повече информация.




```c
#define DATA_MAX 1024 /* Maximum amount of data for a read and write */
struct open_msg {
char name[30];
};
/* Name of the tape */
struct read_msg {
int length;
};
/* Max data to tranfer in the read */
struct write_msg {
int length;
 /* Number of bytes to write */
char data[DATA_MAX]; /* Data to write */
};

struct close_msg {
};

const int OPEN_CODE=0;
const int READ_CODE=1;
const int WRITE_CODE=2;
const int CLOSE_CODE=3;

struct msg {
int msg;
 /* Message type */
union {
struct open_msg open_data;
struct read_msg read_data;
struct write_msg write_data;
struct close_msg close_data

} msg_data;
};
```

Дефиниция

```
union union_name
{
    union_member1;
    union_member2;
    ...
    ...
    ...
    union_member_N;
};
```

Пример:

```
union color
{
    struct color_rgb  rgb;    // rgb color value
    struct color rgba rgba;   // rgba color value
    unsigned int  value;      // hexadecimal or integer value
    char name[20];            // Unique string representing color name
};

// Declare color type variable
union color console_color;
```

Достъп до елементите на union

Оператор . :

Syntax:

```
union_variable.member_name;
```

Example:

```
console_color.value = 999;
```

Чрез указател:

Syntax:

```
union_pointer->member_name;
```

Example:

```
console_color->value = 999;
```

CODE ACADEMY

Задачи

Задача 1. Напишете обединение от три стойности: цяло число, число с плаваща запетая и низ. Инициализирайте отделните членове и ги изведете на екрана.

Задача 2. Напишете обединение от число и низ, както и изброим тип за съдържанието на обединението. Напишете функция, която получава указател към обединението и изброимия тип и извежда съответно низ или число.

Изброим тип - enum

Изброимият тип данни се проектира за променливи, които съдържат само ограничено множество от стойности.

Пример. Ако ви трябва променлива, която да съдържа дните от седмицата

```
enum week_day {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};
```

Тогава се използва по следния начин

```
enum week_day today = TUESDAY;
```

Променливата today е от изброим тип с име **week_day**
и приема стойност TUESDAY

Изброим тип - enum

```
enum flag {const1, const2, ..., constN};
```

където const1...constN са интегрални константи

Промяна на стойностите по подразбиране:

```
enum suit {
     club = 0,
     diamonds = 10,
     hearts = 20,
     spades = 3,
 };
```

CODE ACADEMY

Изброим тип - enum

```
#include <stdio.h>

enum bool{false, true};

int main(){
	enum bool b = true;

	switch (b){
	case true:
		printf("true");
	   break;
	case false:
	   printf("False");
	   break;
	}
	return 0;
}
```

CODE ACADEMY

Изброим тип - enum

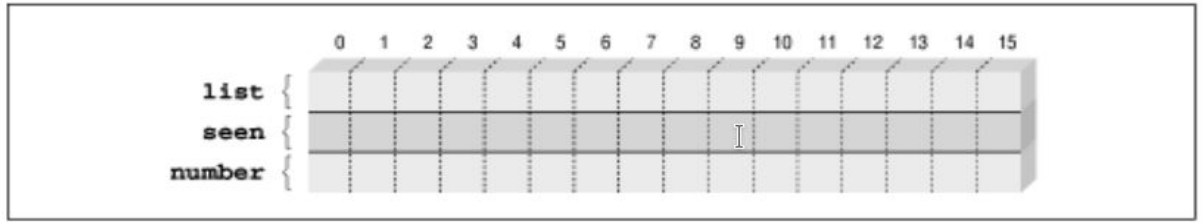
Анонимен **enum** т.е. няма име:

```
enum{
      VAL0 = 100,
      VAL1 = 200,
      VAL3 = 300,
      VALN = 19
};

int main(void){
      int a = VALN;
      printf("%d\n", a);
   return 0;
}
```

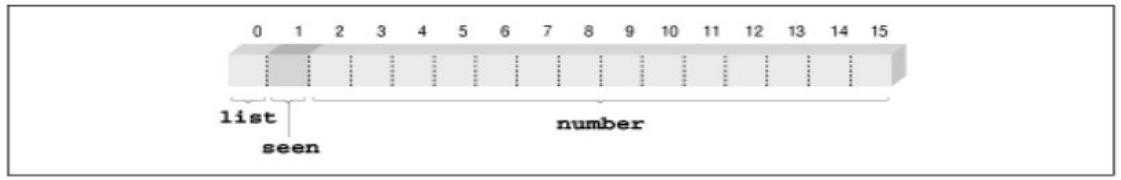
Bit Fields (Packed Structures)
Битови полета

Опакованите структури ни позволяват да декларираме структури по начин, който заема минимално количество място за съхранение. Например, следната структура заема шест байта (на 16-битова машина).



Непакетирана структура - 6 байта

```
struct item {
  unsigned int list;   /* true if item is in the list */
  unsigned int seen;  /* true if this item has been seen */
  unsigned int number;  /* item number */
};
```



Пакетирана структура

```
struct item {
unsigned int list:1;          /* true if item is in the list */  заема 1 бит
unsigned int seen:1;        /* true if this item has been seen */заема 1 бит
unsigned int number:14;     /* item number */
};
```

CODE ACADEMY

Bit Fields (Packed Structures)
Битови полета

- Битовите полета, подобно на структурите, групират членове от различен целочислен тип: char, int (с модификатори)

- членовете използват част от битовете на типовете

- предаването на битови полета като параметър към функция е подобно на предаване на структура

- връщането на битови полета като резултат от функция е подобно на връщането на структура

- използва се ключовата дума **struct**

CODE ACADEMY

Битови полета

```c
/* define simple structure */
struct {
   unsigned int widthValidated;
   unsigned int heightValidated;
} status1;

/* define a structure with bit fields */
struct {
   unsigned int widthValidated : 1;
   unsigned int heightValidated : 1;
} status2;

int main( ) {
   printf( "Memory size occupied by status1 : %d\n", sizeof(status1));
   printf( "Memory size occupied by status2 : %d\n", sizeof(status2));
   return 0;
}
```

CODE ACADEMY

Дефиниция, декларация и инициализация

```
struct {
    type [member_name] : width ;
};
```

1. **type** – целочислен тип, който определя как се интерпретира стойността на битовото поле. Типът може да бъде **int, signed int** или **unsigned int.**

2. **member_name** – Името на битовото поле.

3. **width** – Брой битове в битовото поле. Ширината трябва да е

<= от броят битове на определения тип.

Битови полета

CODE ACADEMY

Друг начин за представянето на числата в компютъра с използване на обединения и битови полета:

```
union
{
    struct {
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char reserved:4;
    } bits;

    unsigned char byte;

} HW_RegisterB;

HW_RegisterB reg;

x = reg.bits.b2;
```

CODE ACADEMY

Задачи

Задача 3. Дефинирайте структура, която да се състои от цели числа, битови полета и union. Попълнете всички елементи на структурата, като използвате операторите “.” и “->” за достъп до елементите. Разпечатайте отделните елементи, както и размера на структурата.

Задача 4. Напишете тип за дата, използвайки битови полета. Направете функция, която извежда дата, използвайки новия тип. Направете функция, която валидира датата.

CODE ACADEMY

Задачи

Задача 5. Дефинирайте структура с 4 полета (int. char[10], double, enum), направете масив от 20 структури и ги попълнете. Изведете масива от структури на стандартния изход в CSV формат.

Задача 6. Използвайки предната задача, напишете програма, която да чете от стандартния вход CSV формат на описаната структура и да попълва масив от 20 структури. **Пример: a.exe < structs20.csv**