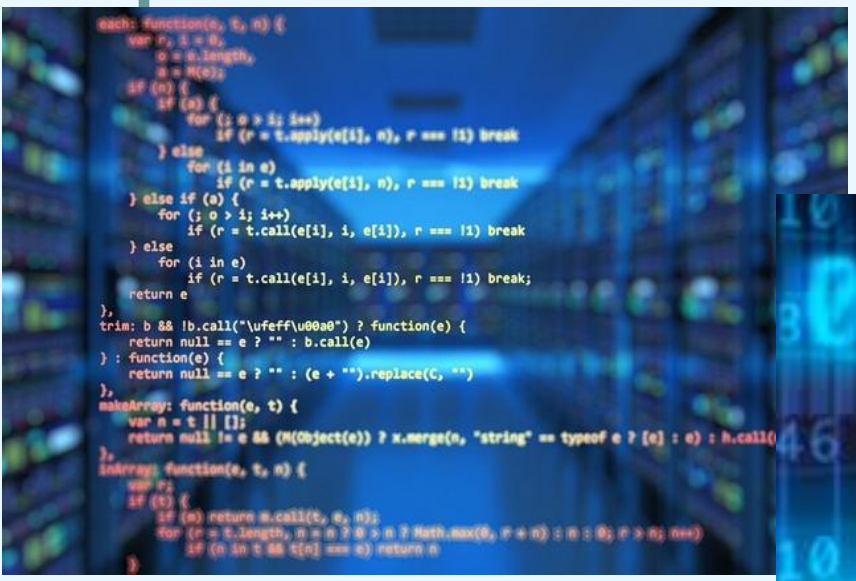
Структури. Об’єднання



Частина 3

Об’єднання

Сьогодні розглянемо один із важливих складних типів даних – об’єднання, їх особливості та практичне застосування.

Об’єднання широко використовуються у:

- ✓ системному програмуванні
- ✓ мережевих протоколах
- ✓ вбудованих системах
- ✓ роботі з пам’яттю

У структурі кожне поле має свою пам’ять.

```
struct A {
    int x;
    char y;
};
```

На відміну від структури, де для всіх її полів (елементів) виділяється окрема пам’ять, в об’єднанні всі поля спільно використовують одну й ту саму область пам’яті.

Проблема полягає в тому, що struct не завжди підходить:

- ✓ займає більше пам’яті
- ✓ неефективно, якщо використовується лише одне поле

Об’єднання

Об'єднаннями називають складний тип даних, що дозволяє розміщувати в тому самому місці оперативної пам'яті дані різних типів.

Об'єднання – це спеціальна структура в пам'яті, яка може містити об'єкти різних типів і розмірів таким чином, що вони зберігаються в пам'яті за однією й тією ж адресою.

Поля об’єднання застосовуються не для окремого зберігання даних, а для можливості звертатися до одних і тих самих даних різними способами (залежно від типу поля).

Об’єднання

Коли використовується **елемент меншої довжини**, ніж найдовший елемент об'єднання, цей елемент **використовує лише частина відведеної пам'яті**.

Всі елементи **об'єднання зберігаються в одній області пам'яті**, починаючи з однієї адреси.

Вимоги до розміру і вирівнювання значень полів в пам'яті **покладаються на компілятор**.

За допомогою об'єднань можна працювати з даними різних типів в межах однієї ділянки пам'яті, не привносячи в програму елементи низькорівневого, машинно-залежного програмування.

Об’єднання

Загальна форма оголошення об'єднання

```
union ІмяОб’єднання
{
  тип ІмяОб’єкта1;
  тип ІмяОб’єкта2;
  . . .
  тип ІмяОб’єктаn;
};
```

Ім’я 1			
Ім’я 2			
Ім’я 3			

В один момент часу коректним є **тільки одне поле**

Об'єднання застосовуються для таких цілей:

- ✓ для ініціалізації об'єкта, якщо у кожний момент часу лише один із багатьох об'єктів є активним;
- ✓ для інтерпретації подання одного типу даних як іншого типу.

Об’єднання

Наприклад, зручно використовувати об'єднання, коли необхідно дійсне число типу float подати у вигляді сукупності байтів (N = 15.3)

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

union types{
    float f;
    unsigned char b[4];
};

int main(){
    union types value;
    printf("N = ");
    scanf("%f", &value.f);
    printf("%f = %x %x %x %x", value.f, value.b[0], value.b[1], value.b[2], value.b[3]);
    getchar();
    getchar();
    return 0;
}
```

```
N = 15.3
15.300000 = cd cc 74 41


Program finished with exit code 0
```

Число **15.3** у пам’яті комп’ютера зберігається не як "**15.3**", а у форматі **IEEE 754** (float).
Його двійкове представлення: **0x4174CCCD**

Об’єднання

Поміняти місцями два молодші байти у введеному числі

(Введіть число : 450)

Потрібно:

- ✓ ввести ціле число (наприклад, **450**)
- ✓ представити його як **послідовність байтів**
- ✓ **поміняти місцями два молодші байти**
- ✓ отримати нове число

Об’єднання

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

int main() {
    unsigned char temp;

    union {
        unsigned char p[2];
        unsigned int t;
    } type;

    printf("Введіть число: ");
    scanf("%d", &type.t);

    printf("До заміни:\n");
    printf("%d = %04x шістн.\n", type.t, type.t);
    printf("Байти: %02x %02x\n", type.p[0], type.p[1]);

    // Заміна двох молодших байтів
    temp = type.p[0];
    type.p[0] = type.p[1];
    type.p[1] = temp;

    printf("\nПісля заміни байтів одержали:\n");
    printf("%u = %04x шістн.\n", type.t, type.t);
    printf("Байти: %02x %02x\n", type.p[0], type.p[1]);

    return 0;
}
```

```
Введіть число: 450
До заміни:
450 = 01c2 шістн.
Байти: c2 01

Після заміни байтів одержали:
49665 = c201 шістн.
Байти: 01 c2


...Program finished with exit code 0
Press ENTER to exit console.
```

450 (десяткове) = 0x01C2

Байт	Значення
p[0]	C2
p[1]	01

Нове число:
01 C2 → 0xC201

Об’єднання

Ще один приклад, який ілюструє визначення синоніма типу для об’єднання, ініціалізацію об’єднання, анонімні структури, а також можливість варіантного (тобто здійснюваного різними способами) звернення до об’єднання та його компонентів.

Дослідити об’єднання (union), яке дозволяє:

- зберігати дані в одній області пам’яті;
- звертатися до них різними способами:
  - ✓ як до числа (v2);
  - ✓ як до масиву байтів (s[ ]);
  - ✓ як до полів анонімних структур (x, y, s0, s1).

Об’єднання

```c
#include <stdio.h>

typedef union {
   short  v2;
   char   s[2];
   struct { char x, y; };
   struct { char s0, s1; };
} char2;

char2 C2 = { 0x1234 };

int main(void)
{
    printf("%04x\n", C2.v2);
    printf("%02x %02x\n", C2.s[0], C2.s[1]);
    printf("%02x %02x\n", C2.x, C2.y);
    printf("%02x %02x\n", C2.s0, C2.s1);

    return 0;
}
```

```
1234
34 12
34 12
34 12


...Program finished with exit code 0
```

Для значення: 0x1234

- ✓ число виводиться як 0x1234
- ✓ байти: 34 12 – через порядок **little-endian**
- ✓ всі способи доступу (s, x,y, s0,s1) дають однаковий результат

Об’єднання

Об’єднання в мові С дають змогу зберігати дані в одній області пам’яті та звертатися до них у різний спосіб.

У наведеному прикладі одне й те саме значення розглядається як ціле число, як масив байтів і як поля анонімних структур.

Це дозволяє гнучко працювати з поданням даних у пам’яті та є особливо корисним у системному програмуванні, при роботі з форматами даних, протоколами та низькорівневими операціями.

Бітові поля

Використовуючи структури, можна запакувати цілі компоненти ще більш щільно, ніж це було зроблено з використанням масиву.

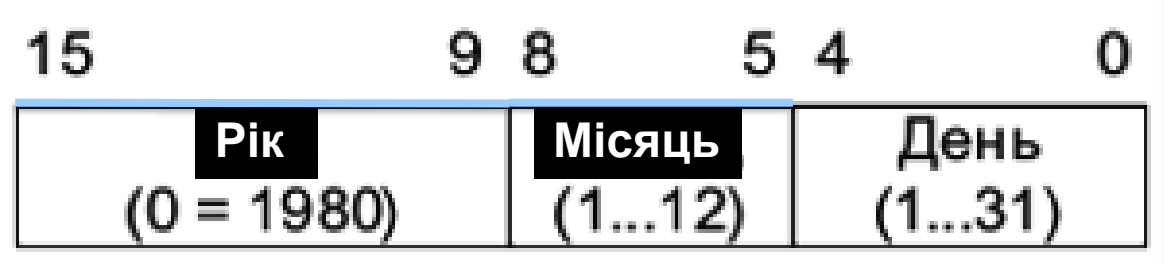
Набір розрядів цілого числа можна розбити на бітові поля, кожне з яких виділяється для певної змінної.

При роботі з бітовими полями кількість бітів, що виділяється для зберігання кожного поля, відокремлюється від імені двокрапкою.

```
тип ім’я: КількістьБіт
```

Працюючи з бітовими полями потрібно уважно стежити, щоб значення змінної не зажадало пам'яті більше, ніж під неї виділено.

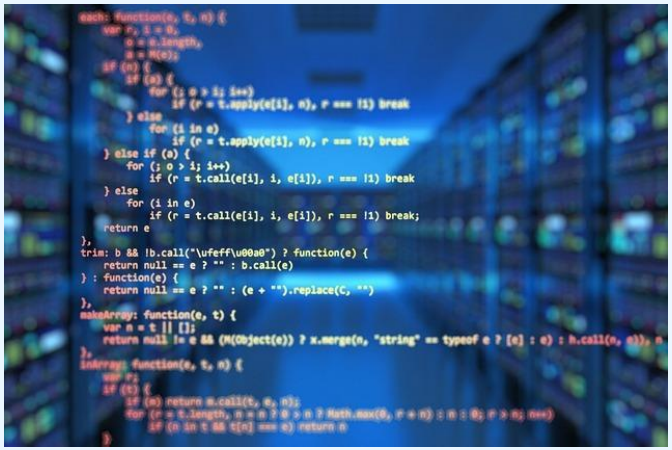
Приклад. Розробити програму, яка здійснює упаковку дати у формат



```c
#include <stdio.h>
#include <stdlib.h>
#define YEAR0 1980

struct date{
    unsigned short day: 5;
    unsigned short month: 4;
    unsigned short year:7;
};

int main() {
    struct date today;
    today.day = 26;
    today.month = 03;
    today.year = 2026 - YEAR0; //today.year = 33
    printf("\n Сьогодні %u.%u.%u \n", today.day, today.month, today.year + YEAR0);
    printf("\n Розмір структури today : %d байт", sizeof(today));
    printf("\n Значення елемента today = %hu = %hx шістн.", today, today);
    return 0;
}

```

input

```
Сьогодні 26.3.2026

Розмір структури today : 2 байт
Значення елемента today = 23674 = 5c7a шістн.

...Program finished with exit code 0
```



Дякую за увагу!