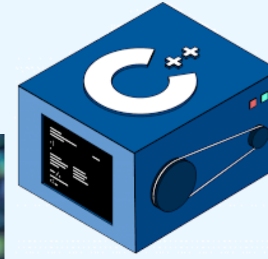
Бітові поля, флаги, маски



Бітові поля

Базові поняття

Мовам програмування властива деяка марнотратність при роботі з цілими числами.

Оскільки мінімальний розмір адресованої одиниці пам'яті дорівнює 8 байт, при роботі з цифрами велика кількість бітів дорівнює нулю і є не інформативним.

Для подолання цього недоліку в мові С/С++ передбачена підтримка **бітових полів**.

Сфера їхнього застосування досить велика: при роботі з логічними змінними ***true*** і ***false***, при роботі з індикаторами стану фізичних пристроїв (портів і т.п.) і при шифруванні.

Базові поняття

Програмуючи мовою асемблер, ми часто використовували окремі байти як резервуари для 8-ми прапорців (бітів), за допомогою яких ми реалізовували логіку програми.

Такий підхід давав нам можливість економити регістри загального призначення.

Мовою C/C++ також можна реалізувати такий підхід, працюючи з окремими бітами вибраного байта.

Для цього ми оголошують певну змінну типу **unsigned char** та за допомогою порозрядних операцій працюємо з її окремими бітами.

Базові поняття

В основу бітових полів покладене поняття структури.

Бітове поле - це біт-орієнтований член структури.

Бітове поле є різновидом елемента структури, розмір якого можна задати в бітах.

Визначення бітового поля має наступний вигляд:

```
struct ім'я_структури {
        тип ім'я1:кількість_біт;
        тип ім'я2:кількість_біт;
         . . .
        тип имя:кількість_біт;
} список_змінних;
```

Базові поняття

Членами бітового поля можуть служити лише змінні типів **char, int** з відповідними модифікаціями.

Якщо розмір бітового поля дорівнює одиниці, його тип можна вказати за допомогою специфікатора **unsigned**.

Наприклад, бітове поле **ftime**, визначене в заголовному файлі **io.h**, виглядає в таким чином:

```
struct ftime {
    unsigned ft_tsec : 5;     /* Двосекундний інтервал */
    unsigned ft_min : 6;      /* Хвилини */
    unsigned ft_hour : 5;     /* Години */
    unsigned ft_day : 5;      /* Дні */
    unsigned ft_month : 4;    /* Місяці */
    unsigned ft_year : 7;     /* Роки */
};
```

Базові поняття

Оскільки бітове поле, власне кажучи, є різновидом структури, при роботі з ним використовуються ті ж оператори: **. (“точка”)** і **-> (“стрілка”)**.

Наприклад, щоб змінити установку часу, можна виконати наступні оператори.

```
ftime.ft_min = 30;
ftime.ft_hour = 12;
```

Якщо на бітове поле встановлений вказівник, для доступу до його елементів застосовується оператор **->**.

Базові поняття

```
struct Flags
{
     unsigned f0:1;
     unsigned f1:1;
     unsigned f2:1;
     unsigned f34:2;
     unsigned f57:3;
}
```

Доступ до окремих бітів байта виконується таким чином:

```
struct Flags cond;
cond.f0=1;
cond.f1=0;
cond.f2=0;
cond.f34=3; // 0b11
cond.f57=5; // 0b101
```

Змінна cond розміщується компілятором в оперативній пам’яті SRAM.

Тому зміна значень окремих бітів виконується за допомогою порозрядних операцій та числових масок, які обчислюються компілятором

Базові поняття

Елементи бітового поля іменувати не обов’язково.

Це дає можливість виділяти інформативні біти й ігнорувати непотрібні.

Наприклад, в одному з наведених прикладів з бітовими полями можна “відключити” елементи, що зберігають інформацію про час.

```
struct ftime {
    unsigned : 5;              // Двосекундний інтервал
    unsigned : 6;              // Хвилини
    unsigned : 5;              // Години
    unsigned ft_day : 5;       // Дні
    unsigned ft_month : 4;     // Місяці
    unsigned ft_year : 7;      // Роки
};
```

Базові поняття

Бітові поля можуть бути елементами звичайних структур.

Крім того, у звичайній структурі можна використовувати поля з зазначеними розмірами.

```
struct TimeData {
    unsigned int day : 5
    char month : 4;              // 1 ... 12
    unsigned int year : 7;
    unsigned int hour : 5;
    unsigned int minute : 6;
    float second;
};
```

```
struct Data {
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year : 7;
};

struct Time {
    unsigned int hour : 5;
    unsigned int min : 6;
    unsigned int sec : 5;
};

struct TimeData {
    struct Data a;
    struct Time b;
};
```

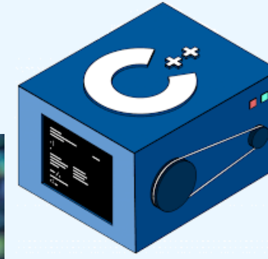
Базові поняття

У 32-розрядній операційній системі розмір цієї структури дорівнює 16 байт. Якби бітові поля не використовувалися, то її розмір дорівнював би 24 байт.

Бітові поля є машинно-орієнтованими.

Отже, їхнє застосування супроводжується певними обмеженнями:

1. Адресу бітового поля одержати неможливо.
2. Бітові поля не можуть бути елементами масиву.
3. Бітові поля не можуть бути статичними.



Бітові маски та прапорці (Flags). Їх використання

Бітові флаги

Використовуючи цілий байт для зберігання значення логічного типу даних, ви займаєте тільки 1 біт, а решта 7 з 8 — не використовуються.

Хоча в цілому це нормально, але в особливих, ресурсоємних випадках, пов'язаних з безліччю **логічних значень**, може бути корисно “упакувати” 8 значень типу bool в 1 байт, заощадивши при цьому пам'ять і збільшивши, таким чином, продуктивність.

Ці окремі біти і називаються **бітовими флагами**. Оскільки прямого доступу до цих біт немає, то для операцій з ними використовуються **побітові оператори**.

```
// Визначаємо 8 окремих бітових флаги (вони можуть представляти все, що ви захочете).
// Зверніть увагу, що в C++11 краще використовувати "uint8_t" замість "unsigned char"
const unsigned char option1 = 0x01; // шістнадцятковий літерал для 0000 0001
const unsigned char option2 = 0x02; // шістнадцятковий літерал для 0000 0010
const unsigned char option3 = 0x04; // шістнадцятковий літерал для 0000 0100
const unsigned char option4 = 0x08; // шістнадцятковий літерал для 0000 1000
const unsigned char option5 = 0x10; // шістнадцятковий літерал для 0001 0000
const unsigned char option6 = 0x20; // шістнадцятковий літерал для 0010 0000
const unsigned char option7 = 0x40; // шістнадцятковий літерал для 0100 0000
const unsigned char option8 = 0x80; // шістнадцятковий літерал для 1000 0000

// Байтове значення для зберігання комбінацій з 8 можливих варіантів
unsigned char myflags = 0; // всі флаги/параметри вимкнені до старту
```

Щоб дізнатися бітовий стан, використовуйте побітове І:

```
if (myflags & option4) ... // якщо встановлено option4, то робимо що-небудь
```

Щоб увімкнути біти, використовуйте побітове АБО:

```
myflags |= option4; // вмикаємо option4
myflags |= (option4 | option5); // вмикаємо option4 і option5
```

Щоб *вимкнути біти*, використовуйте побітове І (в зворотній послідовності):

```
myflags &= ~option4; // вимикаємо option4
myflags &= ~(option4 | option5); // вимикаємо option4 і option5
```

Для перемикання між станами бітів, використовуйте побітове виключне АБО (XOR):

```
myflags ^= option4; // вмикаємо або вимикаємо option4
myflags ^= (option4 | option5); // змінюємо стан option4 і option5
```

Уявіть, що у вас є функція, яка може приймати будь-яку комбінацію з 32 різних варіантів.
Одним із способів написання такої функції є використання 32 окремих логічних параметрів:

```
void someFunction(bool option1, bool option2, bool option3,
bool option4, bool option5, bool option6, bool option7, bool option8,
bool option9, bool option10, bool option11, bool option12,
bool option13, bool option14, bool option15, bool option16,
bool option17, bool option18, bool option19, bool option20,
bool option21, bool option22, bool option23, bool option24,
bool option25, bool option26, bool option27, bool option28,
bool option29, bool option30, bool option31, bool option32);
```

Потім, якщо ви захочете викликати функцію з 10-м і 32-м параметрами, встановленими як true — вам доведеться зробити щось типу наступного:

```
someFunction(false, false, false, false, false, false, false, false, false,
true, false, false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false, false,
true);
```

Тобто порахувати всі варіанти як ***false***, крім 10 і 32 — вони ***true***.
Читати такий код складно, та й потрібно пам’ятати порядкові номери потрібних параметрів (10 і 32 чи 11 і 33?). Такий код не може бути ефективним.

А ось якщо визначати функцію, використовуючи бітові флаги:

```
void someFunction(unsigned int options);
```

То можна вибирати і передавати тільки потрібні параметри:

```
someFunction(option10 | option32);
```

Крім того, що це читабельніше, це також ефективніше і продуктивніше, оскільки включає тільки 2 операції (одне побітове АБО і одна передача параметрів).

Бітова маска – це сукупність бітів-«прапорців», яка використовується для виділення, встановлення (в значення 1) або скидання (в 0) певної групи бітів за допомогою побітових операцій.

Бітова маска, як правило, являє собою *ціле число без знаку*, внутрішнє двійкове представлення якого має певний зміст для програміста.

Розглянемо для спрощення деяку величину x довжиною 8 біт, наприклад, число 121, яке має внутрішнє двійкове представлення 0111 1001.

Для виділення старшого біта (біта 7) цієї величини введемо маску y = 1000 0000, в якій встановлений (у значення 1) лише потрібний біт 7.

Використаємо записану маску для виконання різних побітових операцій над x.

Результати виконання цих операцій зведені в таблицю:

x	0111 1001	x	0111 1001
y	1000 0000	~y	0111 1111
x & y	0000 0000	x & ~y	0111 1001
x	0111 1001	x	0111 1001
y	1000 0000	~y	0111 1111
x | y	1111 1001	x | ~y	0111 1111
x	0111 1001	x	0111 1001
y	1000 0000	~y	0111 1111
x ^ y	1111 1001	x ^ ~y	0000 0110

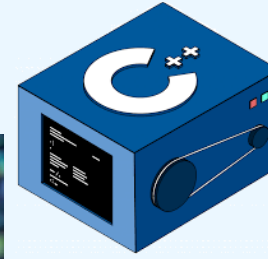
Як видно, операція x & y дає нульовий результат.

Це означає, що біт 7 (єдиний встановлений біт у масці y) має значення 0 для величини x.

Операція x & y дає можливість перевірити встановлені, чи ні всередині x біти, що відповідають масці.

Для встановлення бітів у числі x достатньо виконати операцію x | y – 162 тоді ті біти, які були встановлені у масці y, будуть встановлені і в x.

Нарешті, операція x & ~y може бути використана для скидання всередині x всіх бітів, встановлених у масці y.



Приклади.

Приклад 1

Розробити програму на С, яка зберігає стан елементів текстового редактора:

- виділено (*sel*),
- активна вкладка (*act*),
- режим вставки (*ins*),
- автозбереження (*as*).

Змоделювати ввімкнення/вимкнення кожної опції та виводити поточний стан.

Код в середовищі Code::Bloks

```
1  #include <stdio.h>
2  #include <windows.h>
3  #include <conio.h>
4
5  struct EdFlags {
6      unsigned int sel : 1;
7      unsigned int act : 1;
8      unsigned int ins : 1;
9      unsigned int as  : 1;
10 };
11
12 void drEdFl(struct EdFlags fls) {
13     printf("\nСтан редактора:\n");
14     printf("  1 - Виділено:         %s\n", fls.sel ? "Так" : "Ні");
15     printf("  2 - Активна вкладка: %s\n", fls.act ? "Так" : "Ні");
16     printf("  3 - Режим вставки:   %s\n", fls.ins ? "Вставка" : "Перезапис");
17     printf("  4 - Автозбереження:  %s\n", fls.as  ? "Увімкнено" : "Вимкнено");
18     printf("Натисніть цифри від 1 до 4, щоб змінити параметр. Для виходу - ESC.\n");
19 }
```

Використовується структура з **бітовими полями** (по 1 біту на кожен стан)

Функція **drEdFl() -** виведення стану.
Ця функція читає значення полів і виводить їх у зручному для користувача вигляді.
fls.sel ? "Так" : "Ні" означає:
якщо біт sel = **1**, то виводиться "**Так**", інакше — "Ні".

Код в середовищі Code::Bloks

```
20
21  int main() {
22      SetConsoleCP(1251);
23      SetConsoleOutputCP(1251);
24
25      struct EdFlags ed = {0};
26      ed.sel = 1;
27      ed.as = 1;
28
29      char a;
30      do {
31          drEdFl(ed);
32
33          a = _getch();
34
35          switch (a) {
36              case '1': ed.sel = !ed.sel; break;
37              case '2': ed.act = !ed.act; break;
38              case '3': ed.ins = !ed.ins; break;
39              case '4': ed.as = !ed.as; break;
40          }
41      } while (a != 27);
42
43      printf("\nВихід із програми.\n");
44      return 0;
45  }
```

SetConsoleCP(1251) та **SetConsoleOutputCP(1251)** встановлюють кодування **Windows-1251** (кирилиця), щоб коректно виводити/зчитувати українські символи в консолі Windows.

Всі біти обнуляються ({**0**}), потім прапорці **sel** і **as** встановлюються в 1 (тобто "**ввімкнено**").

Результат тестування

C:\Codes\Ci\BitF_1\bin\Debug\BitF_1.exe

```
Стан редактора:
  1 - Виділено:        Так
  2 - Активна вкладка: Ні
  3 - Режим вставки:   Перезапис
  4 - Автозбереження:  Увімкнено
Натисніть цифри від 1 до 4, щоб змінити параметр. Для виходу - ESC.

Стан редактора:
  1 - Виділено:        Так
  2 - Активна вкладка: Ні
  3 - Режим вставки:   Перезапис
  4 - Автозбереження:  Увімкнено
Натисніть цифри від 1 до 4, щоб змінити параметр. Для виходу - ESC.

Стан редактора:
  1 - Виділено:        Так
  2 - Активна вкладка: Ні
  3 - Режим вставки:   Вставка
  4 - Автозбереження:  Увімкнено
Натисніть цифри від 1 до 4, щоб змінити параметр. Для виходу - ESC.

Стан редактора:
  1 - Виділено:        Так
  2 - Активна вкладка: Ні
  3 - Режим вставки:   Вставка
  4 - Автозбереження:  Вимкнено
Натисніть цифри від 1 до 4, щоб змінити параметр. Для виходу - ESC.

Вихід із програми.

Process returned 0 (0x0)   execution time : 70.588 s
Press any key to continue.
```

Код в середовищі OnlineGDB

Середовище OnlineGDB:
- ✓ не підтримує windows.h, SetConsoleCP та SetConsoleOutputCP
- ✓ не підтримує _getch() з conio.h
- ✓ працює з scanf()

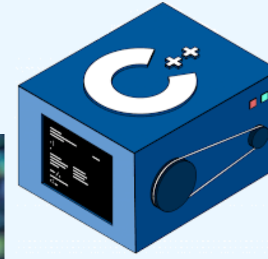
main.c

```
1  #include <stdio.h>
2
3  struct EdFlags {
4      unsigned int sel : 1; // Виділення
5      unsigned int act : 1; // Активна вкладка
6      unsigned int ins : 1; // Режим вставки
7      unsigned int as  : 1; // Автозбереження
8  };
9
10 void drEdFl(struct EdFlags fls) {
11     printf("\nСтан редактора:\n");
12     printf("  1 - Виділено:        %s\n", fls.sel ? "Так" : "Ні");
13     printf("  2 - Активна вкладка: %s\n", fls.act ? "Так" : "Ні");
14     printf("  3 - Режим вставки:   %s\n", fls.ins ? "Вставка" : "Перезапис");
15     printf("  4 - Автозбереження:  %s\n", fls.as  ? "Увімкнено" : "Вимкнено");
16     printf("  0 - Вихід\n");
17     printf("Виберіть параметр для зміни: ");
18 }
19
```

Код і результати тестування

```c
20 int main() {
21     struct EdFlags ed = {0};
22     ed.sel = 1;
23     ed.as = 1;
24 
25     int a;
26 
27     do {
28         drEdFl(ed);
29 
30         if (scanf("%d", &a) != 1) {
31             // Очистити вхідний потік у разі помилки
32             while (getchar() != '\n');
33             continue;
34         }
35 
36         switch (a) {
37             case 1: ed.sel = !ed.sel; break;
38             case 2: ed.act = !ed.act; break;
39             case 3: ed.ins = !ed.ins; break;
40             case 4: ed.as = !ed.as; break;
41             case 0: printf("\nВихід із програми.\n"); break;
42             default:
43                 printf("Невірний вибір! Введіть число від 0 до 4.\n");
44         }
45     } while (a != 0);
46 
47     return 0;
48 }
49
```

```
Стан редактора:
  1 - Виділено:         Ні
  2 - Активна вкладка: Так
  3 - Режим вставки:    Перезапис
  4 - Автозбереження:   Увімкнено
  0 - Вихід
Виберіть параметр для зміни: 3

Стан редактора:
  1 - Виділено:         Ні
  2 - Активна вкладка: Так
  3 - Режим вставки:    Вставка
  4 - Автозбереження:   Увімкнено
  0 - Вихід
Виберіть параметр для зміни: 4

Стан редактора:
  1 - Виділено:         Ні
  2 - Активна вкладка: Так
  3 - Режим вставки:    Вставка
  4 - Автозбереження:   Вимкнено
  0 - Вихід
Виберіть параметр для зміни: 5
Невірний вибір! В�едіть число від 0 до 4.

Стан редактора:
  1 - Виділено:         Ні
  2 - Активна вкладка: Так
  3 - Режим вставки:    Вставка
  4 - Автозбереження:   Вимкнено
  0 - Вихід
Вибе�іть параметр для зміни: 0

Вихід із програми.


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



Дякую за увагу!