

# Типи даних, що визначає програміст

*Всякий необходимо причиняет пользу,  
употребленный на своем месте.*

*Козьма Прутков*

\*

# Типи даних, що визначає програміст

---

У реальних задачах інформація, що потребує обробки, може мати досить складну структуру. Для її представлення використовують типи даних побудовані на основі простих типів даних, масивів, покажчиків.

Мова C++ дозволяє програмісту визначати свої типи даних та правила роботи з ними. Історично для основних таких типів склалися свої власні назви, але їх можна об'єднати під назвою *типів, що визначає програміст*.

# Структури

---

Структури (записи) – гнучкий структурований тип даних, що дозволяє поєднувати в єдине ціле елементи різних типів.

Структури – використовують для адекватного представлення реальних об'єктів, де елементи (поля) визначають різні характеристики об'єктів.

Використання:

- таблиці із різнотипною складноструктурованою інформацією;
- файли, бази даних;
- динамічні структури даних;
- ...

# Структури

```
struct [ім`я типу] {  
    <тип_1> <елемент_1>;  
    <тип_2> <елемент_2>;  
    ...  
    <тип_N> <елемент_N>;  
} [<список змінних>;];
```

Елементи структури називають *полями*.

Типи полів можуть бути різними, але не функціональними (не типами функцій та покажчиків на них), а також не типом самої структури (але може бути покажчиком на нього).

Розмір пам`яті не обов`язково дорівнює сумі розмірів полів (за рахунок вирівнювання на границі слова).

# Приклади

---

```
struct date {  
    int day;  
    int month;  
    int year;  
};  
date dt_brt[10], *pdt;  
  
struct {  
    float x, y;  
} position;  
//struct position tk1, tk2;
```

# Приклади ініціалізації

---

```
struct person {  
    char name[20];  
    date b_d;};  
person man = {"Bill", {7, 11, 1935}};
```

```
struct complex {  
    float re, im;  
} compl [2][3] = {  
    {{1,1}, {1,2}, {1,3}},  
    {{2,1}, {2,2}, {2,3}}  
};
```

# Дії

Для змінних одного структурного типу – *операція присвоювання* (поелементне копіювання).

Структури можна передавати у функцію через параметри та повертати як значення функції.

Доступ до полів структури здійснюється операціями:

- . (вибору) через ім'я змінної-структури;
- -> через покажчик на структуру.

```
dt_brt[1].day = 15;
```

```
pdt -> month = 6;
```

```
if (pdt->year == 2012 && pdt->month == 2) pdt->day =  
29;
```

```
compl [1][2].re = 2.4; compl [1][2].im = 5.55;
```

До структур можна застосовувати операцію `&`.

# Приклад

```
struct complex {  
    float re, im;  
};  
complex add_complex (complex c1, complex c2){  
    complex c3 = c1;  
    c3.re += c2.re;  
    c3.im += c2.im;  
    return c3;  
}  
...  
complex x, y, z;  
...  
z = add_complex(x, y);  
cout << "z = x+y = " << z.re << " + " << z.im << "i" << endl;
```



# Структури та функції

---

При використанні структур в якості параметрів функцій потрібно враховувати можливості параметрів-значень та параметрів-посилань.

Якщо функція не змінює значення полів структури її можна передавати як параметр-значення.

Якщо функція повинна змінювати значення полів структури передавати структуру як параметр-посилання.

*Приклади.*

# Бітові поля

---

Особливий вид полів структури. Використовується для щільної упаковки невеликих даних. При визначенні поля вказується його розмір у бітах.

```
struct optins {  
    bool centerX:1;  
    bool centerY:1;  
    unsigned int shadow:2;  
    unsigned int palette:4;  
} pd;  
...  
pd.centerX = 0; pd.centerY = 0;  
pd.shadow = 3; pd.palette = 7;
```

# Бітові поля

---

Бітові поля можуть бути довільного цілого типу. Доступ здійснюється традиційним чином через ім'я поля.

Фактично такі поля надають доступ до окремих частин байтів – бітів пам'яті.

Отримати адресу такого поля не можна. Не можна створювати масиви бітових даних.

Ім'я поля може бути відсутнім. Такі поля використовуються для вирівнювання на апаратні границі. Доступ до самого такого поля відсутній.

Дозволяють заощадити пам'ять, але слід враховувати, що операції з окремими бітами реалізуються суттєво менш ефективно ніж операції з байтами та словами. Такі програми значною мірою є машино-залежними.

# Об'єднання

---

Змінні типу об'єднання (суміш) можуть набувати значень кількох різних типів. Можна трактувати їх як структури, всі поля яких розташовані за однією адресою, в одній спільній ділянці, достатній для збереження найбільшого за розміром поля.

Фактично вони надають можливість доступу до однієї й тієї самою ділянки пам'яті за допомогою змінних різних типів. В об'єднанні допускаються й структури з бітовими полями, що надає можливості доступу до окремих бітів ділянки. Зрозуміло що в кожен момент часу у такій змінній зберігається лише одне значення.

Відповідальність за правильність використання покладається на програміста.

# Об`єднання

---

```
union onefrom {  
    int int_val;  
    long long_val;  
    double double_val;  
};  
onefrom zmn;  
...  
zmn.int_val = 15;  
cout << zmn.int_val;  
...  
zmn.double_val = 1.38;  
cout << zmn.double_val;
```

# Об'єднання

---

```
struct widget { //структура реєстру
    char brand[20];
    int type;
    union id { //залежить від типу
        long id_num;
        char id_char[20];
    } id_val;
};
...
widget prize;
...
if (prize.type == 1) cin >> prize.id_val.id_num;
else cin >> prize.id_val.id_char;
```

# Об'єднання

---

```
struct widget { //структура реєстру
    char brand[20];
    int type;
    union { //залежить від типу
        long id_num;
        char id_char[20];
    };
};

...
widget prize;

...
if (prize.type == 1) cin >> prize. id_num;
else cin >> prize. id_char;
```

# Перелічення

(порядкові типи)

Спосіб створення іменованих констант, зв'язаних між собою. Сприяє наочності програмних рішень, а також забезпечує контроль з боку системи програмування за значеннями відповідних змінних.

**Формат:**

```
enum [<ім`я_типу>] {<список_констант>} [<змінні>];
```

<ім`я\_типу> - визначається, якщо потрібно окремо визначати змінні цього типу. Змінні можуть приймати значення тільки з <список\_констант>.

Константи з <список\_констант> - цілі. Можуть бути ініціалізовані звичайним чином, або за умовчанням у першої константи значення – 0, у кожній наступній значення – на 1 більше ніж у попередньої.

Операції: присвоювання, порівняння. При виконанні арифметичних операцій перетворюються на цілі.



# Перелічення. Приклади

---

```
enum spectrum {red, orange, yellow, green, blue, violet,  
               indigo, ultraviolet};
```

Визначили новий тип `spectrum`, а також іменовані константи з значеннями 0 – 7.

```
spectrum band; //змінна з типом spectrum
```

```
band = blue;
```

```
band = 2000; //не припустимо
```

значення змінної `band` обмежені 8 переліченими.

```
int color = blue; //припустимо
```

```
band = spectrum(3); //приведення 3 до типу spectrum
```

```
band = spectrum(40003); //невизначеність
```

# Перелічення. Приклади

---

```
enum Err{ERR_READ, ERR_WRITE, ERR_CONVERT};
```

```
Err error;
```

```
...
```

```
switch (error) {
```

```
    case ERR_READ: /* оператори */ break;
```

```
    case ERR_WRITE: /* оператори */ break;
```

```
    case ERR_CONVERT: /* оператори */ break;
```

```
};
```

```
enum {two = 2, three, ten = 10, eleven, fifty = ten + 40};
```

```
enum {zero, null = 0, one, numero_uno = 1};
```

//імена констант різні, значення можуть співпадати

# Приклади

---

```
enum paytype {CARD, CHECK};
paytype ptype;
union payment {
    char card[25];
    long check;
} info;
... //отримання значень info та ptype
switch (ptype) {
    case CARD: cout << "Оплата карткою: " << info.card;
               break;
    case CHECK: cout << "Оплата чеком: " << info.check;
                break;
}
...
```

# Приклади

---

```
enum paytype {CARD, CHECK};
struct {
    paytype ptype;
    union {
        char card[25];
        long check;
    };
} info;
... //отримання значень info
switch (info.ptype) {
    case CARD: cout << "Оплата карткою: " << info.card;
               break;
    case CHECK: cout << "Оплата чеком: " << info.check;
               break; } //...
```

# Визначення типів

---

Можна створювати додаткові імена для існуючих та нових типів.

**Формат:**

```
typedef <тип> <нове_ім`я> [<розмірність>];
```

**Наприклад:**

```
typedef int LENGTH;  
typedef unsigned int UINT;  
typedef char Msg[100];  
typedef struct {  
    char fio[30];  
    int date, code;  
    double salare;  
} Worker;
```

# Приклади

---

```
typedef int LENGTH;  
typedef unsigned int UINT;  
typedef char Msg[100];  
typedef struct {  
    char fio[30];  
    int date, code;  
    double salare;  
} Worker;  
LENGTH n, m, k;  
UINT i, j;  
Msg str[10];  
Worker staff[100];
```

# Приклади

---

```
typedef struct tnode {  
    char *word;  
    struct tnode *left;  
    struct tnode *right;  
} TREENODE, *TREEPTR;
```

```
TREEPTR talloc(){  
    return ((TREEPTR) calloc(1, sizeof(TREENODE)));  
}
```

```
TREEPTR p, t;  
p = talloc();
```

# Приклади

---

```
struct my_struct {  
    int a;  
    double b;  
};  
//або  
typedef struct {  
    int a;  
    double b;  
} my_struct;  
//використання типу  
my_struct *s;  
s->a = 3;  //(*s.a)=3;  
s->b = 4.5; //(*s.b)=4.5;
```



# Підсумки

---

- Розглянули спектр різних можливостей, що до визначення власних типів даних.
- Розглянуті типи дозволяють найбільш адекватним чином представляти різноманітну внутрішньо складну інформацію, забезпечувати гнучкість, наочність та перенесення програмного забезпечення.
- Окремі системи програмування можуть надавати додаткові можливості, мати певні особливості в роботі з такими даними.

# Поради

---

- Грамотно, з розумінням використовувати розглянуті можливості для визначення потрібних типів даних.
- Не зловживати “недокументованими можливостями”, особливостями окремих систем програмування.

# Задачі

---

- У послідовності дат (число, місяць, рік) знайти зимові дати.
- У заданій множині точок площини знайти 2 найбільш віддалені точки.
- Визначити тип, що дозволить представляти геометричні фігури : прямокутник, коло, правильний багатокутник.
- $N$  трикутників задані координатами своїх вершин. Визначити, який з них має найбільшу площу.

# Задачі

---

- Написати логічну функцію  $\text{Equally}(a,b)$  , що порівнює два раціональних числа.
- Написати функцію для скорочення раціональної дробі.
- Написати функцію для додавання раціональних дробів.
- Написати функцію  $\text{Rmax}(x, m)$  , що присвоює параметру  $m$  найбільше з раціональних чисел масиву  $x$ .
- Є координати вершин трикутника та координати деякої внутрішньої точки. Знайти відстань від даної точки до найближчої сторони трикутника.