ТИПЫ ДАННЫХ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

Типы данных, определяемые пользователем

- Переименование типов (typedef).
- Перечисления(enum).
- Структуры (struct).
- Битовые поля.
- Объединения (union).

Переименование типов (typedef)

- Для того чтобы сделать программу более ясной, можно задать типу новое имя с помощью ключевого слова **typedef** .

typedef тип новое_имя [размерность];

Размерность может отсутствовать.

Примеры:

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

Переименование типов (typedef).

ПЕРЕИМЕНОВАНИЕ ТИПОВ

Кроме задания типам с длинными описаниями более коротких псевдонимов, **typedef** *используется для облегчения переносимости программ:*

- *если машинно-зависимые типы объявить с помощью операторов* **typedef**, *при переносе программы потребуется внести изменения только в эти операторы*

ПЕРЕИМЕНОВАНИЕ ТИПОВ

Новые имена типов можно использовать таким же образом, как и имена стандартных типов.

```
UINT i, j;        // две переменных типа unsigned int

Msg str[10];  // массив из 10 строк по 100 символов

Worker staff[100];    // массив из 100 структур
```

Перечисления (enum)

- При написании программ часто возникает потребность определить несколько именованных констант, для которых требуется, чтобы все они имели различные значения, *при этом конкретные значения могут быть не важны*
- Для этого удобно воспользоваться перечисляемым типом данных, все возможные значения которого задаются списком целочисленных констант.

Формат:

enum [имя_типа] { список_констант };

Имя типа задается в том случае, если в программе требуется определять переменные этого типа.

Компилятор обеспечивает, чтобы эти переменные принимали значения только из списка констант

Константы должны быть целочисленными и могут инициализироваться обычным образом.

При отсутствии инициализатора первая константа обнуляется, а каждой следующей присваивается па единицу большее значение, чем предыдущей

П Е Р Е Ч И С Л Е Н И Я

Пример:

```
enum Err {ERR_READ,  ERR_WRITE, ERR_CONVERT};
Err error;
……
switch (error){
    case ERR_READ;  /* операторы */ break;
    case ERR_WRITE;     /* операторы */ break;
    case ERR_CONVERT;  /* операторы */ break;
}
```

Константам **ERRREAD**, **ERRWRITE**, **ERRCONVERT** присваиваются значения **0**, **1** и **2** соответственно.

П Е Р Е Ч И С Л Е Н И Я

Пример:

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

Константам **three** и **four** присваиваются значения 3 и 4, константе **eleven** — 11.

П Е Р Е Ч И С Л Е Н И Я

Структуры (**struct**)

- В языке **C++** структура является видом класса и обладает всеми его свойствами.
- Во многих случаях достаточно использовать структуры так, как они определены в языке **C**.

Формат:

```
struct [ имя_типа ] {
    тип_1  элемент_1;
    тип_2  элемент_2;
    тип_n  элемент_n;
} [ список_описателей ];
```

Элементы структуры называются ***полями структуры*** и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него.

С Т Р У К Т У Р Ы

Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов.

Определение массива структур и указателя на структуру:

```
struct {
    char  fio[30];
    int  date, code;
    double  salary;
} staff[100], *ps;
```

В этом случае описание структуры служит определением элементов этого списка.

С Т Р У К Т У Р Ы

Если список отсутствует, описание структуры определяет новый тип, имя которого можно использовать в дальнейшем наряду со стандартными типами.

Например:

```
struct Worker{          // описание нового типа Worker
    char fio[30];
    int date, code;
    double salary;
};                      // описание заканчивается точкой с запятой
    /* определение массива типа Worker и указателя на тип Worker*/
Worker staff[100], *ps;
```

С Т Р У К Т У Р Ы

Имя структур можно использовать сразу после объявления в тех случаях, когда компилятору не требуется знать размер структуры.

Например:

```
struct List;                    // объявление структуры List
struct Link{
    List *p;            // указатель на структуру List
    Link *prev, *succ;      // указатели на структуру Link
};
. . . . . .
struct List { /* определение структуры List */};
```

Это позволяет создавать связные списки структур.

С Т Р У К Т У Р Ы

Для **инициализации** структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

```
struct{
    char fio[30];
    int date, code;
    double salary;
}worker = {"Страусенко", 31, 215, 3400.55};
```

Инициализация структур

Инициализация структур

При инициализации массивов структур следует заключать в фигурные скобки каждый элемент массива:

```
struct complex{
    float re,  im;
} compl[2][3] = {
    {{1, 1}, {1, 1}, {1, 1}}, // строка 1, то есть массив compl[0]
    {{2, 2}, {2, 2}, {2, 2}}  // строка 2. то есть массив compl[1]
    };
```

Учтите, что многомерный массив — это массив массивов

Доступ к полям структуры

- Доступ к полям структуры выполняется с помощью :
  - операции выбора . (точка)
  - при обращении к полю через имя структуры
  - **->**
  - при обращении через указатель

```
Worker worker, staff[100], *ps;
    ……
    worker.fio = "Страусенко";
    staff[8].code = 215;
    ps -> salary = 0.12;
```

Если элементом структуры является другая структура, то доступ к ее элементам выполняется через две операции выбора.

```
struct A {int a; double x;};
struct B {A a; double x;} x[2];
x[0].a.a = 1;
x[1].x = 0.1;
```

Для переменных одного и того же структурного типа определена **операция присваивания**

- при этом происходит поэлементное копирование

Структуру можно передавать в функцию и возвращать в качестве значения функции

Другие операции со структурами могут быть определены пользователем

Операции со структурами

Битовые поля

- Битовые поля — это особый вид полей структуры.
- Они используются для плотной упаковки данных, например, флажков типа «**да**/**нет**».

При описании битового поля, после имени через двоеточие указывается длина поля в битах (целая положительная константа).

```
struct Options{
    bool centerX: 1;
    bool centerY: 1;
    unsigned int shadow: 2;
    unsigned int palette: 4;
};
```

Битовые поля

Битовые поля могут быть любого целого типа.

Имя поля может отсутствовать

- такие поля служат для выравнивания на аппаратную границу

Доступ к полю осуществляется обычным способом — по имени

Адрес поля получить нельзя, однако в остальном битовые поля можно использовать точно так же, как обычные поля структуры

Битовые поля

Следует учитывать, что операции с отдельными битами реализуются гораздо менее эффективно, чем с байтами и словами и экономия памяти под переменные оборачивается увеличением объема кода программы.

Битовые поля

Объединения (union)

- Объединение (**union**) представляет собой частный случай структуры, все поля которой располагаются по одному и тому же адресу.
- Объединения применяют для экономии памяти в тех случаях, когда известно, что больше одного поля одновременно не требуется.
- Формат описания такой же, как у структуры, только вместо ключевого слова **struct** используется слово **union**.

Длина объединения равна наибольшей из длин его полей

В каждый момент времени в переменной типа объединение хранится только одно значение,
и ответственность за его правильное использование лежит на программисте

Пример:

```
#include <iostream.h>
int main(){
    enum paytype {CARD, CHECK};
    paytype ptype;
    union payment{
     char card[25];
     long check;
    } info;
                /* присваивание значений info и ptype */
    switch (ptype){
     case CARD: cout << "Оплата по карте: " << info.card; break;
     case CHECK: cout << "Оплата чеком: " << info.check; break;
    }
    return 0;
}
```

О Б Ъ Е Д И Н Е Н И Я

По сравнению со структурами на объединения налагаются некоторые ограничения:

- Объединение может инициализироваться только значением его первого элемента
- Объединение не может содержать битовые поля
- Объединение не может содержать виртуальные методы, конструкторы, деструкторы и операцию присваивания
- Объединение не может входить в иерархию классов

О Б Ъ Е Д И Н Е Н И Я