



ТИПЫ ДАННЫХ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

**Дальневосточный государственный университет путей сообщения,
кафедра «Вычислительная техника и компьютерная графика»,
к.т.н., доцент Белозеров Олег Иванович.**

Вопросы лекции:

1

Переименование типов данных typedef

2

Перечисление enum

3

Структуры struct

4

Битовые поля

5

Объединения union

В реальных задачах информация, которую требуется обрабатывать, может иметь достаточно сложную структуру. Для ее адекватного представления используются типы данных, построенные на основе простых типов данных, массивов и указателей.

Язык C++ позволяет программисту определять свои типы данных и правила работы с ними.

Существует пять способов создания своих типов данных:

Структура - это совокупность переменных, объединенных одним именем – составной (или смешанный) тип данных.

Битовое поле – разновидность структуры, предоставляющая легкий доступ к отдельным битам.

Объединение позволяет одному участку памяти содержать два или более различных типов данных.

Перечисление – список символов.

`typedef` – ключевое слово, которое создает новое имя существующему типу.

1. Переименование типов данных typedef

Помимо явного объявления типа в C++ предусмотрены дополнительные средства описания имён типов. Таким средством является typedef-объявление.

Другими словами, для того чтобы сделать программу более ясной, можно задать типу новое имя с помощью ключевого слова typedef:

typedef тип новое_имя [размерность];

В данном случае квадратные скобки являются элементом синтаксиса. Размерность может отсутствовать.

С помощью typedef в программу можно ввести новые имена, которые затем можно использовать для обозначения производных и основных типов.

Далее рассмотрены примеры работы с typedef-объявлением.

```
typedef unsigned int UINT;
```

```
typedef char Msg[100];
```

```
typedef struct{
```

```
char fio[30];
```

```
int date, code;
```

```
double salary;}
```

```
Worker;
```

Введенное таким образом имя 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 stuff[100];   // массив из 100 структур
```

Это объявление начинается спецификатором `typedef`, содержит спецификатор объявления `int` и список описателей, в который входит два элемента: имя `Step` и имя `pInteger`, перед которым стоит символ `ptr` Операции `*`.

Объявление эквивалентно паре `typedef`-объявлений следующего вида:

```
typedef int Step;
```

```
typedef int *pInteger;
```

В соответствии с `typedef`-объявлениями, транслятор производит серию подстановок, суть которых становится понятной из анализа примера, в котором пара операторов объявления

```
Step StepVal;
```

```
extern pInteger pVal;
```

Заменяется следующими объявлениями:

```
int StepVal;
```

```
extern int *pVal;
```

Синтаксис typedef

Таким образом, typedef - объявление является объявлением, которое начинается спецификатором typedef и состоит из последовательностей разнообразных спецификаторов объявления и описателей. Список описателей (элементы списка разделяются запятыми) может содержать языковые конструкции разнообразной конфигурации. В него могут входить описатели с символами ptrОпераций (* и &), описатели, заключённые в круглые скобки, описатели в сопровождении заключённых в скобки списков объявлений параметров, описателей const и volatile, а также заключённых в квадратные скобки константных выражений.

Пример typedef-объявления:

```
typedef int Step, *pInteger;
```


Синтаксис typedef

- после возможного этапа декомпозиции списка описателей typedef-объявления, в результате которого может появиться новая серия typedef-объявлений, транслятор переходит к анализу операторов объявлений;
- в очередном операторе объявления выделяется идентификатор, стоящий на месте спецификатора объявления;
- среди typedef-объявлений производится поиск соответствующего объявления, содержащего вхождение этого идентификатора в список описателей. Таким образом, транслятор находит соответствующий контекст для подстановки. Этот контекст называется контекстом замены. Контекст замены оказывается в поле зрения транслятора вместе с оператором объявления, в котором транслятор различает спецификатор объявления и описатель;
- оператор объявления заменяется контекстом замены, в котором совпадающий со спецификатором объявления идентификатор заменяется соответствующим описателем.

Пример. Можно создать новое имя для `float`:

```
typedef float balance;
```

Данный оператор сообщает компилятору о необходимости распознавать `balance` как другое имя для `float`. Далее можно создать вещественную переменную, используя `balance`:

```
balance past_due;
```

Здесь `past_due` – вещественная переменная типа `balance` (`float`). Можно использовать `typedef` для создания имен для более сложных типов.

Например:

```
typedef struct {  
    float due;  
    int over_due;  
    char name[40];  
} client; /* здесь client - это имя нового типа */  
client clist[NUM_CLIENTS]; /* массив структур типа client */
```

Использование `typedef` может помочь при создании более легкого для чтения и более переносимого кода.

Правила

Использование спецификатора `typedef` подчиняется следующим правилам:

1. Спецификатор `typedef` может переопределять имя как имя типа, даже если это имя само уже было ранее введено `typedef` спецификатором:

```
typedef int I;
```

```
typedef I I;
```

2. Спецификатор `typedef` не может переопределять имя типа, объявленное в одной и той же области действия, и замещающее имя другого типа.

```
typedef int I;
```

```
typedef float I; // Ошибка: повторное описание
```

3. На имена, введённые в программу с помощью спецификатора `typedef`, распространяются правила области действия, за исключением разрешения на многократное использование имени.

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char* argv[])
7
8      setlocale(LC_ALL, "Russian");
9
10     typedef float I;
11     I V = 0, t = 0, S = 0;
12
13     cout << "Введите t(мин)" << endl;
14     cin >> t;
15
16     cout << "Введите S(км)" << endl;
17     cin >> S;
18
19     t /= 60;
20
21     V = S / t;
22
23     cout << "V=" << V << "км/ч" << "=" << V / 3.6 << "м/с" << endl;
24
25     return 0;
26 }
27
```

2. Перечисление 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.

Перечисления представляют собой список идентификаторов, введенных пользователем:

```
enum name_list {name1,name2,...};
```

За каждым таким именем по умолчанию закрепляются целочисленные константы:

имени name1 соответствует константа 0;

имени name2 соответствует константа 1;

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 на единицу большее значение, чем предыдущей:

```
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 ERR_READ, ERR_WRITE, ERR_CONVERT присваиваются значения 0, 1 и 2 соответственно.

Преимущество применения перечисления перед описанием именованных констант и директивой `#define` состоит в том, что связанные константы нагляднее; кроме того, компилятор при инициализации констант может выполнять проверку типов.

При выполнении арифметических операций перечисления преобразуются в целые. Поскольку перечисления являются типами, определяемыми пользователем, для них можно вводить собственные операции.

Для инициализации значений нумератора не с 0, а с другого целочисленного значения, следует присвоить это значение первому элементу списка значений перечислимого типа.

```
{
enum eDirection {RIGHT, LEFT, DOWN, UP}; // создаем перечисление с
// именем дескриптора eDirection
enum eDirection dir; // создаем переменную dir
// с перечислимым типом eDirection
dir = UP; // присваиваем переменной dir константу UP
}
```

```
{
enum {RIGHT, LEFT, DOWN, UP}; // создаем перечисление без дескриптора,
// просто набор констант
int dir; // создаем переменную dir с перечислимым типом int
dir = UP; // присваиваем переменной dir константу UP
}
```

```
{
typedef enum {RIGHT, LEFT, DOWN, UP} eDirection; // создаем тип
// перечисления с именем eDirection
eDirection dir; // создаем переменную dir с перечислимым типом eDirection
dir = UP; // присваиваем переменной dir константу UP
}
```



```
1      #include <iostream>
2
3      using namespace std;
4
5      int main(int argc, char* argv[])
6      {
7          setlocale(LC_ALL, "Russian");
8          {enum eDirection { RIGHT, LEFT, DOWN, UP };
9          enum eDirection dir;
10         dir = UP;
11         cout << dir << endl;
12     }
13     {enum { RIGHT, LEFT, DOWN, UP };
14     int dir;
15     dir = UP;
16     cout << dir << endl;
17     }
18     {
19         typedef enum { RIGHT, LEFT, DOWN, UP } eDirection;
20         eDirection dir;
21         dir = UP;
22         cout << dir << endl;
23     }
24     return 0;
25 }
26
```

З
З
З

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Г
leApplication9.exe (процесс 11484) завершил работу с к
Чтобы автоматически закрывать консоль при остановке от
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:

```
1      #include <iostream>
2
3      using namespace std;
4
5      int main(int argc, char* argv[])
6      {
7          setlocale(LC_ALL, "Russian");
8          {enum eDirection { RIGHT, LEFT, DOWN, UP };
9          enum eDirection dir;
10         dir = RIGHT;
11         cout << dir << endl;
12     }
13     {enum { RIGHT, LEFT, DOWN, UP };
14     int dir;
15     dir = LEFT;
16     cout << dir << endl;
17     }
18     {
19         typedef enum { RIGHT, LEFT, DOWN, UP } eDirection;
20         eDirection dir;
21         dir = DOWN;
22         cout << dir << endl;
23     }
24     return 0;
25 }
26
```

0
1
2

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программир
Чтобы автоматически закрывать консоль при остановке отладки, вк
Нажмите любую клавишу, чтобы закрыть это окно:


Конечный тип перечисления зависит от реализации компилятора, также все может зависеть от того, какие значения имеют константы. Если значение первой константы не указано, оно по умолчанию равно 0, все следующие за ней, на 1 больше предыдущей.

```
{ enum eDirection
{ RIGHT, // по умолчанию = 0
  LEFT,  // = 1
  DOWN,  // = 2
  UP     // = 3
};
}
```

```
{ enum eDirection
{ RIGHT, // по умолчанию = 0
  LEFT = 4, // = 4
  DOWN,    // = 5
  UP       // = 6
};
}
```

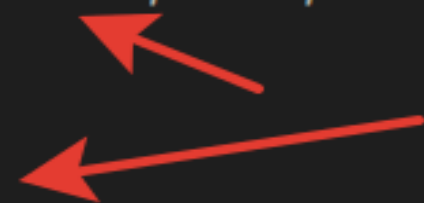
```
{ enum eDirection
{ RIGHT = 8, // = 8
  LEFT,      // = 9
  DOWN = 100, // = 100
  UP         // = 101
};
}
```

```
1      #include <iostream>
2
3      using namespace std;
4
5      int main(int argc, char* argv[])
6      {
7          setlocale(LC_ALL, "Russian");
8          {enum eDirection { RIGHT=1, LEFT, DOWN, UP };
9          enum eDirection dir;
10         dir = RIGHT;
11         cout << dir << endl;
12     }
13     {enum { RIGHT, LEFT, DOWN, UP };
14     int dir;
15     dir = LEFT;
16     cout << dir << endl;
17     }
18     {
19         typedef enum { RIGHT, LEFT, DOWN, UP } eDirection;
20         eDirection dir;
21         dir = DOWN;
22         cout << dir << endl;
23     }
24     return 0;
25 }
26
```



```
1
1
2
C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программное обеспечение\leApplication9.exe (процесс 13540) завершил работу с кодом 0
Чтобы автоматически закрывать консоль при остановке отладки,
автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

```
1      #include <iostream>
2
3      using namespace std;
4
5      int main(int argc, char* argv[])
6      {
7          setlocale(LC_ALL, "Russian");
8          {enum eDirection { RIGHT=1, LEFT, DOWN, UP };
9          enum eDirection dir;
10         dir = RIGHT;
11         cout << dir << endl;
12         dir = LEFT;
13         cout << dir << endl;
14     }
15     {enum { RIGHT, LEFT, DOWN, UP };
16     int dir;
17     dir = LEFT;
18     cout << dir << endl;
19     }
20     {
21     typedef enum { RIGHT, LEFT, DOWN, UP } eDirection;
22     eDirection dir;
23     dir = DOWN;
24     cout << dir << endl;
25     }
26     return 0;
27 }
```



2

Чтобы автоматически закрывать консоль при остановке отлад

Нажмите любую клавишу, чтобы закрыть это окно:

Использование элементов перечисления должно подчиняться следующим правилам:

1. Переменная может содержать повторяющиеся значения.
2. Идентификаторы в списке перечисления должны быть отличны от всех других идентификаторов в той же области видимости, включая имена обычных переменных и идентификаторы из других списков перечислений.
3. Имена типов перечислений должны быть отличны от других имен типов перечислений, структур и смесей в этой же области видимости.
4. Значение может следовать за последним элементом списка перечисления.

3. Структуры struct

Перечисления очень широко используются многими системными программами, особенно графическими:

```
enum line_style{SOLID_LINE,    //сплошная линия
                DOTTED_LINE,    //пунктирная линия
                CENTER_LINE,    //штрих-пунктирная линия
                DASHED_LINE,    //штриховая линия
                USERBIT_LINE}; //линия, определяемая пользователем
enum  COLORS {BLACK, BLUE, GREEN, CYAN, RED, MAGENTA,
             BROWN,...};
```

Системный набор операций над переменными типа перечислений довольно ограниченный: им можно присваивать значения из объявленного списка, сравнивать значения однотипных переменных, передавать в качестве параметров другим функциям.

Предыдущую запись можно представить как:

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

Далее представлено использование различных структур:

```
struct tovar {  
char name[15];      /* Наименование */  
int price;          /* Оптовая цена */  
float percent;      /* Наценка в % */  
int vol;            /* Объем партии */  
char date [9]; }    /* Дата поставки */
```

Общая форма объявления структуры:

```
struct ярлык {  
    тип ИмяЭлемента1;  
    тип ИмяЭлемента2;  
    ...  
    тип ИмяЭлементап;  
};
```

Определён тип, переменная не определена

Ярлык – имя типа структуры, а не имя переменной.

Структурные переменные – разделенный запятыми список имен переменных. Ярлык, или структурные переменные могут отсутствовать, но не одновременно.

Объявление структуры - это оператор. Поэтому её объявление завершается точкой с запятой.

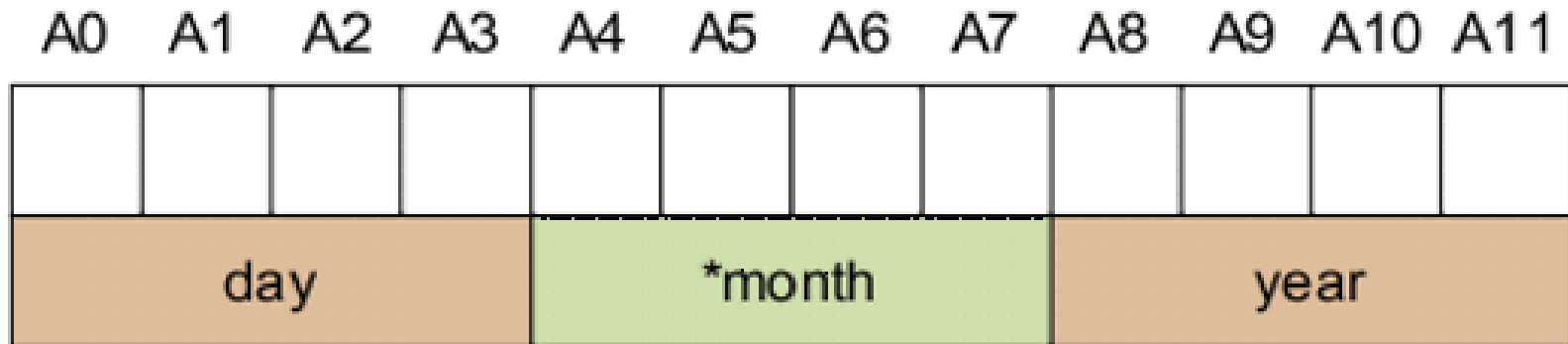
```
struct ярлык {  
    тип ИмяЭлемента1;  
    тип ИмяЭлемента2;  
    ...  
    тип ИмяЭлементап;  
} структурные_переменные;  
Определён тип, переменная определена
```

```
struct {  
    тип ИмяЭлемента1;  
    тип ИмяЭлемента2;  
    ...  
    тип ИмяЭлементап;  
} структурные_переменные;  
Переменная типа структура определена
```

Пример объявления структуры

```
struct date {  
    int day;        // 4 байта  
    char *month;    // 4 байта  
    int year;       // 4 байта  
};
```

Поля структуры располагаются в памяти в том порядке, в котором они объявлены:



Структура date занимает в памяти 12 байт.

Указатель *month при инициализации будет началом текстовой строки с названием месяца, размещенной в памяти

Доступ к отдельным членам структуры осуществляется с помощью оператора «.» (называется «точкой»). Стандартный вид доступа следующий:

`имя_структуры.имя_члена`

Вывод поля структуры на экран (например):

```
printf("const char *format", имя_структуры.имя_члена);  
имя_структуры.имя_члена = 1000;
```

Инициализация полей структуры может осуществляться двумя способами:

- присвоение значений элементам структуры в процессе объявления переменной, относящейся к типу структуры;
- присвоение начальных значений элементам структуры с использованием функций ввода-вывода (например, `printf()` и `scanf()`).

Инициализация структуры

При инициализации структур непосредственно в определении конкретной структуры после ее имени и знака “=” в фигурных скобках размещается список начальных значений элементов.

Например:

struct

tovar coat = { “пиджак черный”, 4000, 7.5, 220, “12.11.99” } ;

tovar tea = { “чай зеленый”, 2500, 6, 100, “25.02.00” } ;

students student_1 = { “Игнатъев”, “Олег”, 1 } ;

Для структур одного типа допустимо следующее присваивание:

tea=coat;

Для структур не определены операции сравнения. Сравнить структуры можно только поэлементно.

Доступ к элементам структур обеспечивается с помощью уточненных имен. Уточненное имя – это выражение с двумя операндами, операцией доступа к элементам структуры («точка» между ними). Уточненное имя используется для выбора правого операнда операции «точка» из структуры, задаваемой левым операндом, следующим образом:

имя структуры. имя элемента

Элементы структуры называются полями структуры и могут иметь любой тип. Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов. В этом случае описание структуры служит определением элементов этого списка:

// Определение массива структур и указателя на структуру:

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

Примеры использования

// объявление массива структур

```
struct addr addr_list[100];
```

// объявление указателя на структуру

```
struct addr *addr_pointer;
```

// использование вложенных структур

```
struct emp {  
    struct addr address; /* вложенная структура */  
    float salary;  
} worker;
```

```
1  #include <iostream>
2  #include <string>
3
4  struct person
5  {
6      int age;
7      std::string name;
8  };
9
10 int main()
11 {
12     person tom;
13     tom.name = "Tom";
14     tom.age = 34;
15     std::cout << "Name: " << tom.name << "\tAge: " << tom.age << std::endl;
16     return 0;
17 }
18
```

Name: Tom Age: 34

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программиров
leApplication9.exe (процесс 6380) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включ
томатически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно:

```
person tom = { 34, "Tom" };
```

Кроме того мы можем инициализировать структуру, присвоив ей переменным значения с помощью синтаксиса инициализации.

Инициализация структур аналогична инициализации массивов: в фигурных скобках передаются значения для элементов структуры по порядку.

```
1  #include <iostream>
2
3  using namespace std;
4
5  struct xampl {
6      int x;
7  };
8
9  int main()
10 {
11     xampl structure;
12     xampl* ptr;
13
14     structure.x = 12;
15     ptr = &structure;
16     cout << ptr->x;
17     cin.get();
18 }
19
```

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\Дисциплины\Программиро

12

4. Битовые поля

Битовые поля – возможность, позволяющая работать с отдельными битами.

Некоторые причины полезности битовых полей:

- ограниченное место для хранения информации и возможность сохранить несколько логических (истина/ложь) переменных в одном байте;
- передача битовой информации, закодированной в один байт;
- процедурам кодирования необходимо получать доступ к отдельным битам в байте.

Использование битовых полей для доступа к битам основано на структурах. Битовое поле – особый тип структуры, определяющей, какую длину имеет каждый член.

Используя битовые поля, можно упаковать целочисленные компоненты очень плотно, обеспечив максимальную экономию памяти.

Набор разрядов целого числа можно разбить на битовые поля, каждое из которых выделяется для определенной переменной. При работе с битовыми полями количество битов, выделяемое для хранения каждого поля, отделяется от имени двоеточием.

Стандартный вид объявления битовых полей :

```
struct имя структуры {  
    тип имя1: длина;  
    тип имя2: длина;  
    ...  
    тип имяN: длина;  
}
```

При работе с битовыми полями нужно внимательно следить за тем, чтобы значение переменной не потребовало памяти больше, чем под неё выделено.

Битовые поля должны объявляться как `int`, `unsigned` или `signed`.
Битовые поля длиной 1 должны объявляться как `unsigned`,
поскольку 1 бит не может иметь знака.

Пример структуры, содержащей три переменные по одному биту каждая:

```
struct device {  
    unsigned active : 1;  
    unsigned ready : 1;  
    unsigned xmt_error : 1;  
} dev_code;
```



К каждому полю структуры обращение осуществляется с помощью оператора «`.`». Если обращение к структуре происходит с помощью указателя, то следует использовать оператор «`->`».

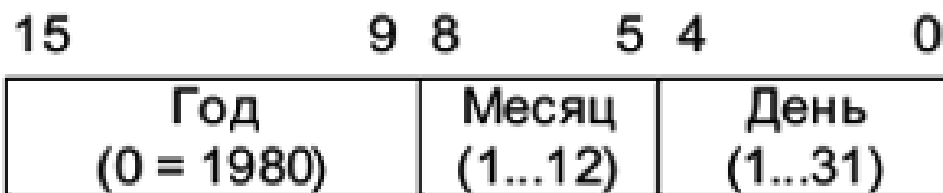
Пример программы упаковки даты в битовые поля

```
#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() {
    setlocale(LC_ALL, "Russian");
    struct date today;
    today.day = 16;
    today.month = 12;
    today.year = 2021 - YEAR0; //today.year = 41
    printf("\n Сегодня %u.%u.%u \n", today.day, today.month, today.year + YEAR0);
    printf("\n Размер структуры today : %d байт", sizeof(today));
    printf("\n Значение элемента today = %hu = %hx шестн.", today, today);
    return 0;
}
```



Сегодня 16.12.2021

Размер структуры today : 2 байт

Значение элемента today = 21392 = 5390 шестн.

Process exited after 0.1975 seconds with return value 0
Для продолжения нажмите любую клавишу . . .

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <iostream>
4  #define YEAR0 1980
5  struct date {
6      unsigned short day : 5;
7      unsigned short month : 4;
8      unsigned short year : 7;
9  };
10 int main() {
11     setlocale(LC_ALL, "Russian");
12     struct date today;
13     today.day = 16;
14     today.month = 12;
15     today.year = 2021 - YEAR0; //today.year = 41
16     printf("\n Сегодня %u.%u.%u \n", today.day, today.month, today.year + YEAR0);
17     printf("\n Размер структуры today : %d байт", sizeof(today));
18     printf("\n Значение элемента today = %hu = %hx шестн.", today, today);
19     return 0;
20 }
```

Сегодня 16.12.2021

Размер структуры today : 2 байт

Значение элемента today = 21392 = 5390 шестн.

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программирование-2 семестр

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр

Нажмите любую клавишу, чтобы закрыть это окно:

Нет необходимости называть каждое битовое поле. К полю, имеющему название, легче получить доступ:

```
struct device {  
    unsigned active : 1;  
    unsigned ready : 1;  
    unsigned xmt_error : 1;  
    unsigned : 2;  
    unsigned EOT : 1;  
} dev_code;
```

В битовых полях можно смешивать различные структурные переменные. Например:

```
struct emp {  
    struct addr address;  
    float pay;  
    unsigned lay_off:1;  
    unsigned hourly:1;  
    unsigned deductions:3;  
};
```

Битовые поля имеют некоторые ограничения:

- Нельзя получить адрес переменной битового поля.
- Переменные битового поля не могут помещаться в массив.
- Переходя с компьютера на компьютер нельзя быть уверенным в порядке изменения битов (слева направо или справа налево).
- Любой код, использующий битовые поля, зависит от компьютера.

5. Объединения `union`

Объединение (`union`) представляет собой частный случай структуры, все поля которой располагаются по одному и тому же адресу.

Формат описания такой же, как у структуры, только вместо ключевого слова `struct` используется слово `union`.

Объединения – сложный тип данных, позволяющий размещать в одном и том же месте оперативной памяти данные различных типов.

Размер оперативной памяти объединения определяется размером памяти данных того типа, который требует максимального количества байт.

Элемент меньшей длины объединения использует только часть отведенной памяти. Все элементы объединения хранятся в одной и той же области памяти, начиная с одного адреса.

```
union ИмяОбъединения
{
    тип ИмяОбъекта1;
    тип ИмяОбъекта2;
    ...
    тип ИмяОбъектаN;
};
```

имя 1			
имя 2			
имя 3			

Как и для структур, можно объявить переменную, поместив ее имя в конце определения или используя отдельный оператор объявления.

union ИмяОбъединения структурные_переменные;

Когда объявлено объединение, компилятор автоматически создает переменную достаточного размера для хранения наибольшей переменной, присутствующей в объединении.

Для доступа к членам объединения используется синтаксис, применяемый для доступа к структурам - с помощью операторов «.» и «->».

Чтобы работать с объединением напрямую, надо использовать оператор «.».

Если к переменной объединения обращение происходит с помощью указателя, надо использовать оператор «->».

Использование объединений помогает создавать машинно-независимый (переносимый) код. Поскольку компилятор отслеживает настоящие размеры переменных, образующих объединение, уменьшается зависимость от компьютера. Не нужно беспокоиться о размере целых или вещественных чисел, символов или чего-либо еще.

Объединения часто используются при необходимости преобразования типов, поскольку можно обращаться к данным, хранящимся в объединении, совершенно различными способами.

Инициализировать объединение при его объявлении можно только заданием значения первого элемента объединения.

Например:

```
union unionA {  
    char ch1;  
    float f1;} a1={ 'M' };
```

Доступ к элементам объединения, аналогично доступу к элементам структур, выполняется с помощью операторов . и ->.

Например:

```
union TypeNum  
{ int i;  
  long l;  
  float f;  
};  
union TypeNum vNum = { 1 };  
// Инициализация первого элемента объединения i = 1  
cout<< vNum.i;  
vNum.f = 4.13;  
cout<< vNum.f;
```

Элементы объединения не могут иметь модификаторов доступа и всегда реализуются как общедоступные (public).

Объединения применяются для следующих целей:


- для инициализации объекта, если в каждый момент времени только один из многих объектов является активным;
- для интерпретации представления одного типа данных в виде другого типа.

Пример использования объединения для представления вещественное числа типа float в виде совокупности байтов:

```
#include <stdio.h>
#include <stdlib.h>
union types {
    float f;
    unsigned char b[4];
};
int main() {
    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]);
    return 0;
}
```

```
N = 1.5467
1.546700 = 44 fa c5 3f
-----
Process exited after 5.705 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  union types {
4      float f;
5      unsigned char b[4];
6  };
7  int main() {
8      types value;
9      printf("N = ");
10     scanf_s("%f", &value.f);
11     printf("%f = %x %x %x %x", value.f, value.b[0], value.b[1], value.b[2], value.b[3]);
12     return 0;
13 }
14
```

100 %  Проблемы не найдены.

Вывод

Показать выходные данные из: Отладка



"ConsoleApplication9.exe" (Win32). Загружено "C:\Windows\System32\kernel.appcore.dll".

"ConsoleApplication9.exe" (Win32). Загружено "C:\Windows\System32\user32.dll".

N = 1234355

1234355.000000 = 98 ad 96 49

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программирование-2 с

Чтобы автоматически закрывать консоль при остановке отладки, включите пар

Нажмите любую клавишу, чтобы закрыть это окно:

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.

СПАСИБО ЗА ВНИМАНИЕ!