Элементы ЯПВУ
Пользовательские типы данных

C / C++

В C / C++ есть пять способов создания пользовательских типов данных:

- **Структура** – совокупности переменных разных типов, объединённых под одним именем
- **Объединения** – позволяют использовать один и тот же участок памяти для хранения переменных разных типов
- **Битовое поле** – специальный тип элемента структуры или объединения для доступа к отдельным битам
- **Перечисления** – набор именованных целых констант
- **Переименование типов - typedef** – ключевое слово, которое определяет новое имя для существующего типа данных

Элементы ЯПВУ
Структуры

C / C++

C / C++

Структура (struct) состоит из фиксированного числа элементов – полей. В отличие от массивов, поля структуры могут быть различного типа. Объявление (декларирование) структуры описывает шаблон, по которому строятся переменные структуры (при объявлении структуры память не выделяется, она будет выделена при объявлении переменных этого типа структуры). В C++ структура может быть классом и обладать всеми его свойствами.

```
struct     [<имя_типа>]
{ <тип_поля_1> <имя_поля_1>;
  . . . . . .
 <тип_поля_N> <имя_поля_N>;
} [<переменные_структуры>];
```

Могут быть опущены или тег структуры, или её переменные, но не оба одновременно.

где – **имя_типа** – ***идентификатор типа структуры (тег структуры)*** ,

– **struct** – ***ключевое слово описания структуры,***

– **тип_поля_N имя_поля_N** – ***список описаний N полей, разделенных точкой с запятой*** – **char fio[60];**

Поля структуры могут быть любого типа, кроме типа этой же структуры, но могут быть указателями на него.

– **переменные_структуры** – ***список переменных или указателей.***

Формат объявления переменных структуры:

```
struct  <имя_типа>  <список_переменных_структуры> ;
```

Элементы ЯПВУ
Структуры

C / C++

C / C++

Примеры объявления структур

Вариант 1 – объявление структуры с одновременным объявлением переменных:

```
struct addr
{  char name[40];
   char street[40];
   char city[20];
   char state[3];
   unsigned long int zip, date;
}    addr_info, binfo, dinfo;
```

Вариант 2 – объявление структуры без объявления переменных, а затем объявление переменных отдельно:

```
struct addr
{  char name[40];
   char street[40];
   char city[20];
   char state[3];
   unsigned long int zip, date;
};
struct addr addr_info, dibfo;
```

Вариант 3 – объявление структуры с одновременным объявлением только одной переменной (тег опускается):

```
struct
{  char name[40];
   char street[40];
   char city[20];
   char state[3];
   unsigned long int zip, date;
}    addr_info;
```

Вариант 4 – объявление массива структур и указателя на структуру:

```
struct
{  char name[40];
   char street[40];
   char city[20];
   char state[3];
   unsigned long int zip, date;
}  addr_inf[100], *ps;
```

Элементы ЯПВУ
Структуры

C / C++

C / C++

Инициализация структуры

```
struct
{  char fio[40];
   int date, zip;
   double salary;
} worker={"КалмыковВ.", 31, 249030,  6.400,00};
```

Для инициализации структуры значения её элементов (полей) перечисляют при объявлении переменной в фигурных скобках в порядке их описания.

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

При инициализации массивов структур надо заключать в фигурные скобки каждый элемент массива, т.к. многомерный массив – это массив массивов.

```
// строка 1, т.е. массив compl[0]
// строка 2, т.е. массив compl[1]
```

Доступ к полям структуры

Доступ к элементам структуры осуществляется через операторы:

. (точка) - **имя_структуры.имя_поля** - доступ через имя структуры.

Пример: addr.sity = "Омск";

-> (стрелка) - **имя_указателя->имя_поля** - доступ через указатель.

Пример: ps->street = "Онежская";

Элементы ЯПВУ
Структуры

C / C++

C / C++

Присваивание структур

При выполнении операции присваивания для переменных одинакового структурного типа происходит поэлементное копирование, нет необходимости присваивать значения каждого элемента в отдельности.

Пример:

```
#include <stdio.h>
int main(void)
{
  struct
   {  int a;
      int b;
   }    x, y;
  x.a = 10;
  y = x;  // присваивание одной структуры другой
  printf ("%d", y.a);
  return 0;
}
```

После присвоения в y.a хранится значение 10.

Массивы структур

Вначале объявляют структура, а затем объявляют массив этого же типа.

struct addr addr_list[100]; - создает 100 наборов переменных, каждый организован по структуре addr.

Для получения доступа к определенной структуре указывается имя массива с индексом:

printf ("%1u"', addr_list[2].zip);

Индексирование массива начинается с 0.

Пример: в результате выполнения выражения addr_list[2].name[0] = 'X'; первому символу поля name, находящегося в третьей структуре из addr_list, присваивается значение 'X'.

Указатели на структуры

srtuct addr *addr_pointer;

Они используются обычно в двух случаях: когда структура передается функции с помощью вызова по ссылке, и когда создаются связанные списки и др. структуры с динамическими данными.

Элементы ЯПВУ

Объединения

C / C++

C / C++

Объединения (union) – это частный случай структуры, все поля которой расположены по одному адресу. Размер объединения равен размеру наибольшего из его полей. В каждый момент времени в переменной типа объединение хранится только значение одного из полей (какое именно должен отслеживать программист). Объединение используют для экономии памяти, если точно известно, что больше одного поля одновременно не обрабатывается.

```
union     [<имя_типа>]
{ <тип_поля_1> <имя_поля_1>;
  . . . . . .
  <тип_поля_N> <имя_поля_N>;
} [<переменные_объединения>];
```

где – **имя_типа** – *идентификатор типа объединения (mmemeг) ,*

– **union** – *ключевое слово описания объединения,*

– **тип_поля_N имя_поля_N** – *список описаний N полей, разделенных точкой с запятой* – **int dh;**.

Формат объявления переменных объединения:

```
union  <имя_типа>  <список_переменных_объединения>;
```

Объединения часто используют в качестве поля структуры, а так же когда надо выполнить специфическое преобразование типов, например используя объединения можно манипулировать байтами, составляющими значение типа double, чтобы менять его точность или выполнять какое-либо необычное округление.

Элементы ЯПВУ
Битовые поля

C / C++

C / C++

Битовые поля – это особый вид полей структуры. Они позволяют получить доступ к единичному биту и используются для плотной упаковки булевых переменных, обработки информации от аппаратных устройств, в процедурах шифрования и пр.
Битовое поле может быть членом структуры или объединения и сочетаться с другими типами.

Формат описания битового поля:

<имя_типа> <имя_поля> : <длина_поля>

где – **имя_типа** – *идентификатор типа битового поля,*
– **имя_поля** – *идентификатор битового поля,*
– **длина_поля** – *количество бит, занимаемое полем (целая положительная константа).*

Тип битового поля может быть:
- int,
- signed,
- unsigned,
- bool (в C++).

Пример: байт состояния адаптера модема

```
struct status_type
{ unsigned delta_cts : 1;
  unsigned delta_dsr : 1;
  unsigned tr_edge : 1;
  unsigned delta_rec : 1;
  unsigned cts : 1;
  unsigned dsr : 1;
  unsigned ring :1;
  unsigned rec_line :1;
}  status;
```

Присвоение значения битовому полю:

```
status.ring = 0;
```

Если нужны не все поля – ненужные до нужных объявляются как одно поле, но не именуются, а после - опускаются.

```
struct status_type
{ unsigned : 4;
  unsigned cts : 1;
  unsigned dsr : 1;
}  status;
```

Ограничения: нельзя получить адрес битового поля; нет массивов битовых данных; и др.

Элементы ЯПВУ
Перечисления

C / C++

C / C++

Перечисления (enum) – это набор именованных констант.

```
enum    [<имя_типа>]
{ <список_констант>}
[<переменные_перечисления>];
```

где – **имя_типа** – *идентификатор типа объединения (ттетег) ,*

– **enum** – *ключевое слово описания перечисления,*

– **список_констант** – *список целочисленных констант. При отсутствии инициализатора значение первой константы = 0, остальных +1.*

Формат объявления переменных:

```
enum <имя_типа>
<переменные_перечисления>;
```

Примеры: enum coin {penni, nickel, dime, quarter, half_dollar, dollar};

enum coin money; **а значить можно выполнить:**

money = dime; if (money == quarter) printf("Денег – четверть доллара");

Оператор printf ("%d %D", penny, dime); **выведет на экран 0 и 2.**

Инициализация: enum coin {penny=2, nickel, dime, quarter=100, half_dollar, dollar};

Значения элементов будут:

penny	2
nickel	3
dime	4
quarter	100
half_dollar	101
dollar	102

Имена перечисляемых констант должны быть уникальными, а значения могут совпадать.

При выполнении арифметических операций перечисления преобразуются в целые числа.

Элементы ЯПВУ. Переименование типов

C / C++

C / C++

Переименование типов (typedef) – задание типу данных нового имени. Новый тип при этом не создается, а определяется новое имя (синоним) для уже существующего типа.

typedef <тип> <новое_имя_типа> [<размерность>]

где* – тип – *любой тип данных языка С,
– новое_имя_типа – *новое имя для этого типа,*
– typedef – *ключевое слово переименования типа,*
– размерность – *размерность нового типа данных.*

Пример:

```
typedef char SIMB[40]; //SIMB - массив из сорока символов
  SIMB person; //переменная person - тоже массив из сорока символов
```

Это эквивалентно объявлению - char person[40];.

typedef используется для:

1. **Задания типам с длинными описаниями (именами) коротких псевдонимов.**
2. **для облегчения переносимости программ – если машинно-зависимые типы данных объявить через typedef, то при переносе программы надо изменить только эти операторы.**

Элементы ЯПВУ
Пользовательские типы данных

C / C++

C / C++

Практические занятия

Найти ошибки

Пример 3

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
   struct  growth {char *name;
                   int  value;};
    growth man;
    man->value = 45;
    man->name = "Anton";
  cout<<man->name<< " - "
       <<man ->value;
}
```

Неправильное объявление элемента структуры.

Пример 4

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
     struct  book  {char  title[10];  char
auther[10]; };      struct  book  *lib;
   gets (book.title);
   gets (book.auther);
cout     <<book.auther     <<     ".     "
<<book.title;
}
```

Неправильное объявление элемента структуры.

Элементы ЯПВУ
Пользовательские типы данных

C / C++

C / C++

Практические занятия

Объяснить тексты программ

Пример 1

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
   struct  man {int value; char name;};
    man my;
    my.value = 45;
    my.name = 'Anton';
  cout<<my.name<< " - "<<my.value;
}
```

Пример 2

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
  struct  my_struct {int a; char b[10];};
  struct  new_struct
   {  int b;
       my_struct c;
   };
    new_struct f[2];
  f[1].c.b[5] = 'g';
}
```

Элементы ЯПВУ
Пользовательские типы данных

C / C++

C / C++

Практические занятия
Найти и объяснить ошибки

Пример 5

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
   struct  my
{  int s;
   struct {char b};};
   struct example(int g;);
};
}
```

Попытка вложенного объявления структур.

Пример 6

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
  struct  example
  {  int b=2;
       char c='a';
   };
}
```

Попытка инициализировать поля структуры в декларации.

Элементы ЯПВУ.
Пользовательские типы данных

C / C++

C / C++

Практические занятия

Найти, объяснить, исправить ошибку

Пример 7

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
   struct  name
{    char a[5];
       int b;
       char c[10];   };
  name my = { 'a', 1, "123456"};
}
```

Компилятор ожидает 5 значений для инициализации первого поля – символьного массива. При попытке использовать последующие значения возникает проблема соответствия типов.

Пример 7'

```
#include <iostream.h>
#include <conio.h>
void main()
{   clrscr();
   struct  name
{    char a[5];
       int b;
       char c[10];   };
  name my = {{'a'}, 1, "123456"};
}
```

А в этом примере фигурные скобки {'a'} указывают, что для массива есть только один инициализатор, а остальные выражения предназначены для инициализации прочих полей структуры.