ЕН.Ф.02 – Информатика и программирование

Лекция 6. Конструируемые типы данных. Массивы

Конова Елена Александровна
E_Konova@mail.ru

План

Классификация типов данных

Примерная классификация типов данных C++:

1) базовые типы – предопределенные, стандартные, встроенные, примитивные – определены стандартом языка;

2) конструируемые – строятся на основе других типов, не обязательно базовых – определяются программистом под нужды конкретной задачи, и нуждаются в предварительном построении (описании).

Классификация производных типов

1. Функции. Функция возвращает значение, обладающее типом, значит, она есть частный случай типа.

2. Массивы.

3. Указатели − инструмент для работы с адресами памяти.

4. Строки − частный случай массива, к которому прилагается библиотека функций обработки строк.

5. Структуры − для объединения в одно целое данных разных типов.

6. Объединения.

7. Классы − инструмент объектного проектирования и программирования.

Массив как производный тип

Массив – упорядоченное множество однородных величин, объединенных общим именем.

Важно! Все элементы массива имеют один и тот же тип.

Тип элементов массива – почти любой, не обязательно базовый тип, например,

- массив целых чисел;
- массив координат точек на плоскости, один элемент массива хранит пару значений координат (x, y);
- массив сведений о студенте, один элемент массива хранит разнородную информацию – Имя, Фамилия, Рост, Вес, Цвет_глаз и т.д.

Обязательные атрибуты массива

Строя массив, необходимо определить атрибуты массива, связанные с механизмами выделения памяти:

1) размерность (стандарт C++ – 1-мерные массивы, векторы данных);

2) число элементов (знать, сколько данных);

3) тип элементов (знать, сколько выделить памяти).

Объявление массива

Назначение объявления массива:

1) конструирование нового типа;

2) инициализация (необязательна) − присваивание значений элементам массива.

Синтаксис описания массива:

Тип Имя_массива [Кол_во_элементов];

// Возможно, список инициализации

Здесь

Кол_во_элементов – константная величина, то есть константа в чистом виде или `define` константа.

Тип – практически любой тип данных.

Примеры описания массива

```
#define N 100
...
int  Arr[4];          // 4 int значения, все имеют имя Arr
int  Matr[2][3];      // Массив массивов
float W[N];   // N=100
char  Symb[N][80];    // Текст N строк, по 80 символов)
```

Внимание! Элементы массива нумеруются с 0, таким образом, для

```
int Arr[4];
```

имеются элементы `Arr[0], Arr[1], Arr[2], Arr[3]`

Размещение массивов в памяти

Имя массива сопоставлено всей совокупности данных.

Элементы массива размещаются подряд в соответствии с ростом номера элемента внутри массива (индекса).

В C++ нет контроля выхода за границу массива.

Пусть объявлен и инициализирован массив:

```
int Arr[4] = {10,20,30,40};
int i; // Для нумерации элементов массива
```

Пример – размещение в памяти.

Операция sizeof

sizeof позволяет определить общий размер памяти, занимаемой:

а) данным любого типа (в том числе конструируемого),

б) типом данного.

Имя типа или имя объекта указываются в параметре.

Примеры:

```
sizeof(x)          // x, это имя переменной
int Count = sizeof(Arr)   // в байтах
sizeof(long int)   // long int , это имя типа
```

Операции с массивами

Никаких операций с массивами не разрешено, кроме операции обращения к элементам массива [], синтаксис которой:

Операнд_1 [Операнд_2]

Фактически, это составное имя:

Имя_массива [Индекс]

Семантика операции [] позволяет выделить один элемент с указанным номером.

Индекс – выражение целого типа, определяющее номер элемента внутри массива (счет с 0).

Инициализация элементов массива

Инициализация, это присвоение значения при объявлении, например:

```
int a=99;
```

Для массива можно задать список инициализирующих значений.

```
int month [12] =
{31,28,31,30,31,30,31,31,30,31,30,31};
```

Удобно использовать инициализацию:

а) если значения не меняются,

б) при отладке.

Эквивалент – присваивание вида:

```
month[0] = 31;      // Январь
          .. и т.д.
month[11] = 31;     // Декабрь
```

Возможны варианты

```
int month []    =
{31,28,30,31,30,31,30,31,30,31};
        //Количество = 12.
int month [12] = {31,28,31,30};
        // Выделено 12, остальные = 0
int month [2]  = {31,28,31,30};
        // Ошибка!
```

Пример.

Как определить число элементов

Пусть есть объявление:

```
int a[] = {1,2,3,4,5,6,7,8,9,10,11,12};
// Сколько элементов в массиве?
int len_a;
len_a = sizeof(a)/sizeof(int);
```

Пример.

Управление обработкой массивов

Как правило, при обработке данных массива, действия применяются ко всем элементам массива по очереди.

Управление выполняется в циклах, где управляющей переменной является индекс элемента массива.

Пусть

```
#define N 10
...
int Arr[N];
```

Индекс меняется от `0` до `N-1`, приращение = 1.

Синтаксис управления

```
// Здесь N = число элементов  i < N
for (int i=0; i<N; i++)
   {
      // обращение к Arr[i]
   }
for (int i=N-1; i>=0; i--)
   {
      // обращение к Arr[i]
   }
```

Массивы переменной длины

Один из недостатков массивов – длина статического массива жестко задана в программе.

Выходы:

1) `#define` определенные константы;

2) массив условно переменной длины;

3) динамические массивы.

Кроме того, использование функций обработки массивов, которые решают задачу «в общем виде», принимая абстрактный массив произвольной длины.

#define константы как длина массива

Механизм #define константы, это изменение текста программы перед ее компиляцией.

Длина массива записывается в директиве #define, например:

```
#define N  20
```

Имя N – макроподстановка. Числовое значение константы 20 записывается в тексте один раз.

В тексте программы для управления алгоритмами обработки массива используется имя N.

Перед очередным запуском программы может быть изменено один раз, остальные изменения выполнит препроцессор. После этого программа нуждается в повторной компиляции и сборке.

Пример.

Массивы условно переменной длины

Длина массива может быть оценена заранее. Это, возможно, наибольшее значение, следует выбрать для описания массива.

Чтобы знать реальную длину массива, вводится специальная переменная, которая принимает значение при выполнении программы, а затем использует его для управления алгоритмами обработки массива.

Пример.

Оба приема можно совместить.

Функции работы с массивами

Функции работы с массивами:

1) решают задачу обработки массива «в общем виде»;
2) получают массив через параметры «в общем виде»;
3) могут обрабатывать массив любой длины.

Случайное присваивание значений

Используется функция `random`, прототип которой:

```
int random(int число).
```

Диапазон генерируемых значений (от 0 до «Число–1»)

Для запуска генератора случайных чисел используется функция `randomize()`, которая вызывается один раз при старте программы (от системного времени), ее прототип в `stdlib.h` и `time.h`

Некоторые алгоритмы работы с одномерными массивами

1. Вывод элементов массива.
2. Ввод элементов массива.
3. Прямой поиск − поиск первого, последнего, любого, поиск с флагом.
4. Суммирование и его клоны.
5. Алгоритмы с изменением длины массива − вставка и удаление.
6. Сортировки.

И так далее.

Примеры реализации алгоритмов работы с массивами.

Примеры алгоритмов работы

Примеры.