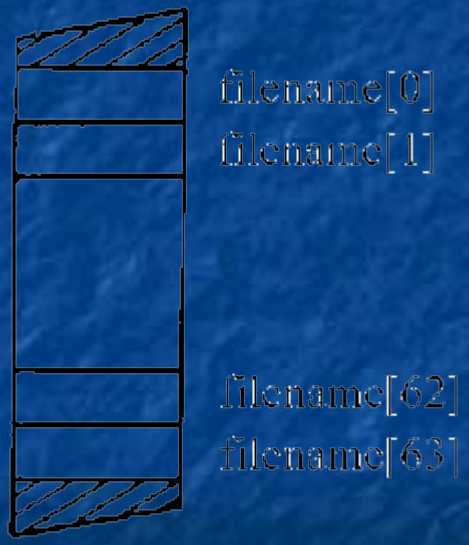
# Символьные строки

char filename[64];

- *Символьная строка представляет собой массив символов, завершающийся 0 (символом NULL).*
- *используют символьные строки для хранения имен пользователей, имен файлов и другой символьной информации.*



# Символьные строки

```
#include <iostream.h>
void main(void)
{
   char alphabet [34]; // 33 буквы плюс NULL char letter;
   int index;
   for (letter = 'A', index = 0; letter <= 'Я';
   letter++, index++) alphabet[index] = letter;
   alphabet[index] = NULL;
   cout << "Буквы " << alphabet;
}
```

Когда цикл *for* инициализирует или увеличивает несколько переменных, разделяйте операции запятой (запятая тоже является оператором C++)

# Как 'A' отличается от "A"

- двойные кавычки ("A"). Символ внутри одинарных кавычек представляет собой *символьную константу.* Компилятор C++ выделяет только один байт памяти для хранения символьной константы. Однако символ в двойных кавычках представляет собой *строковую константу* — указанный символ и символ NULL (добавляемый компилятором). Таким образом, компилятор будет выделять два байта для символьной строки.
- char title[64] = "Учимся программировать на языке C++";
- #include <iostream.h>
- void main(void)
- {
  char title[64] = "Учимся программировать на языке C++";
  char lesson[64] = "Символьные строки";
  cout << "Книга: " << title << endl;
  cout << "Урок: " << lesson << endl;
  }

# ПЕРЕДАЧА СТРОК В ФУНКЦИИ

- Программа передает несколько различных символьных строк в функцию, отображая длину каждой из них на экране:
- #include <iostream.h>
- int string_length(char string[])
- {
  int i;
  for (i = 0; string[] != '\0'; i++); // Ничего не делать, но перейти к
  // следующему символу return(i); Длина строки
  }
- void main(void)
- {
  char title[] = "Учимся программировать **на** языке C++";
  char lesson[] = "Символьные строки";
  cout << "Строка " << title << " содержит " << string_length(title) << " символов" **<< endl;**
  cout << "Строка " << lesson << " содержит " << string_length(lesson) << " символов" << endl;
  }

# Хранение связанной информации в структурах

- struct name
- {
  int member_name_l **; |** ———————— *Объявления элементов структуры*
  float member_name_2; } variable; **|** ————————————————
  *Объявление переменной*
  }
- Следующее определение создает структуру, содержащую информацию о служащем:
- struct employee
- {
  char name [64] ;
  long employee_id;
  float salary;
  char phone[10];
  int office_number;
  };

# СТРУКТУРЫ И ФУНКЦИИ

**Если функция не изменяет структуру, вы можете передать структуру в функцию по имени. Используем функцию *show_employee для* вывода элементов структуры типа *employee:***

```
#include <iostream.h>
#include <string.h>
struct employee
{
   char name[64];
   long employee_id;
   float salary;
   char phone[10];
   int office_number;
};
void show_employee(employee worker)
{
   cout << "Служащий: " << worker.name << endl;
   cout << "Телефон: " << worker.phone << endl;
   cout << "Номер служащего: " << worker. employee_id << endl;
   cout << "Оклад: " << worker.salary << endl;
   cout << "Офис: " << worker.office_number << endl;
}
void main(void)
{
   employee worker;
// Копировать имя в строку strcpy(worker.name, "Джон Дой");
   worker.employee_id = 12345;
   worker.salary = 25000.00;
   worker.office_number = 102;
// Копировать номер телефона в строку strcpy(worker.phone, "555-1212");
   show_employee(worker);
}
```

# СТРУКТУРЫ И ФУНКЦИИ

- если функция изменяет параметр, вам следует передавать этот параметр в функцию с помощью адреса. Если функция изменяет элемент структуры, вы должны передавать эту структуру в функцию с помощью адреса, Для передачи переменной типа структуры с помощью адреса вы просто предваряете имя переменной оператором адреса C++ (&), как показано ниже:
- **some_function(&worker);**
- Внутри функции, которая изменяет один или несколько элементов, необходимо работать с указателем. Если использовать указатель на структуру, то легче всего обращаться к элементам структуры, используя следующий синтаксис:
- **pointer_variable->member = some_value;**

# Объединения

- Объединения C++ очень похожи на структуры, за исключением того, как C++ хранит их в памяти; кроме того, объединение может хранить значение только для одного элемента в каждый момент времени.
- Объединение представляет собой структуру данных, подобную структуре C++, и состоит из частей, называемых элементами.
- Объединение определяет шаблон, с помощью которого программы далее объявляют переменные.
- Для обращения к определенному элементу объединения ваши программы используют оператор C++ точку.
- Чтобы изменить значения элемента объединения внутри функции, ваша программа должна передать переменную объединения в функцию с помощью адреса.
- Анонимное объединение представляет собой объединение, у которого нет имени (тэга).

# Объединения

- **следующая структура определяет объединение с именем *distance,* содержащее два элемента:**
- *union distance*
- *{*
  *int miles;*
  *long meters;*
  *};*
- **Чтобы объявить переменную объединения, можно использовать любой из следующих форматов:**
- *union distance*
- *{*
  *union distance*
- *{*
  *int miles; int miles;*
  *long meters; long meters;*
  *} japan, germany, franee;*
  *};*
  *distance japan, germany, franee;*

# Объединения

- *Анонимное объединение представляет собой безымянное объединение. Анонимные объединения обеспечивают вашим программам способ экономии памяти, и при этом можно не использовать имя объединения и точку. Следующие олераторы определяют анонимное объединение, способное хранить две символьные строки:*
- *union*
- *{*

```
    char short_name[13];
    char long_name[255];
};
```

# ПРЕДСТАВЛЕНИЕ ОБ ОБЪЕКТАХ И ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

**В C++ вы используете класс для определения своих объектов. Ваша цель состоит в том чтобы включить в класс столько информации об объекте, сколько требуется. Исходя из этого, можно подобрать класс, созданный вами для одной программы, и использовать его в нескольких разных программах.**

**Класс позволяет вашим программам группировать данные и функции которые выполняют операции над этими данными. Большинство книг и статей об объектно-ориентированном программировании называют функции класса методами. Подобно структуре, класс C++ должен иметь уникальное имя, за которым следует открывающая фигурная скобка, один или несколько элементов и закрывающая фигурная скобка**:

```
class class_name
{
   int data_member; // Элемент данных
   void show_member(int); // Функция-элемент
};
```

**После определения класса вы можете объявлять переменные типа этого класса (называемые *объектами),* как показано ниже:**

**class_name object_one, object_two, object_three;**

**Следующее определение создает класс *employee,* который содержит определения данных и метода**:

```
class employee
{
   public:
   char name[64] ;
   long employee_id;
   float salary;
   void show_employee(void)
   {
      cout << "Имя: " << name << endl;
      cout << "Номер служащего: " << employee_id << endl;
      cout << "Оклад: " << salary << endl;
   };
};
```

# ПРЕДСТАВЛЕНИЕ ОБ ОБЪЕКТАХ И ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

- программа EMPCLASS.CPP создает два объекта employee. Используя оператор точку, программа присваивает значения элементам данных Затем программа использует элемент *show_employee* для вывода информации о служащем:

```
#include <iostream.h>
#include <string.h>
class employee
{
public:
  char name [64];
  long employee_id;
  float salary;
  void show_employee(void)
  {
    cout << "Имя: " << name << endl;
    cout << "Номер служащего: " << employee_id << endl;
    cout << "Оклад: " << salary << endl;
  };
};
void main(void)
{
  employee worker, boss;
  strcpy(worker.name, "John Doe");
  worker.employee_id = 12345;
  worker.salary = 25000;
  strcpy(boss.name, "Happy Jamsa");
  boss.employee_id = 101;
  boss.salary = 101101.00;
  worker.show_employee();
  boss.show_employee();
}
```

# ОПРЕДЕЛЕНИЕ МЕТОДОВ КЛАССА ВНЕ КЛАССА

- class employee
- {
  public:
  char name[64];
  long employee_id;
  float salary;
  void show_employee(void); |————————> *Прототип функции*
  };

- Так как разные классы могут использовать функции с одинаковыми именами, вы должны предварять имена определяемых вне класса функций именем класса и оператором глобального разрешения (::). В данном случае определение функции становится следующим:
- void employee:: show_employee (void) //---------->Имя класса
- {
  cout << "Имя: " << name << endl; Имя элемента cout << "Номер служащего: " << employee_id << endl;
  cout << "Оклад: " << salary << endl;
  };