Введение в объектно-ориентированное программирование

Перегрузка функций, шаблоны функций и шаблоны классов

Перегрузка функций не является отличительной особенностью объектно-ориентированного программирования. Однако, в ООП, она используется очень часто, например при перегрузке в классе операций (действий), присущих стандартным типам данных.

Введение в объектно-ориентированное программирование

Часто бывает удобно, чтобы функции, реализующие один и тот же алгоритм для различных типов данных, имели одно и то же имя. Использование одного и того же имени для нескольких функций, но имеющих различную сигнатуру, в C++ называется перегрузкой.

Введение в объектно-ориентированное программирование

Рассмотрим пример известной функции swap, меняющую местами значения своих аргументов.

```
void swap(int &a,  int &b)
{
   int temp;
   temp = a;
   a = b;
   b = temp;
}
```

Введение в объектно-ориентированное программирование

Эта функция работает с целым типом данных, ее можно перегрузить для других типов, указав эти типы в качестве параметров. Например, для вещественных чисел или символов.

```
void swap(double &a,  double &b)
{
    double temp;
    temp = a;
    a = b;
    b = temp;
}
```

Введение в объектно-ориентированное программирование

```
void swap(char &a,  char &b)
{
   char temp;
   temp = a;
   a = b;
   b = temp;
}
```

Введение в объектно-ориентированное программирование

В рассмотренном примере есть три функции с одинаковыми именами, но с различными типами входных параметров. Это и есть перегрузка. Перегрузка может быть осуществлена с одним типом параметров, но их количество должно отличаться.

При вызове перегруженной функции компилятор в первую очередь определяет тип фактических параметров, после чего вызывает

Введение в объектно-ориентированное программирование

конкретный экземпляр функции. Как уже было сказано, перегрузка функций считается как слабая форма полиморфизма, то есть, вызывается та функция, типы аргументов которой совпадают с типами фактических параметров.

Введение в объектно-ориентированное программирование

Дальнейшим развитием перегружаемых функций стали шаблоны функций, которые позволяют вместо множества перегружаемых функций определять один единственный экземпляр функции, работающий с любым типом данных.

Появилась возможность параметризировать алгоритм, то есть, сделать его не зависящим от типов фактических параметров.

Введение в объектно-ориентированное программирование

Общий формат шаблона функции следующий:

```
template<class Type_Name> тип_рез_та
имя_функции(список аргументов)
{
   // тело функции
}
```

Введение в объектно-ориентированное программирование

Ключевое слово template сообщает компилятору, что это не просто функция, а шаблон функции. Слово class в данном контексте заменимо на слово typename. В угловых скобках указывается список шаблона. Количество типов в списке стандартом не ограничено, перечисляются они через запятую. Далее идет определение подобное определению обычной функции.

Введение в объектно-ориентированное программирование

Теперь вернемся к перегруженным функциям swap , реализуя их через единственный шаблон:

```
template<typename Type>
void swap(Type &a,  Type &b)
{
    Type temp;
    temp = a;
    a = b;
    b = temp;
}
```

Область действия типа Type только в пределах данного блока.

Введение в объектно-ориентированное программирование

Теперь, вместо трех функций у нас есть одно единственное определение, работающее с любым типом данных, поддерживающим данный алгоритм. Работа программиста в этом случае существенно упрощается, с другой стороны нагрузка на компилятор возрастает, так как, для каждого типа он обязан генерировать свой собственный объектный код.

Введение в объектно-ориентированное программирование

Разработчики языка C++ пошли еще дальше – параметризировали тип. Речь идет о шаблонах классов. Для понимания сути шаблонных классов, вернемся к определению обычного класса:

```
class Integer
{
protected:
    int  item;
public:
    Integer(){};
    Integer(int i):item(i){};
```

Введение в объектно-ориентированное программирование

```
Integer(const Integer &);
   ~Integer(){}
   Integer operator -(const Integer&);
   bool operator <(const Integer &);
   Integer &operator =(const Integer &);
   friend ostream &operator <<(ostream &, const
   Integer &);
};
```

Введение в объектно-ориентированное программирование

В этом классе одно единственное поле int item; целого типа. Может появиться необходимость в аналогичном классе, но содержащим поле типа double, char или bool. Можно создать три отдельных класса, а можно воспользоваться услугами шаблонного класса, создав единственный шаблон.

Введение в объектно-ориентированное программирование

Пример такого подхода:

```
template<typename Type>
class Number
{
protected:
    Type  item;
public:
    Number(){};
    Number(Type i):item(i){};
    Number(const Number<Type>&);
    ~Number(){}
    Number operator -(const Number<Type>&);
    bool operator <(const Number<Type> &);
    Integer &operator =(const Number<Type> &);
    friend ostream &operator << <>(ostream &, const  Number<Type>&);
};
```

Введение в объектно-ориентированное программирование

Шаблоны классов очень часто используются в практическом программировании. Силами разработчиков была создана стандартная библиотека шаблонов (STL), используемая во многих приложениях.

Типы данных языка C++

Концепция типов данных

Отметим сразу, что язык C++ является строго типизированным языком, то есть, любой объект данных, объявленный в программе, должен принадлежать какому-либо типу.

Суть любой программы состоит в обработке данных. Данные различных типов обрабатываются по-разному.

Типы данных языка C++

Типы данных определяют:

- *внутреннее представление* данных в программе;
- *множество значений*, которые могут принимать величины (переменные, объекты) данного типа;
- *операции (функции)* , применимые к величинам данного типа.

Типы данных языка C++

Все типы языка делят на основные и составные. В языке C++ есть шесть основных (стандартных) типов, на основе которых строятся описания составных (пользовательских) типов.

Типы данных языка C++

Основные типы данных

Основные (стандартные) типы часто называют арифметическими, поскольку их можно использовать в арифметических операциях. Для их описания используются следующие ключевые слова:

- **int** – целый тип;
- **char** – символьный тип;

Типы данных языка C++

- **wchar_t** – расширенный символьный тип;
- **bool** – логический тип;
- **float** – вещественный тип;
- **double** – вещественный с двойной точностью.

Существует четыре спецификатора, уточняющих внутренне представление и диапазон значений величин:

- short – короткий;
- long – длинный;

Типы данных языка C++

- signed – знаковый;
- unsigned – беззнаковый.

Целый тип (int)

Стандартом языка C++ не оговаривается размер типа int. Он зависит от разрядности процессора и версии компилятора.

С определенной точностью можно сказать, что под данные этого типа компилятор выделяет 4 (четыре) байта.

Типы данных языка C++

Спецификатор short перед именем типа указывает компилятору, что под объект выделяется 2 (два) байта.

Внутреннее представление величин целого типа – это целое число в двоичном представлении, старший разряд которого обозначает знак числа (0 – положительные числа, 1 – отрицательные). По умолчанию числа считаются знаковыми (signed).

Типы данных языка C++

Спецификатор unsigned предназначен для обозначения без знаковых положительных чисел.

Символьный тип (char)

Под величины этого типа компилятор всегда выделяет один байт. Этого достаточно для размещения 256-символьного набора ASCII.

Символы по сути это целые числа в диапазоне от -128 до 127 или от 0 до 255.

Типы данных языка C++

Расширенный символьный тип (wchar_t)

Предназначен для работы с символами для кодировки которых не достаточно 1 байта, например, Unicode. Обычно этот тип занимает 2 байта.

Типы данных языка C++

Логический тип (bool)

Величины этого типа принимают всего два значения true (1) или false (0). Для их хранения выделяется один байт.

Типы данных языка C++

Типы с плавающей запятой (float, double)

Стандарт языка предусматривает три типа данных для хранения вещественных чисел:

float, double и long double. Отличаются они диапазонами представляемых величин и способами хранения их в памяти машины.

В отличие от целочисленных величин, вещественные имеют мантиссу и порядок числа.

Типы данных языка C++

Типы char, bool и int относятся к целочисленным типам.

Типы данных языка C++

Типы данных, определяемые пользователем

Язык C++ позволяет создавать свои собственные типы, ориентированные на решение конкретной задачи.

Переименование типов (typedef)

Переименование по своей сути не является определением типа, оно задает типу новое имя, что делает программу более ясной

Типы данных языка C++

Общий формат переименования следующий:
typedef тип новый_тип [размерность];
Примеры:

```
typedef unsigned int UINT;
typedef struct
{
    char name[10];
    int date;
    int group;
}  Student;
```

Типы данных языка C++

***Перечислимый тип данных (enum)**

В некоторых случаях возникает необходимость определения конечного множества именованных констант с различными значениями. Для этого можно воспользоваться перечислимым типом данных или перечислением. В перечислении данные задаются списком целочисленных констант.

Типы данных языка C++

Общий формат перечисления следующий:

enum [имя_типа]{список констант};

Имя типа задается в случае, если необходимо определять переменные типа перечисление. Компилятор обеспечивает, чтобы эти переменные принимали значения только из списка констант.

Константы могут инициализироваться обычным образом. При отсутствии инициализации первая константа обнуляется, а каждой следующей присваивается значение на единицу большее, чем предыдущее.

Типы данных языка C++

Примеры:

```
enum Color {red, blue, green};

int main()
{
    Color color;
    color = red;
    cout << " Color: " << color << endl;

    //color = 1;ошибка!!!!
    color = static_cast<Color>(1);
    cout << " Color: " << color << endl;

    system("pause");
    return 0;
}
```

Типы данных языка C++

Следующее объявление задает инициализацию:

```
   enum Color {red=3, blue=5, green};
//
   color = green;
   cout << " Color: " << color << endl;
// ошибка !!!
// color = 15;
```

Типы данных языка C++

К объектам перечислимого типа можно применять все операции, допустимые к целочисленным типам, например, арифметические:

```
int var_int(20);
Color color;
color = blue;
cout << var_int+color << endl;
```

Типы данных языка C++

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

Еще один пример, объявление логического (булевского) типа:

```
enum boolean {false, true};
```

Типы данных языка C++

Структуры (struct)

Структурные объекты или просто структуры в C++ унаследованы из языка C. Отличия есть, и в первую очередь, возможностью определения функций (операций) в теле структуры.

Все компоненты структуры по умолчанию считаются открытыми.

Типы данных языка C++

Общий формат объявления структуры:

```
struct [name, tag]
{
   тип_1 поле_1;
   тип_2 поле_2;
   ……….
//
   прототипы функций (операций);
}[список_описателей];
```

Типы данных языка C++

Элементы структуры, именуемые далее полями могут иметь любой известный компилятору тип, кроме определяемого типа, но могут быть указателем или ссылкой на данный тип. Инициализация полей структуры в момент их определения стандартом языка не разрешена.

Типы данных языка C++

Пример структуры:

```
struct Student
{
    char Name[20];
    int Age;
    double Mark;
void Show()
{
    cout << " Name: " << Name << endl;
    cout << " Age: " << Age << endl;
    cout << " Mark: " << Mark << endl;
}
};
```

Типы данных языка C++

В данном примере объявлена единственная функция, позволяющая просматривать значение полей экземпляра данного типа.

Объявление объекта (экземпляра) типа структура по следующему формату:

имя_типа имя_объекта [= инициализация];

Типы данных языка C++

Например,

```
Student st_1;
Student st_2 = {"Ivan", 20, 3.4};
```

Доступ к компонентам структуры осуществляется с помощью операции доступа ‘.’ , ‘->’,например, st_2.Show();.

Типы данных языка C++

В C++ помимо структур имеется возможность определять свои типы используя классы. Отличие классов от структур в первую очередь в том, что все компоненты класса по умолчанию являются закрытыми (ключевое слово private).

Типы данных языка C++

Еще одна возможность создание своего собственного типа – объединения (смеси). Их формат похож на формат объявления структур, с той лишь разницей, что вместо слова struct используется union.

На внутреннем представлении отличия более значительны, в частности, в объединениях под все поля выделяется одно единственное поле (место) в памяти.

Типы данных языка C++

Его размер равен размеру поля, занимающему максимальный объем в памяти. Пример:

```
union Student
{
    char Name[20];
    int Age;
    double Mark;
void Show()
{
    cout << " Name: " << Name << endl;
    cout << " Age: " << Age << endl;
    cout << " Mark: " << Mark << endl;
}
};
```

Типы данных языка C++

Для представления объекта этого типа потребуется 160 байт памяти (20 объектов типа char).

Типы данных языка C++

Типы данных языка C++

Типы данных языка C++

Типы данных языка C++

Типы данных языка C++