ЛЕКЦИЯ 10

ОСНОВЫ ПРОГРАММИРОВАНИЯ

ПЕРЕГРУЗКА ФУНКЦИЙ

- **Задача:**
- Реализовать набор функций, реализующих один и тот же алгоритм для различных типов данных.

- В C++ допускаются ***перегруженные*** имена функций, когда функции с одним именем можно идентифицировать по их списку параметров (т.е. контексту, в котором имя употребляется).

- Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется ***перегрузкой*** (англ. resolution) функций.

ПЕРЕГРУЗКА ФУНКЦИЙ

- int Square(int arg)
  {return arg*arg;}
- double Square(double arg)
  {return arg*arg;}
- char *Square(const char *arg, int n)
  {static char res[256];
  int j = 0;
  while (*arg && j < n)
  { if (*arg != ' ') res[j++] = *arg;
  res[j++] = *arg++;}
  res[j] = 0;
  return res;}

ПЕРЕГРУЗКА ФУНКЦИЙ

- При вызове перегруженной функции компилятор определяет, какую именно функцию требуется вызвать, по типу фактических параметров. Этот процесс называется ***разрешением перегрузки***.

- Тип возвращаемого функцией значения в разрешении не участвует.

- Механизм разрешения сводится к тому, чтобы использовать функцию с наиболее подходящими аргументами и выдать сообщение, если такой не найдется.

ПЕРЕГРУЗКА ФУНКЦИЙ

- // наибольшее из двух целых:
- int max(int. int);
- // подстрока наибольшей длины:
- char* max(char*. char*);
- // наибольшее, из первого параметра и длины второго:
- int max (int. char*);
- // наибольшее из второго параметра и длины первого:
- int max (char*, int);

- void f(int a. int b. char* c. char* d){
- cout « max (a. b) « max(c, d) « max(a. c) « max(c, b);}

ПЕРЕГРУЗКА ФУНКЦИЙ

- Если точного соответствия не найдено, выполняются продвижения порядковых типов в соответствии с общими правилами преобразования типов: bool → int, char → int, float → double и т. п.
- Далее выполняются стандартные преобразования типов, например, int → double или указателей в void*.
- Далее выполняются преобразования типа, заданные пользователем, а также поиск соответствий за счет переменного числа аргументов функций.
- Если соответствие на одном и том же этапе может быть получено более чем одним способом, вызов считается неоднозначным и выдается сообщение об ошибке.

ПЕРЕГРУЗКА ФУНКЦИЙ

- Неоднозначность может появиться при:

  1. Преобразовании типа;
  2. Использовании параметров-ссылок;
  3. Использовании аргументов по умолчанию.

ПЕРЕГРУЗКА ФУНКЦИЙ

- **Правила описания перегруженных функций:**

- Функции не могут быть перегружены, если описание их параметров отличается только модификаторами const, volatile или использованием ссылки (например, int и const int или int и int&).

- Нельзя перегружать функции, отличающиеся только типом возвращаемого значения.

ПЕРЕГРУЗКА ФУНКЦИЙ

- Перегруженные функции должны находиться в одной области видимости, иначе произойдет сокрытие аналогично одинаковым именам переменных во вложенных блоках.

- Перегруженные функции могут иметь параметры по умолчанию, при этом значения одного и того же параметра в разных функциях должны совпадать. В различных вариантах перегруженных функций может быть различное количество параметров по умолчанию.

ПЕРЕГРУЗКА ФУНКЦИЙ

- ***Декорирование имен*** – формирование уникального внутреннего имени функции.

- void Func(void); // @Func$qv
- void Func(int); // @Func$qi
- void Func(int, int); // @Func$qii
- void Func(*char); // @Func$qpc
- void Func(unsigned); // @Func$qui
- void Func(const char*); // @Func$qpxc

ШАБЛОНЫ ФУНКЦИЙ

- Многие алгоритмы не зависят от типов данных, с которыми они работают (классический пример — сортировка).

- В C++ есть мощное средство параметризации — **шаблоны**.

- С помощью шаблона функции можно определить алгоритм, который будет применяться к данным различных типов, а конкретный тип данных передается функции в виде параметра на этапе компиляции.

ШАБЛОНЫ ФУНКЦИЙ

- Компилятор автоматически генерирует правильный код, соответствующий переданному типу.

- Таким образом, создается функция, которая автоматически перегружает сама себя и при этом не содержит накладных расходов, связанных с параметризацией.

- template < typename T >
- void sort(T array[], int size); { /* тело функции */ }

ШАБЛОНЫ ФУНКЦИЙ

- Первый же вызов функции, который использует конкретный тип данных, приводит к созданию компилятором кода для соответствующей версии функции. Этот процесс называется ***инстанцированием шаблона*** (instantiation).
- Конкретный тип для инстанцирования либо определяется компилятором автоматически, исходя из типов параметров при вызове функции, либо задается явным образом.
- При повторном вызове с тем же типом данных код заново не генерируется.

ШАБЛОНЫ ФУНКЦИЙ

```
// шаблон функции определения минимума
template < typename T >
T min( T a, T b )
{ return a < b ? a : b; }

// вызов функции по имени:
min( 1, 2 );
min( 'a', 'b' );
min( string( "abc" ), string( "cde" ) );
```

ШАБЛОНЫ ФУНКЦИЙ

- int i[5] = { 5, 4, 3, 2, 1 };
- sort<int>(i, 5);
-
- char c[] = "бвгда";
- sort<char>(c, strlen(c));
-
- sort<int>(c, 5); **// ошибка!**

ШАБЛОНЫ ФУНКЦИЙ

```
template< int BufferSize >
char* read()
{  char *Buffer = new char[ BufferSize ];
  /* считывание данных */
  return Buffer; }

char *ReadString = read< 20 >();
delete [] ReadString;
ReadString = read< 30 >();
```

ШАБЛОНЫ ФУНКЦИЙ

- // выведение значений параметров

- int i[5] = { 5, 4, 3, 2, 1 };
- sort(i, i + 5); // вызывается sort< int >
-
- char c[] = "бвгда";
- sort(c, c + strlen(c)); // вызывается sort< char >

ФУНКЦИЯ MAIN

- Функция, которой передается управление после запуска программы, должна иметь имя main.
- Она может возвращать значение в вызвавшую систему и принимать параметры из внешнего окружения.
- Возвращаемое значение должно быть целого типа.

ФУНКЦИЯ MAIN

- // вариант 1, без параметров:

- int main()
- { /* ... */}

- // вариант 2, с двумя параметрами:

- int main(int argc, char* argv[])
- { /* ... */}

ФУНКЦИЯ MAIN

- Имена параметров в программе могут быть любыми, но принято использовать **argc** и **argv**.
- **argc** определяет количество параметров, передаваемых функции, включая имя самой программы.
- **argv** является указателем на массив указателей типа char*.

ФУНКЦИЯ MAIN

- Каждый элемент массива содержит указатель на отдельный параметр командной строки, хранящийся в виде С-строки, оканчивающейся нуль-символом.

- Первый элемент массива argv[0] ссылается на полное имя запускаемого на выполнение файла, следующий argv[l] указывает на первый параметр, argv[2] – на второй параметр, и так далее.

- Параметр argv[argc] должен быть равен 0.

ФУНКЦИЯ MAIN

- #include <iostream>
- int main(int argc, char* argv[])
- {
- for (int i = 0: i<argc; i++) cout << argv[i] << '\n';
- }

АНАЛИЗ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

```
float number()
{
  int res = 0;
  for (;;)
  {
    char c = cin.get();    //получение символа из потока
    if (c >= '0' && c <= '9')
      res = res * 10 + c - '0';
    else
    {
      cin.putback(c);  //возврат символа в поток
      return res;
} } }
```

АНАЛИЗ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

```
float skobki()       //обработка чисел и скобок
{
  char c = cin.get();
  if (c == '(')
  {
    float x = expr();
    cin.get();
    return x;  }
  else
  {
    cin.putback(c);
    return number();
} }
```

АНАЛИЗ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

```
• float factor()       //умножение и деление
  {  float x = skobki();
    for (;;)
    { char c = cin.get();
      switch (c)
      {  case '*':
        x *= skobki();
        break;
      case '/':
        x /= skobki();
        break;
      default:
        cin.putback(c);  //неизвестный символ
        return x;    }  }  }
```

АНАЛИЗ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

```
float expr()          //сложение и вычитание
{  float x = factor();
  for (;;)
  {    char c = cin.get();
    switch (c)
    {    case '+':
      x += factor();
      break;
    case '-':
      x -= factor();
      break;
    default:
      cin.putback(c);
      return x;
  } } }
```

АНАЛИЗ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

- Простейший вызов:

- ```
  int main()
  {
    cout << "Введите выражение: ";
    float res = expr();
    cout << "Результат: " << res << endl;
  }
  ```
  ```

ЗАДАЧИ

- **1.** Реализовать простейший **калькулятор** на основе рассмотренного кода. Добавить в процесс разбора проверку на ввод пустых скобок в выражении.

- **2.** Выполнить упражнения из раздела **«Одномерные массивы»**, оформив каждый пункт задания в виде **шаблона функции**. Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается. Привести примеры программ, использующих эти шаблоны для типов int, float и double.
- При запуске программы с опцией – **src** вывести на экран исходный код программы.