Занятие 8

Дополнительные сведения о функциях. Ссылочные переменные. Передача функции аргументов по ссылке. Аргументы по умолчанию. Перегрузка функций. Шаблоны функций. Спецификации шаблонов функций.

Пример 1. Обмен значениями с помощью ссылок и указателей.
При использовании имен переменных в аргументах функций копируются значения переменных из вызывающей функции в новые переменные в вызываемой функции. При использованнии указателей копируются адреса, что дает возможность для доступа к переменной, адрес которой копируется. Ссылка на переменную – альтернативное имя этой переменной, то есть никакая новая переменная не создается. Ссылка на переменную в аргументе вызываемой функции дает прямой доступ к этой переменной. В приведенном ниже примере последовательно вызываются три функции (`swapr, swapv, swapv`) с различными аргументами. Проследите, как меняются значения переменных `wallet1, wallet2` после вызовов функций.

```
#include <iostream>
void swapr(int &a, int &b) ;     // a, b - псевдонимы для int
void swapp(int *p, int *q) ;     // p, q - адреса int
void swapv(int a, int b) ;       // a, b - новые переменные

using namespace std;

void output(int wallet1, int wallet2){ // функция для вывода значений
  cout << " walletl = $" << wallet1;
  cout << " wallet2 = $" << wallet2 << endl;
}

int main(void) {
  int wallet1 = 300; // инициализируем переменную
  int wallet2 = 350; // инициализируем переменную

  // Выводим начальные значения на экран
  output(wallet1, wallet2);

  // Использование ссылок для обмена содержимого
  cout << "\nUsing references to swap contents:\n";
  swapr(wallet1, wallet2);  // передача переменных
  output(wallet1, wallet2); // проверяем, что изменилось

  // Использование указателей для обмена содержимого
  cout << "\nUsing pointers to swap contents again:\n";
  swapp(&wallet1, &wallet2); // передача адресов переменных
  output(wallet1, wallet2);  // проверяем, что изменилось

  // Попытка использования передачи по значению
  cout << "\nTrying to use passing by value:\n";
  swapv(wallet1, wallet2);  // передача значений переменных
  output(wallet1, wallet2); // проверяем, что изменилось

  return 0;
}
```

```
/*********************************************************************/
void swapr(int &a, int &b){ // использование ссылок
  /*
   * Ссылки a и b дают доступ к переменным, указанным
   * при вызове фунцкии swapr.
   */
  int temp; // временная переменная
  temp = a; // сохранение значения a в temp.
  a = b;    // копирование из b в a
  b = temp; // копирование значения из temp в b
}
void swapp(int *p, int *q) { // использование указателей
  /*
   * Переменные p и q содержат адреса переменных типа int
   */
  int temp;
  temp = *p; // копирование значения из адреса p
  *p =*q;    // запись значения из адреса q в адрес p
  *q = temp;
}
void swapv(int a, int b){// попытка использования значений
  int temp;
  temp = a; // копирование из a в temp
  a = b;
  b = temp;
}
```

Пример 2. Перегрузка функции

Перегрузка функций – использование нескольких функций с одним и тем же именем, но разными списками аргументов. При вызове перегруженной функции компилятор определяет нужную функцию исходя из соответствия аргументов (их количества и типов); тип возвращаемого значения не играет определяющую роль. Пример ниже иллюстрирует вызов перегруженных функций.

```
#include <iostream>
using namespace std;
// Две функции и одним и тем же именем left:
unsigned long left (unsigned long num, unsigned ct);
char *left(const char *str, int n = 1); // последний аргумент =1, если
                                        // функция вызывается с одним
                                        // аргументом.
/*********************************************************************/
int main(void) {
  using namespace std;
  const char *trip = "Hawaii!";  // тестовое значение
  unsigned long n = 12345678;    // тестовое значение
  int i;
  char * temp;
  for (i = 1; i < 10; i++) {
    cout << left(n, i) << endl; // вызов left(unsigned long, unsigned)
    temp = left(trip,i);        // вызов left(const char *, int n)
    cout <<  temp << endl;
    delete [] temp;     // указатель на временную область хранения
  }
```

```
  return 0;
}
/*****************************************************************/
unsigned long left(unsigned long num, unsigned ct) {
  /*
   * Возвращает число, состоящее из первые ct цифр числа num, если
   * количество цифр в числе num больше ct, или число полностью в
   * противном случае.
   */
  unsigned long n = num;
  if (ct == 0 || num == 0)
    return 0;       // возврат 0 в случае отсутствия цифр

  unsigned digits = 1;
  while (n /= 10) // подсчет количества знаков в числе
    digits++;

  if (digits > ct) { // обрезает лишние цифры
    ct = digits - ct;
    while (ct--)
      num /= 10;
    return num;   // возврат ct знаков слева
  }
  else             // если ct >= количества цифр
    return num;   // возврат числа целиком
}
/*****************************************************************/
char *left(const char * str, int n) {
  /*
   * Возвращает указатель на новую строку, состоящую
   * из n первых символов строки str
   */
  if (n < 0)
    n = 0;
  char * p = new char[n+1];
  int i;
  for (i = 0; i < n && str[i]; i++)
    p[i] = str[i];   // копирование символов
  while (i <= n)
    p[i++] = '\0';   // установка остальных символов строки в '\0'
  return p;
}
```

Пример 3. Использование шаблона функции.
Шаблон функции позволяет создавать функции, использующие один и тот же алгоритм, но различные типы данных. Для этого вводится имя (`T` в примере ниже) для типа данных (обобщенный тип), который будет затем заменяться на конкретный тип. Сама функция создается компилятором при вызове шаблонной функции, причем тип данных определяется типом переменных, используемых для вызова. Пример ниже использует шаблон функции для обмена значениями двух переменных произвольных типов.

```
#include <iostream>

// прототим шаблона функции
```

```
template <typename T>  // или <class T>, T - это обобщенный тип
void Swap(T &a, T &b); // используем ссылки на T

int main(void) {
  using namespace std;

  int i = 10, j = 20; // две переменые
  cout << "i, j = " << i << ", " << j << ".\n";

  cout << "Using compiler-generated int swapper:\n";
  Swap(i,j); // генерирует void Swap(int &, int &),
             // то есть тип T будет заменен на int
  cout << "Now i, j = " << i <<", " << j << " .\n";

  double x = 24.5, y = 81.7; // две переменые
  cout << "x, y= " << x << ", " << y << ".\n";
  cout << "Using compiler-generated double swapper:\n";
  Swap(x,y); // генерирует void Swap(double &, double &),
             // то есть тип T будет заменен на double
  cout << "Now x, y="<< x <<", " << y <<"\n";

  return 0;
}

// определение шаблона функции
template <typename T>
void Swap(T &a, T &b) {
  T temp; // temp - переменная типа T
  temp = a;
  a = b;
  b = temp;
}
```

Пример 4. Специализация переопределяет шаблон.
Иногда возникает необходимость изменить работу алгоритма, описанного в шаблоне, при работе с определенным типом данных. Для этого можно переопределить шаблон, то есть создать явную специализацию для этого типа данных. Тогда при вызове шаблонной функции с этим типом данных будет использоваться специализация шаблона, а не общий шаблон. В примере показано, что при вызове шаблонной функции с аргументами типа `struct job` будет использоваться специализация шаблона.

```
#include <iostream>

// прототип шаблона функции
template <typename T>
void Swap(T &a, T &b) ;

struct job {
  char name[40];
  double salary;
  int floor;
};

// прототип явной специализации при работе с типом struct job
```

```
14 template <> void Swap<job>(job &j1, job &j2);
15
16 void Show(job &j);
17 /*************************************************************************/
18 int main(void) {
19   using namespace std;
20   cout.precision (2);
21   cout.setf(ios::fixed, ios::floatfield);
22
23   // использование шаблона Swap
24   int i = 10, j = 20;
25   cout << "i, j = " << i << ", " << j << ". \n";
26   cout << "Using compiler-generated int swapper. \n";
27   Swap(i,j); // генерирует void Swap (int &, int &)
28   cout << "Now i, j = " << i << ", " << j << ". \n";
29
30   // использование специализации Swap
31   job Sue = {"Susan Yaffee", 73000.60, 7};
32   job Sidney = {"Sidney Taffee", 78060.72, 9};
33   cout << "Before job swapping. \n";
34   Show(Sue);
35   Show(Sidney);
36   Swap (Sue, Sidney); // использует void Swap (job &, job &)
37   cout <<  "After job swapping. \n" ;
38   Show(Sue);
39   Show(Sidney);
40   return 0;
41 }
42 /*************************************************************************/
43 template <typename T> // шаблон функции
44 void Swap (T &a, T &b){
45   T temp;
46   temp = a;
47   a = b;
48   b = temp;
49 }
50 /*************************************************************************/
51 template <> void Swap<job>(job &j1, job &j2){// специализация шаблона
52   /*
53    * Обменивается только значениями элементов salary и
54    * floor структуры job, а не всеми элементами.
55    */
56
57   Swap(j1.salary, j2.salary); // uses void Swap(double &, double &)
58   Swap(j1.floor,  j2.floor);  // uses void Swap(int &, int &)
59
60   // можно использовать также
61   // double dd = j1.salary;  j1.salary = j2.salary;   j2.salary = dd;
62   // int    ii = j1.floor;   j1.floor  = j2.floor;    j2.floor  = ii;
63 }
64 /*************************************************************************/
65 void Show(job &j) { // выводит информацию на экран
66   using namespace std;
67   cout << j.name << " : $" << j.salary
68        << " on floor " << j.floor << endl;
69 }
```

Домашнее задание

Задание 8.1. Напишите функцию, которая принимает ссылку на объект string в качестве параметра и преобразует содержимое string в символы верхнего регистра. Используйте библиотечную функцию `toupper()`. Напишите программу, использующую функцию в цикле. Пример вывода может выглядеть следующим образом:

```
Enter a string (q to quit) : go away
GO AWAY
Next string (q to quit) : good grief!
GOOD GRIEF!
Next string (q to quit) : q
Bye.
```

Задание 8.2. Структура `CandyBar` содержит три члена. Первый член хранит название конфет. Второй – вес в граммах (который может иметь дробную часть), а третий – количество килокалорий (целое значение). Напишите программу, использующую функцию, которая принимает в качестве аргументов ссылку на `CandyBar`, указатель на `char`, значение `double` и значение `int`. Функция использует три последних значения для установки соответствующих членов структуры. Три последних аргумента должны иметь значения по умолчанию: "`Snickers`", 50.5 г и 259 ккал. Кроме того, программа должна использовать функцию, которая принимает в качестве аргумента ссылку на `CandyBar` и отображает содержимое этой структуры. Где необходимо, используйте `const`.

Задание 8.3. Напишите шаблонную функцию `max5()`, которая принимает в качестве аргумента массив из пяти элементов типа T и возвращает наибольший элемент в массиве. (Поскольку размер массива фиксирован, его можно жестко закодировать в цикле, а не передавать в виде аргумента.) Протестируйте функцию в программе с использованием массива из пяти значений `int` и массива из пяти значений `double`.

Задание 8.4. Напишите шаблонную функцию `maxn()`, которая принимает в качестве аргумента массив элементов типа T и целое число, представляющее количество элементов в массиве, а возвращает элемент с наибольшим значением. Протестируйте ее работу в программе, которая использует этот шаблон с массивом из шести значений `int` и массивом из четырех значений `double`. Программа также должна включать специализацию, которая использует массив указателей на `char` в качестве первого аргумента и количество указателей – в качестве второго, а затем возвращает адрес самой длинной строки. Если имеется более одной строки наибольшей длины, функция должна вернуть адрес первой из них. Протестируйте специализацию на массиве из пяти указателей на строки.