Занятие 7

Функции как программные модули C++.

Прототипы функций. Передача аргументов функциям по значению. Проектирование функций для обработки массивов. Проектирование функций для обработки текстовых строк. Указатели на функции.

Пример 1. *Функция с двумя аргументами.*
Функции в языке C/C++ играют роль подпрограмм. Функции позволяют объединить некую последовательность инструкций, которая выполняет определенную задачу. Пример ниже показывает использование функции `n_chars()`, которая распечатывает определенный символ несколько раз. Она принимает два аргумента (символ и целое число) и не возвращает ничего. Перед вызовом функции задается ее прототип, который определяет типы аргументов и возвращаемого значения. Задание прототипа позволяет проверить правильность вызова функции (соответстие типов аргументов и возвращаемого значения). В списке аргументов прототипа можно не писать имена переменных. Описание самой функции может находится в любом месте программы и даже в другом файле. При вызове функции `n_chars()` создаются две новые локальные переменные (`char c` и `int n`), определенные в списке ее аргументов. В эти переменные копируются значения, пересылаемые при вызове функции, то есть значения переменных `ch` и `times`. Если бы внутри `n_chars()` значения переменых `char c` и `int n` менялось, то это бы никак не повлияло на значения переменыых `ch` и `times` в функции `main()`.

```
#include <iostream>
using namespace std;

void n_chars(char, int); // декларация (прототип) функции

int main(void) {
  char ch;
  cout << "Enter a character: "; // запрос на ввод символа
  cin  >> ch;
  while (ch != 'q'){ // q завершает цикл
    int times;
    cout << "Enter an integer: "; // ввод целого числа
    cin  >> times;
    n_chars(ch, times);                  // вызов функции
    cout << "\nEnter another character " // ввод другого символа
      "or press the q-key to quit:";     // или q для завершения
    cin  >> ch;
  }
  return 0;
}

// определение функции
void n_chars(char c, int n) {// вывод значения символа c n раз
  while (n-- > 0) // продолжение, пока n не достигнет 0
    cout << c;
}
```

Пример 2. *Функция с аргументом-массивом.*
Если надо передать массив, то внутри вызываемой функции создается переменная-указатель, в значение которой записывается адрес массива при вызове этой функции. Сам массив не копируется. Используя указатель, вызванная функция получает доступ к элементам массива. Так как указатель содержит адрес начала массива, то обычно передается также и количество элементов в этом массиве в виде еще одного аргумента функции.

```
#include <iostream>
const int ArSize = 8;
int sum_arr(int arr[], int n); // прототип, можно int *arr  вместо int arr[]

int main(void) {
  using namespace std;
  int numbers[ArSize] = {1,2,4,8,16,32,64,128}; // задаем массив
  int sum = sum_arr(numbers, ArSize);  // получение суммы элементов
  cout << "Sum of all numbers: " << sum << "\n";  // вывод суммы
  return 0;
}
/*****************************************************************/
int sum_arr(int arr[], int n) {
  /*
   * Возвращает сумму элементов массива целых чисел.
   * Переменная arr является указателем на массив,
   * то есть ее значение - адрес массива.
   *
   * В списке аргументов указатель можно декларировать как
   * int arr[] или как int *arr
   */

  int total = 0;
  for (int i = 0; i < n; i++)
    total = total + arr[i];
  return total;
}
```

Пример 3. *Функция со строковым аргументом.*
Строки используются также как и массивы, то есть внутри вызываемой функции создается указатель, куда копируется адрес. В отличие от обычного массива использование строки не требует передачи длины, так как предполагается, что строка заканчивается нулевым символом.

```
#include <iostream>
unsigned int c_in_str(const char *str, char ch);

int main(void) {
  using namespace std;
  char mmm[15] = "minimum";           // строка в массиве
  const char *wail = "ululate";       // *wail указывает на строку
  unsigned int ms = c_in_str (mmm, 'm');
  unsigned int us = c_in_str(wail, 'u');
```

```
10
11    // вывод количества символов m и n
12    cout << ms << " m characters in " << mmm << endl;
13    cout << us << " u characters in " << wail << endl;
14  }
15
16  /*****************************************************************/
17  unsigned int c_in_str(const char *str, char ch) {
18    /*
19     * Подсчитывает количество символов ch в строке,
20     * адрес которой записан в str.
21     */
22
23    unsigned int count = 0;
24    while (*str) {// завершение, когда *str равно '\0'
25      if (*str == ch)
26        count++;
27      str++; // перемещение указателя на следующий символ
28    }
29    return count;
30  }
```

Пример 4. *Функция, возвращающая указатель на char.*
Функции могут возвращать адреса памяти. В примере показано выделение памяти для строки внутри функции `buildstr`. Сама функция затем возвращает адрес строки, чтобы ее можно было использовать в вызывающей функции.

```
1  #include <iostream>
2  char *buildstr(char c, int n);     // прототип
3  int main(void) {
4    using namespace std;
5    int times;
6    char ch;
7    cout <<  "Enter a character: ";
8    cin >> ch;                       // ввод символа
9    cout << "Enter an integer: ";
10   cin >> times;                    // ввод целого числа
11   char *ps = buildstr (ch, times);// также выделяет память
12   cout << "ps=" << ps << endl;
13   delete [] ps;                    // освобождение памяти
14   ps = buildstr('+', 20) ;         // повторное использование указателя
15   cout << ps << "-DONE-" << ps << endl;
16   delete [] ps;                    // освобождение памяти !
17   return 0;
18 }
19 /*****************************************************************/
20 char *buildstr(char c, int n) {
21   /*
22    * Выделяет память, записывает туда строку из n символов c,
23    * возвращает адрес строки.
24    */
25
26   char *pstr = new char[n + 1]; // выделение памяти
27   pstr[n] = '\0';  // ставит нулевой символ в конец строки
```

```
  while (n-- > 0)  // запись символа в строку
    pstr[n] = c;
  return pstr;     // возвращаем адрес строки
}
```

Пример 5. *Указатели на функции.*
Указатели могут содержать адреса памяти, где записаны функции. С такими указателями можно работать так же, как самими функциями. В примере ниже используется общий алгоритм (функция `deriv`) для нахождения численно производной любой функции. Затем находятся производные от двух функций, передавая их адреса в списке аргументов. Запустите программу и проверьте, правильно ли рассчитывается производная путем сравнения с аналитическим значением. Как меняется точность расчета при уменьшении параметра `d` в два раза?

```
#include <iostream>

double func1(double x){// определение функции func1
  return 4.0*x*x;
}

double func2(double x){ // определение функции func2
  return 1.0/x;
}

double deriv(double x, double d, double (*func)(double)); // прототип
/*****************************************************************/
int main(void) {
  using namespace std;

  double x = 2.0;     // точка, где находится производная
  double d = 1e-2;    // отступ от точки для расчета производной

  double der1 = deriv(x, d, func1); // находим производную от func1
  double der2 = deriv(x, d, func2); // находим производную от func2

  cout.precision(15); // увеличиваем количество знаков для вывода
  cout << "Derivative of func1 =" << der1 << endl;
  cout << "Derivative of func2 =" << der2 << endl;

  return 0;
}

/*****************************************************************/
double deriv(double x, double d, double (*func) (double)){
  /*
   * Вычисляет численно примерное значение производной от функции
   * func в точке x, используя значения функции в окрестности этой
   * точки. Переменная func является указателем на функцию,
   * которая принимает один аргумент типа double и возвращает
   * переменную типа double. Аргумент d определяет отступ от точки x,
   * используемый при численном расчете производной.
   */

```

```
40    double f1 = func(x-0.5*d); // значение функции слева,  при x-0.5*d
41    double f2 = func(x+0.5*d); // значение функции справа, при x+0.5*d
42
43    double deriv = (f2-f1)/d;  // производная как тангенс наклона
44
45    return deriv;
46 }
```

Задания

Задание 7.1. Напишите программу, которая переводит температуру по Цельсию в значение по Фарангейту. Используйте формулу $T_F = 32 + 9/5 * T_C$. Программа должна запрашивать значение температуры по Цельсию, вызывать функцию для перевода, а затем распечатывать получаемое значение. Диалог может выглядеть так:

```
Enter a Celsius value: 20
20 degrees Celsius = 68 degrees Fahrenheit
```

Задание 7.2. Напишите программу, которая многократно запрашивает у пользователя пару чисел до тех пор, пока хотя бы одно из этой пары не будет равно 0. С каждой парой программа должна использовать функцию для вычисления среднего гармонического (среднее гармоническое = 2xy/(x+y)) этих чисел. Функция должна возвращать ответ в `main()` для отображения результата.

Задание 7.3. Напишите программу, которая запрашивает у пользователя, сколько чисел он хочет ввести, а затем сохраняет вводимые значения в массиве. Программа должна рассчитать параметры набора введённых чисел: их среднее и дисперсию. Реализуйте ввод, отображение и расчёт параметров в отдельных функциях, работающих с массивами. Среднее значение N чисел:

$$M = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Дисперсия N чисел:

$$D = \frac{1}{N}\sum_{i=1}^{N} (x_i - M)^2.$$

Задание 7.4. Напишите программу, которая просит пользователя ввести строку и два символа. Программа должна затем вызывать функцию, которая заменяет в строке все символы, совпадающие с первым, на второй, и подсчитывает количество сделанных замен. После этого программа должна выводить на экран новую строку и количество замен.

Задание 7.5. Напишите функцию `sampling()`, которая в качестве аргументов принимает массив действительных чисел, их количество, а также указатель на другую функцию с одним действительным аргументом. Функция `sampling()` должна находить значения передаваемой функции при всех значениях из передаваемого массива, сохранять их в новом массиве и возвращать указатель на этот массив. Продемонстрируйте работу функции `sampling()` путем использования ее в программе.