Курс лекций "Программирование"
Основы программирования на языках C и C++

Лекция 7. Функциональная декомпозиция в языках C и C++

Глухих Михаил Игоревич, к.т.н., доц.

mailto: glukhikh@mail.ru

Что такое функция (процедура)?

- Законченный участок программы, решающий часть требуемой задачи
- У функции могут быть входные данные – аргументы
- У функции может быть результат
- **Сигнатура** = типы аргументов + имя + тип результата

Прототип (объявление) функции

```
int getNOD(int a, int b);
// В отличии от заголовка функции,
// в конце прототипа ставится
// точка с запятой
// Перед использованием функции
// в программе необходимо задать
// ее прототип ИЛИ определение
```

Определение функции

```
// Определение функции, вычисляющей НОД
// Название формируется по тем же правилам,
// что и имена переменных
// a и b - формальные параметры (аргументы) функции
int getNOD(int a, int b) // Заголовок
{  // Тело функции
   while (a!=b)
   {
      if (a<b)
         b=b-a;
      else
         a=a-b;
   }
   return a; // Результат функции
}
```

Пример использования – поиск НОД 3 чисел

- Воспользуемся уже написанной функцией getNOD, которая ищет НОД 2 чисел
- Известно, что
- НОД(A,B,C)=НОД(НОД(A,B),C);

Текст программы

```
int main(void)
{
   setlocale(LC_ALL, "Russian");
   cout<<"Поиск НОД 3 чисел"<<endl;
   int a=0,b=0,c=0;
   do
   {
      cout<<"Введите натуральные A, B, C:";
      cin>>a>>b>>c;
   } while (a<=0 || b<=0 || c<=0);
   // Функция вызвана дважды (!)
   int result=getNOD(getNOD(a,b),c);
   cout<<"НОД("<<a<<","<<b<<","<<c<<")="<<result<<endl;
   return 0;
}
```

Разберем подробнее вызов (использование) функции…

```
int result=getNOD(getNOD(a,b),c);
// Здесь функция вызвана дважды
// В первом случае (внутреннем)
// фактические параметры функции -
// переменные a и b, определенные
// в функции main
// Во втором случае (внешнем)
// фактические параметры функции -
// результат getNOD(a,b) и переменная c,
// определенная в функции main
```

Связь формальных и фактических параметров

- Независимо от того, какие переменные (константы) являлись **фактическими** параметрами, **формальными** параметрами **getNOD** остаются переменные **a** и **b**, определенные в заголовке **getNOD**
- Перед вызовом функции происходит **копирование** фактических параметров, их значения переписываются в **формальные** параметры. При таком способе вызова говорят, что используются **формальные параметры-значения**. Изменение формальных параметров при этом не повлияет на значение фактических параметров
- После определения значения формальных параметров выполняются операторы функции, до тех пор, пока не выполнится **return**

Хранение и использование формальных параметров

- Формальные параметры хранятся **в стеке** (как переменные, определенные внутри функции). Формальные параметры записываются в стек при вызове функции и вынимаются из стека по окончании работы функции (**return**). Это повторяется столько раз, сколько раз вызывается функция
- Формальный параметр может быть использован **только внутри тела функции**.

Разграничение доступа

- Каждая функция в своих операторах может использовать переменные, определенные внутри нее (**локальные**), и свои формальные параметры
- Также каждая функция может использовать **глобальные** переменные, определенные вне тела всех функций
- **НО!** Функция **не может** использовать переменные, определенные внутри других функций.

Формальный параметр-массив

- Сам массив копировать слишком долго. Поэтому в данном случае формальным аргументом делается указатель на 0-й элемент массива
- Имея адрес 0-го элемента массива, мы **можем** изменять содержимое массива. В такой ситуации говорят, что используется **формальный параметр-адрес**.
- Если мы хотим подчеркнуть, что содержимое массива изменяться не будет, используется константный указатель
- Следует помнить, что нам не будет известна фактическая длина передаваемого массива. Как правило, для ее передачи используется еще один параметр

Поиск максимального элемента в массиве

```
#include <assert.h>
#include <iostream>
#include <fstream>
using namespace std;
// Результат - максимальное число в массиве
int findMax(const int* arr, int length)
{
   // Прекратит выполнение программы, если условие неверно
   assert(length>0);
   int max=arr[0];
   for (int i=1; i<length; i++)
   {
      if (arr[i]>max)
         max=arr[i];
   }
   return max;
}
```

Пример – чтение массива из файла

```
// Результат - число прочитанных элементов, <=0 - ошибка
// Другие варианты заголовка?
int readArray(const char* fileName, int* arr, int maxLength)
{
   ifstream in(fileName);
   if (!in.is_open())
      return 0;
   int elem; // Читаемый элемент
   for (int i=0; i<maxLength; i++) {
      in>>elem; // Чтение очередного элемента
      if (in.fail()) // Не удалось прочитать
         return i;
      arr[i]=elem;
   }
   return maxLength;
}
```

Что делать с многомерными массивами?

- Если массивы ступенчатые, то можно также воспользоваться указателем
- Если обычные, используется следующий способ передачи:
- **int** myFunc(**int** arr[][10]);
- Так можно и с одномерным массивом:
- **int** myFunc(**int** arr[]);
- Фактически, все равно используется указатель

Можно ли сделать массив результатом функции?

```
// Да, можно. Но осторожно…
// Функция создает массив из первых
// чисел Фибоначчи
const int FIB_NUMBER=10;
int* createFib(void)
{
   int arr[FIB_NUMBER]={1, 1};
   for (int i=2; i<FIB_NUMBER; i++)
      arr[i]=arr[i-1]+arr[i-2];
   return arr;
}
```

Можно ли сделать массив результатом функции?

```
// Да, можно. Но осторожно…
// Функция создает массив из первых
// чисел Фибоначчи
const int FIB_NUMBER=10;
int* createFib(void)
{
   int arr[FIB_NUMBER]={1, 1};
   for (int i=2; i<FIB_NUMBER; i++)
      arr[i]=arr[i-1]+arr[i-2];
   // Так нельзя!!!
   // Массив arr умрет вместе с функцией
   return arr;
}
```

А вот так правильно…

```
// Функция создает массив из первых
// чисел Фибоначчи
const int FIB_NUMBER=10;
int* createFib(void)
{
   int* arr=new int[FIB_NUMBER];
   arr[0]=arr[1]=1;
   for (int i=2; i<FIB_NUMBER; i++)
      arr[i]=arr[i-1]+arr[i-2];
   // Только тот, кто будет этим массивом
   // пользоваться, должен в конце освободить память
   return arr;
}
```

Пример использования

```
int main(void)
{
    setlocale(LC_ALL, "Russian");
    int* fibArr=createFib();
    cout<<"Числа Фибоначчи: ";
    for (int i=0; i<FIB_NUMBER; i++)
        cout<<fibArr[i]<<" ";
    cout<<endl;
    delete[] fibArr;
    return 0;
}
```

В каких случаях используются функции?

- В тех случаях, когда одну и ту же последовательность действий приходится выполнять в разных местах программы (иногда проблему можно решить с помощью циклов, но далеко не всегда)
- Пример – расписание автобусов (лекция 3)

В каких еще случаях используются функции?

- В тех случаях, когда задача слишком сложна для того, чтобы получить компактное решение.
- Традиционно считается, что функции размером более 1-2 экранов (что-то около 50 строк) являются слишком громоздкими для понимания и анализа
- В подобных ситуациях в задаче выделяются **законченные подзадачи** и реализуются в виде функций. Если и они являются слишком сложными, в них также выделяются подзадачи.
- Подобный процесс называется **функциональной декомпозицией**

Пример

- Центрировать текст, размещенный во входном файле; результат вывести в выходной файл. Текст выравнен влево, причем длина одной строки в нем не превышает 80 символов.
- Подзадачи?

Какие подзадачи можно выделить?

- Чтение строки из файла
  - необходимо воспользоваться функцией getline
  - после этого необходимо проверить, правильно ли строка прочиталась
- Центрирование строки
  - необходимо добавить в начало строки нужное количество пробелов
- Вывод результата в файл

Чтение строки из файла

- Аргумент 1 – файл (имя или поток?)
  - если мы передаем имя, поток придется каждый раз создавать заново
  - поэтому передавать нужно поток; поток – сложный тип, такие аргументы обычно передаются по ссылке: ifstream& in
  - ссылка – вид указателя, который не нужно разадресовывать
- Аргумент 2 – указатель на буфер, куда будем записывать строку (а почему не стоит делать его результатом?)
- Результат – строка прочитана или нет

```
bool readLine(ifstream& in, char* buffer);
```

Центрирование строки

- Аргумент – указатель на строку, которую центрируем
- Результат – отсутствует

```
void centerLine(char* buffer);
```

Текст программы

```
const int STR_LENGTH = 80; // Константа – макс. длина строки
// Функция чтения очередной строки из файла
// in – ссылка на поток, соответствующий файлу
// buffer – массив, куда следует записать результат
// результат – истина, если строка прочитана успешно
bool readLine(ifstream& in, char* buffer)
{
   buffer[0] = 0; // Для последующей проверки
   in.getline(buffer, STR_LENGTH+1, '\n');
   if (in.fail())
   {
      if (buffer[0]>0)
         cout<<"Слишком длинная строка во входном файле"<<endl;
      return false;
   }
   return true;
}
```

Текст программы

```
// Функция центрирования строки
// buffer - массив со строкой, которую центрируем
void centerLine(char* buffer)
{
   int len=strlen(buffer); // Длина строки
   // На сколько сдвигаем
   int shift = (STR_LENGTH-len)/2;
   if (shift==0)
      return;
   // Сдвинуть len символов + нуль-символ
   // Начиная с последнего! (почему?)
   for (int i=len; i>=0; i--)
      buffer[i+shift]=buffer[i];
   for (int i=0; i<shift; i++)
      buffer[i]=' ';
}
```

Текст программы

```
#include <iostream>
#include <fstream>
#include <locale.h>
#include <string.h>
using namespace std;
bool readLine(ifstream& in, char* buffer) { ... }
void centerLine(char* buffer) { ... }
int main(void)
{
   setlocale(LC_ALL,"Russian");
   ifstream in("in.txt");
   if (!in.is_open())
   {
      cout<<"Файл in.txt не существует"<<endl;
      return -1;
   }
```

Текст программы

```
    ofstream out("out.txt");
    if (!out.is_open())
    {
        cout<<"Невозможно создать выходной файл out.txt"<<endl;
        return -2;
    }
    char buffer[STR_LENGTH+1];
    while (readLine(in, buffer))
    {
        centerLine(buffer);
        out<<buffer<<endl;
    }
    return 0;
}
```

Чем это лучше, чем программа из одной функции main?

- Главным образом тем, что выглядит не столь громоздко – все функции довольно короткие и легко обозримые, нет большого количества вложенных друг в друга конструкций, программа легко читается

В каком порядке писать функции?

- Существуют два разных подхода
- **Программирование сверху** (дедуктивный подход, от сложного к простому): в первую очередь пишем главную функцию; для решения более простых подзадач вызываем в ней другие функции (еще не написанные); затем переходим к написанию более простых функций (если нужно, выделяем еще более простые подзадачи, заменяя их вызовами функций) и так далее.

В каком порядке писать функции?

- Существуют два разных подхода
- **Программирование снизу** (индуктивный подход, от простого к сложному): в первую очередь пишем самые простые функции; затем, с их помощью, создаем чуть более сложные; затем еще более сложные; и в итоге создаем главную функцию.

Какой подход лучше?

- Авторитеты программирования склоняются к дедуктивному подходу (сверху вниз). На практике все не так однозначно.
  - при использовании программирования снизу есть риск, что некоторые функции не понадобятся или понадобятся в несколько другой форме
  - с другой стороны, при использовании программирования сверху есть риск, что выделенные задачи на нижних уровнях окажутся неразрешимыми (например, у функций не хватит необходимой информации)

Возможный порядок проектирования

- Вначале попробуйте определить, какие более простые функции вам понадобятся для решения основной задачи. Напишите их прототипы.
- Затем попробуйте, пользуясь этими прототипами, написать главную функцию программы. При необходимости модифицируйте написанные прототипы или добавляйте новые.
- И уже после этого переходите к написанию определений более простых функций. Если они достаточно сложны, ту же процедуру можно повторить и для них.

Пример проектирования

- Во входном файле перечислены координаты 8 ферзей на шахматной доске. Необходимо определить пары ферзей, угрожающие друг другу, и вывести их координаты в выходной файл
- Координаты ферзей задаются в формате d6, где d задает вертикаль доски (a – 1-я, h – 8-я), 6 – горизонталь доски (от 1-й до 8-й)
- Ферзи угрожают друг другу, когда находятся на одной диагонали, горизонтали или вертикали

Какие подзадачи можно выделить?

- Чтение координат ферзей из файла
- Определение, угрожает ли пара ферзей друг другу
- Вывод пары ферзей в выходной файл

Какие функции им соответствуют?

```
// Чтение ферзей из файла
bool readQueens(const char* fileName,
    int* qx, int* qy);
// Угрожают ли ферзи друг другу
bool isThreaten(int qx1, int qy1,
    int qx2, int qy2);
// Вывод пары ферзей в выходной файл
void writePair(ofstream& out,
    int qx1, int qy1, int qx2, int qy2);
```

Главная функция

```
#include <iostream>
#include <fstream>
#include <locale.h>
using namespace std;
// … прототипы функций …
int main(void)
{
   setlocale(LC_ALL, "Russian");
   int qx[8], qy[8];
   if (!readQueens("in.txt", qx, qy))
   {
      cout<<"Не удалось прочитать координаты"<<endl;
      return -1;
   }
```

Главная функция

```
    ofstream out("out.txt");
    if (!out.is_open())
    {
        cout<<"Не удалось создать выходной файл"<<endl;
        return -2;
    }
    for (int i=0; i<8; i++)
    {
        for (int j=i+1; j<8; j++)
        {
            if (isThreaten(qx[i], qy[i], qx[j], qy[j]))
                writePair(out, qx[i], qy[i], qx[j], qy[j]);
        }
    }
    return 0;
}
```

Как прочитать из файла координаты ферзей?

- Для этого нужно уметь прочитать вертикаль (координату x) и горизонталь (координату y). Поскольку смешиваются буквы и цифры, лучше читать символы и затем преобразовывать их в нужный нам формат.

```
int readVert(ifstream& in);
int readHoris(ifstream& in);
```

Функция readQueens

```
bool readQueens(const char* fileName, int* qx, int* qy)
{
   ifstream in(fileName);
   if (!in.is_open())
      return false;
   for (int i=0; i<8; i++)
   {
      qx[i]=readVert(in);
      qy[i]=readHoris(in);
      if (in.fail() || qx[i]==0  || qy[i]==0)
         return false;
   }
   return true;
}
```

Функции readVert, readHoris

```
int readVert(ifstream& in)
{
   char ch;
   in>>ch;
   if (ch<'a' || ch>'h')
      return 0;
   return ch-'a'+1;
}
int readHoris(ifstream& in)
{
   char ch;
   in>>ch;
   if (ch<'1' || ch>'8')
      return 0;
   return ch-'1'+1;
}
```

Как определить, угрожают ли ферзи друг другу?

- Либо равны x-координаты (одна вертикаль)
- Либо равны y-координаты (одна горизонталь)
- Либо разница между x-координатами совпадает с разницей между y-координатами (одна диагональ)

Функция isThreaten

```
bool isThreaten(int qx1, int qy1, int qx2, int qy2)
{
   return (qx1==qx2) || (qy1==qy2) ||
             (qx1-qx2==qy1-qy2) ||
             (qx2-qx1==qy1-qy2);
}
```

Наконец, функция вывода пары в выходной файл

```
void writePair(ofstream& out,
    int qx1, int qy1, int qx2, int qy2)
{
    // Здесь можно было бы написать
    // еще одну функцию…
    char qv1=(qx1-1)+'a';
    char qv2=(qx2-1)+'a';
    out<<qv1<<qy1<<'-'<<qv2<<qy2<<endl;
}
```

Демонстрация работы

- В среде

Итоги

- Рассмотрено
  - Функции, фактические и формальные параметры
  - Функциональная декомпозиция, преимущества
  - Программирование сверху и снизу
- Далее
  - Модульное программирование