Теория и технология программирования

Основы программирования на языках C и C++

Лекция 11. Области действия и времена жизни

Глухих Михаил Игоревич, к.т.н., доц.
mailto: glukhikh@mail.ru

Основные понятия

- Область действия (определенной переменной или функции) – участок программы, на котором возможно обращение (чтение, запись) к переменной или вызов функции
- Время жизни (определенной переменной) - участок программы, при исполнении которого данная переменная существует в памяти ЭВМ
- Область действия не может распространяться на те участки программы, где переменная не существует в памяти
- С другой стороны, при исполнении некоторых участков программы, где переменная существует в памяти, обращение к ней может быть невозможно

Классификация переменных с точки зрения области действия и времени жизни

- Глобальная (внешняя)
- Статическая (внешняя статическая)
- Локальная (внутренняя, автоматическая)
- Внутренняя статическая (локальная статическая)

Глобальные (внешние) переменные

- Определяются на верхнем уровне, вне текста функций:

```
int globalVar;
int main(void) { … }
```

- Размещаются в статической памяти, по умолчанию инициализируются нулями
- Область действия - файл, в котором переменная определена; область действия может быть распространена на другие файлы
- Время жизни – вся программа

Две глобальных переменных в разных файлах

```
// file1.cpp
int globalVar;
```

```
// file2.cpp
int globalVar;
```

- Определение двух глобальных переменных с одинаковыми именами в разных файлах программы ЗАПРЕЩЕНО
- Вызовет ошибку на этапе связывания (already defined symbol)
- Проблема – если файлы программы пишут разные люди, то определенные в них глобальные переменные могут пересечься

Распространение области действия глобальной переменной на другие файлы

```
// file1.cpp
int globalVar;
```

```
// file2.cpp
extern int globalVar;
```

- Область действия глобальной переменной globalVar будет распространена на file2.cpp
- Как правило, объявления переменных (extern) записываются в заголовочных файлах

Статические (внешние статические) переменные

- Определяются на верхнем уровне, с модификатором **static**:

```
static int staticVar;
int main(void) { … }
```

- Размещаются в статической памяти, по умолчанию инициализируются нулями
- Область действия - файл, в котором переменная определена; область действия НЕ МОЖЕТ быть распространена на другие файлы
- Время жизни – вся программа

Две статических переменных в разных файлах

```
// file1.cpp
static int staticVar;
```

```
// file2.cpp
static int staticVar;
```

- Определение двух статических переменных с одинаковыми именами в разных файлах программы ВОЗМОЖНО
- Будут созданы ДВЕ разные переменные. Область действия каждой – свой файл. Переменные не пересекаются
- В этом отношении статические переменные лучше глобальных

Сочетание глобальной и статической переменной

- // file1.cpp
- **static int** var;

- // file2.cpp
- **int** var;

- Определение глобальной и статической переменных с одинаковыми именами в разных файлах программы ВОЗМОЖНО
- Будут созданы ДВЕ разные переменные. Область действия каждой – свой файл, область действия глобальной переменной может быть распространена на все файлы, кроме file1.cpp.

Распространение области действия статической переменной на другие файлы ЗАПРЕЩЕНО

```
// file1.cpp
static int staticVar;
```

```
// file2.cpp
extern int staticVar;
```

- Вызовет ошибку на этапе связывания (unresolved external symbol)

Статическими бывают и функции

```
// Например
static void myFunc(int a) { … }
```

- Область действия данной функции - файл, в котором она определена. Ее нельзя вызывать из других файлов.
- Функции, которые не нужно вызывать из других файлов (типа insertElement из файла search.cpp), предпочтительнее объявлять статическими
- В программе не может быть двух и более нестатических функций с одинаковыми типами аргументов и результата - вызовет ошибку на этапе связывания (already defined symbol)

Пример использования статических переменных

- Реализовать модуль для работы с целочисленной очередью
- Очередь хранит целые числа и имеет один вход и один выход. Число, вошедшее первым, выходит первым; число, вошедшее последним, выходит последним
- FIFO – first input, first output

Особенность решения

- Используем статические переменные модуля для сохранения информации об очереди
  - содержимое будем хранить в целочисленном массиве (content)
  - кроме этого, необходимо знать размер очереди на данный момент (size)
  - кроме этого, необходимо знать, где очередь начинается и где кончается
- Другой возможный способ – передавать всю перечисленную информацию как аргументы функций
  - (+) позволяет создать много очередей в одной программе
  - (-) резко увеличивает число аргументов функций

Требуемые функции

- Добавить элемент в очередь

```
bool putInteger(int number);
```

  - результат – удалось или нет

- Достать элемент из очереди

```
bool getInteger(int* pNumber);
```

  - результат – удалось или нет
  - pNumber – указатель на переменную, куда следует записать полученный элемент

Файл queue.h

```
#ifndef _QUEUE_H
#define _QUEUE_H

bool putInteger(int number);

bool getInteger(int* pNumber);

#endif
```

Файл queue.cpp

```
#include "queue.h"

const int MAX_SIZE=100;

// Содержимое очереди
static int content[MAX_SIZE];
// Размер очереди
static int size = 0;
// Индексы для чтения и записи чисел
static int readIndex = 0, writeIndex = 0;
```

- Обратите внимание, что ни один модуль, кроме модуля queue, не имеет доступа к данным очереди.

Файл queue.cpp (продолжение)

```
bool putInteger(int number)
{
    if (size==MAX_SIZE)
        return false;
    size++;
    content[writeIndex++]=number;
    if (writeIndex==MAX_SIZE)
        writeIndex=0;
    return true;
}
```

Файл queue.cpp (продолжение)

```
bool getInteger(int* pNumber)
{
    if (size==0)
        return false;
    size--;
    *pNumber = content[readIndex++];
    if (readIndex==MAX_SIZE)
        readIndex=0;
    return true;
}
```

Пример использования

```
#include <iostream>
#include "queue.h"
using namespace std;

int main(void)
{
   putInteger(7);
   putInteger(10);
   putInteger(8);
   int num;
   while (getInteger(&num))
      cout<<"Number got: "<<num<<endl;
   return 0;
}
```

Локальные (внутренние, автоматические) переменные

- Определяются внутри блока команд:

```
{
    int a;
}
```

- Или внутри заголовка инструкции **for**:

```
for (int i=0; i<10; i++) { … }
```

- Или внутри заголовка функции (формальные параметры):

```
void myFunc(int a) { … }
```

Локальные (внутренние, автоматические) переменные

- В любом случае, размещаются в стеке, по умолчанию не инициализируются
- Область действия – блок, или инструкция **for**, или функция, где переменная определена
- Время жизни – время выполнения блока, или цикла **for**, или функции, где переменная определена

Скрытие имен

```
int i;
int main(void) {
   cin>>i; // Ввод глобальной переменной
   int sum=0;
   // Оператор разрешения области действия ::
   for (int i=0; i<::i; i++) {
      sum += i;
      // Добавляем локальную переменную (счетчик цикла)
      if (sum % 3 > 0) {
         int i=sum % 3;
         sum -= i; // Вычитаем локальную переменную (sum % 3)
      }
   }
   cout<<"sum*i="<<sum*i<<endl; // Умножаем на глобальную i
   return 0;
}
```

Оператор разрешения области видимости ::

```
int i;
int main(void)
{
    …
    for (int i=0; i<::i; i++) { … }
    …
}
// ::i - обращение к глобальной или
// статической переменной с именем i
```

Имена переменных

- Короткие незначащие имена (i,j) обычно даются разного рода счетчикам (которые действуют только в течение цикла)
- Другим переменным лучше давать имена со значением
- Чем важнее переменная, чем больше ее область действия, тем длиннее должно быть имя

Преобразование программы

- Работая программистом, неизбежно сталкиваешься с программами или функциями, написанными другими людьми
- Не всегда стиль, в котором они написаны, вам понравится
  - часто там не будет достаточного количества комментариев
  - документация тоже будет очень примерная
  - переменные будут иметь названия типа k123
  - и даже спросить, что это значит, иногда будет не у кого
- Однако необходимо уметь разобраться в том, что написал другой человек
- И даже не только разобраться, но и переписать более понятным образом

Внутренние статические переменные

- Определяются внутри блока команд (или внутри заголовка **for**) с модификатором **static**

```
void f1(int a)
{
    static int sum=5;
}
```

- Размещаются в статической памяти, по умолчанию инициализируются нулем
- Область действия – блок команд (или инструкция **for**)
- Время жизни – ВСЯ ПРОГРАММА. Инициализация выполняется при запуске программы и НЕ ПОВТОРЯЕТСЯ при вызове f1
- Скрывают имена других переменных с большей областью действия

Внутренние статические переменные

- Как можно использовать?
- Например, можно узнать, сколько раз была вызвана та или иная функция:

```
void someFunc(void)
{
    static int callNum=0;
    callNum++;
    cout<<"Call #"<<callNum<<endl;
    … // Какие-то команды
}
```

- При первом вызове будет выведено Call #1, при втором Call #2 и так далее
- Как правило, удобнее применять внешние статические переменные – но у них шире видимость
- Пример с простыми числами и массивом

Что выведет на экран данная программа (при вводе 3)?

```
#include <iostream>
using namespace std;
void f1(int a) {
   static int sum = 0;
   sum += a;
   cout<<"a="<<a<<" sum="<<sum<<endl;
   for (static int j=0; j<a; j++)
      cout<<"j="<<j<<endl;
}
int main(void) {
   int a;
   cin>>a;
   for (int i=1; i<=a; i++)
      f1(i);
   return 0;
}
```

Решение

- Цикл **for** в функции main выполнится 3 раза, функция f1 вызовется 3 раза (с аргументами 1, 2, 3)
- Внутренняя статическая переменная sum последовательно примет значения 1 (при a=1), 3 (при a=2), 6 (при a=3)
- Внутренняя статическая переменная j:
  - при первом вызове будет равна 0, после него 1(a);
  - при втором вызове будет равна 1, после него 2;
  - при третьем вызове будет равна 2, после него 3;
  - цикл **for** в функции f1 будет все время выполняться один раз.

Итог

- a=1 sum=1
- j=0
- a=2 sum=3
- j=1
- a=3 sum=6
- j=2

Что выведет на экран данная программа?

```
#include <iostream>
using namespace std;
int main(void)
{
   char str[]="Hello world!";
   char* ptr=str;
   while (*++ptr!=' ');
   char* ptr2=str;
   for (int i=0; i<5; i++)
   {
      char temp=*++ptr;
      *ptr=*ptr2;
      *ptr2++=temp;
   }
   cout<<str<<endl;
}
```

Решение

- В цикле while мы увеличиваем значение ptr и затем проверяем, не указывает ли он на пробел. Если указывает, то останавливаемся.
- В цикле for мы увеличиваем значение ptr, читаем символ, на который ptr указывает (последовательно w, o, r, l, d), сохраняем его в temp, на его место записываем символ, на который указывает ptr2 (изначально H), затем на место, куда указывает ptr2, записываем temp и увеличиваем ptr2
- Таким образом, символы с индексами 0-4 поменяются местами с символами с индексами 6-10
- Результат: world, Hello!

Итоги

- Рассмотрены
  - Области действия и времена жизни
  - Внешние, внешние статические, локальные, внутренние статические переменные
  - Задачи «что делает программа»?
- Далее: перегрузка операций