УДК 004.43

ОСОБЕННОСТИ РАБОТЫ В ЯЗЫКАХ C# / C++ С УКАЗАТЕЛЯМИ И ССЫЛКАМИ В ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

Голубь Надежда

Национальный аэрокосмический университет имени Н.Е. Жуковского «ХАИ»

Аннотация

Данная статья на конкретном примере показывает разницу в использовании указателей и ссылок при реализации динамических данных – членов класса при обучении основам объектно-ориентированного программирования на двух популярных алгоритмических языках C# и C++, которые обеспечивают работу с управляемыми и неуправляемыми объектами.

Abstract

This article shows, on a concrete example, the difference in the use of pointers and references when implementing dynamic data-class members when learning the basics of object-oriented programming in the two popular algorithmic languages C# and C++ that provide work with managed and unmanaged objects.

Введение

В алгоритмическом языке C++ есть очень распространенное, эффективное и активно применяемое средство программирования – работа с динамическими переменными посредством использования указателей и ссылок. С помощью оператора new программист помещает такие объекты в «кучу» (heap) и должен сам заботиться о своевременном её освобождении с помощью оператора delete. Это порождает ряд проблем, особенно у начинающих программистов, а также и у более опытных - при использовании наследования и динамического полиморфизма.

Язык C# работает с динамическими объектами в так называемом защищенном (safe) режиме без явного использования указателей и ссылок, организуя с помощью одноименного оператора new размещение объектов в «управляемой куче» (managed heap) [1]. При этом автоматически, по специальным алгоритмам делается анализ и освобождение уже ненужных объектов из «управляемой кучи» - так называемая «сборка мусора».

Постановка задачи

Язык C# при необходимости тоже может работать с указателями и ссылками, но в так называемом незащищенном (unsafe) режиме, сводя к минимуму проблемы с распределением и освобождением памяти, но уже не в управляемой куче, а в стеке. Для этого используется ключевое слово stackalloc, которое является допустимым только для выделения памяти локальным переменным. После выхода из локального блока они автоматически прекращают своё существование, избавляя тем самым программиста от заботы по очистке оперативной памяти.

В качестве базового примера был взят простейший пример вычисления y = a+b на языке C++, причем все переменные динамические и являются членами класса TCalc. Этот пример затем был переписан на язык C#. Это позволило увидеть особенности в реализации, а также в идеях управляемого (язык C#) и неуправляемого кода (unsafe C# и C++), а также некоторые «подводные камни».

Изложение основного материала

Описание основного фрагмента класса для вычисления арифметического выражения (язык C++):

```
class TCalc
{
private:
  // описание динамических данных - членов класса:
  // входные
    int* A;       double* B;  // указатели
  // результат
    double* Y;
    void Calc()  // функция вычисления y=a+b
   {    *Y = *A +*B; }  // разыменование указателей
public:
   // Конструктор
   TCalc(int av, float bv)
   // выделение памяти в «куче» и инициализация входных данных - членов класса
  {
    A = new int; *A=av;
    B = new float; *B=bv;
    Y = new double;
   }
   // Деструктор - уничтожение данных-членов класса
   // ОБЯЗАТЕЛЬНОЕ освобождение «неуправляемой кучи»
   ~TCalc()
  {  delete A;   delete B;  delete Y; }
   // Вычисление и выдача результата
   int show ()
  {   Calc();
      cout << "\nРезультат y = " << *Y << " по адресу: " << Y << endl;
      return (_getch());
   }
};    // TCalc
```

Фрагмент главной программы:

```
int main()
 {  int a = 100; double b = 2.657;
    // Создание объекта
    TCalc Object (a, b);
    // Вычисления и вывод результата
    Object.show();
  }
```

Теперь посмотрим, как работу с указателями для данных – членов класса можно реализовать на языке C#. При этом надо обязательно при создании проекта разрешить использование небезопасного кода, а также всюду, где появляются указатели или ссылки в коде, должно быть ключевое слово unsafe.

Описание класса для вычисления арифметического выражения (язык C#):

```
unsafe class TCalc
 {
  // Исходные данные (поля класса) - УКАЗАТЕЛИ!
  private int* a ;
  private double* b ;
  public unsafe TCalc(int* x, double* y) // Конструктор
    {
```

```
        // Но память полям – членам класса в конструкторе НЕ выделяется!!!
        a = x; // присваивание АДРЕСОВ исходных данных
        b = y;
        // Разыменование адресов (поля a и b ПОЛУЧАЮТ значения - инициализация)
        Console.WriteLine("\na = {0}, Address = {1:X}" , *a, (int)a);
        Console.WriteLine("b= {0}, Address = {1:X}", *b, (int)b);
      }
  private unsafe double Add()   // функция вычисления y=a+b
     { return *a + *b; }
  public unsafe void Show()
     {
       // выделение памяти для ЛОКАЛЬНОЙ ПЕРЕМЕННОЙ (результат) в СТЕКЕ!!!
       double* ans = stackalloc double[1];
       *ans = Add();
       Console.WriteLine("a+b= {0}, Address = {1:X}", *ans, (int)ans);
      }
 }    // TCalc
```

Фрагмент главной программы:

```
static unsafe void Main() // режим UNSAFE
      {
         int a=100;
         double b=2.657;
         TCalc c = new TCalc(&a, &b); // передача АДРЕСОВ ЛОКАЛЬНЫХ переменных
         c.Show();
          …….
      }
```

В C# можно инициализировать указатель, используя и УПРАВЛЯЕМЫЕ элементы: массив, строку, буфер фиксированного размера. Для этого используется оператор fixed, который не позволяет сборщику мусора переносить перемещаемую переменную. Оператор fixed задает указатель на управляемую переменную и "закрепляет" эту переменную на всё время выполнения оператора. Этот оператор допускается только в небезопасном (unsafe) контексте. Без оператора fixed указатели на перемещаемые управляемые переменные были бы бесполезны, так как при сборке мусора переменные переносились бы непредсказуемым образом [2].

Выводы

- Язык C# может явно обрабатывать ссылки и указатели только в небезопасном (unsafe) режиме.
- При использовании C++ программист сам должен следить за освобождением «неуправляемой» кучи.
- При обучении профессиональных программистов объектно-ориентированному программированию очень важно понимание студентами разницы в обработке динамических данных – членов класса, чтобы избежать трудно понимаемых проблем, связанных с «уборкой мусора» или его накоплением.

Список использованных источников:

1. C#. Руководство по C#. Базовые сведения о времени жизни объектов [Электронный ресурс]. - Режим доступа: www. URL: https://professorweb.ru/my/csharp/charp_theory/level13/13_1.php - 2018.

2. Оператор fixed (Справочник по C#) [Электронный ресурс]. - Режим доступа: www. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/fixed-statement - 20.07.2015.