Лабораторная работа № 22-23

«Программная реализация операции с указателями и ссылками»

Цель работы:

Теория

Если вы программировали на C/C++, то, возможно, вы знакомы с таким понятием как указатели. Указатели позволяют получить доступ к определенной ячейке памяти и произвести определенные манипуляции со значением, хранящимся в этой ячейке.

Ссылки - это, по сути, абстракция, которая ничего не говорит о своем значении (хотя по факту, CLR в них тоже помещает адрес).

Ссылка в C# всегда ссылается на валидный объект или равна null, указатель же может указывать в случайную область памяти.

Ключевое слово ref указывает на значение, переданное по ссылке. Оно используется в четырех разных контекстах:

· В сигнатуре и вызове метода для передачи аргумента в метод по ссылке. Дополнительные сведения см. в разделе Передача аргумента по ссылке.

· В сигнатуре метода для возврата значения вызывающему объекту по ссылке. Дополнительные сведения см. в разделе Возвращаемые ссылочные значения.

· В теле элемента для указания на то, что возвращаемые ссылочные значения хранятся локально в виде ссылки, которая может быть изменена вызывающим объектом, или, в общем случае, что локальная переменная обращается к другому значению по ссылке. Дополнительные сведения см. в разделе Ссылочные локальные переменные.

· В объявлении struct, чтобы объявить ref struct или ref readonly struct. Для получения дополнительной информации см. раздел Семантика ссылок с типами значений.

Для использования параметра `ref` и при определении метода, и при вызове метода следует явно использовать ключевое слово `ref`, как показано в следующем примере.

```
void Method(ref int refArgument)
{
    refArgument = refArgument + 44;
}

int number = 1;
Method(ref number);
Console.WriteLine(number);
// Output: 45
```

Аргумент, передаваемый в параметр ref или in, нужно инициализировать перед передачей. В этом заключается отличие от параметров out, аргументы которых не требуют явной инициализации перед передачей.
Члены класса не могут иметь сигнатуры, отличающихся только ref, in или out. Если единственное различие между двумя членами типа состоит в том, что один из них имеет параметр ref, а второй — параметр out или in, возникает ошибка компилятора.

В языке C# указатели очень редко используются, однако в некоторых случаях можно прибегать к ним для оптимизации приложений. Код, применяющий указатели, еще называют небезопасным кодом. Однако это не значит, что он представляет какую-то опасность. Просто при работе с ним все действия по использованию памяти, в том числе по ее очистке, ложится целиком на нас, а не на среду CLR. И с точки зрения CLR такой код не безопасен, так как среда не может проверить данный код, поэтому повышается вероятность различного рода ошибок.

Ключевое слово unsafe
Блок кода или метод, в котором используются указатели, помечается ключевым словом unsafe:

```
static void Main(string[] args) { // блок кода, использующий указатели unsafe { } }
```

Метод, использующий указатели:

```
unsafe private static void PointerMethod() { }
```

Также с помощью unsafe можно объявлять структуры:

```
unsafe struct State
{
}
```

Операции * и &

Ключевой при работе с указателями является операция *, которую еще называют операцией разыменовывания. Операция разыменовывания позволяет получить или установить значение по адресу, на который указывает указатель. Для получения адреса переменной применяется операция &:

```
static void Main(string[] args) {
    unsafe
    {
        int* x; // определение указателя
        int y = 10; // определяем переменную
        x = &y; // указатель x теперь указывает на адрес переменной y
        Console.WriteLine(*x); // 10
```

```
        y = y + 20; Console.WriteLine(*x);//
        30 *x = 50; Console.WriteLine(y); // переменная y=50
    }
    Console.ReadLine();
}
```

При объявлении указателя указываем тип int* x; - в данном случае объявляется указатель на целое число. Но кроме типа int можно использовать и другие: sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double, decimal или bool. Также можно объявлять указатели на типы enum, структуры и другие указатели.

Выражение x = &y; позволяет нам получить адрес переменной y и установить на него указатель x. До этого указатель x не на что не указывал.

После этого все операции с y будут влиять на значение, получаемое через указатель x и наоборот, так как они указывают на одну и ту же область в памяти.

Для получения значения, которое хранится в области памяти, на которую указывает указатель x, используется выражение *x.

Получение адреса

Используя преобразование указателя к целочисленному типу, можно получить адрес памяти, на который указывает указатель:

```
int* x; // определение указателя
int y = 10; // определяем переменную
x = &y; // указатель x теперь указывает на адрес переменной y
// получим адрес переменной y
uint addr = (uint)x;
Console.WriteLine("Адрес переменной y: {0}", addr);
```

Так как значение адреса - это целое число, а на 32-разрядных системах диапазон адресов 0 до 4 000 000 000, то для получения адреса используется преобразование в тип uint, long или ulong. Соответственно на 64-разрядных системах диапазон доступных адресов гораздо больше, поэтому в данном случае лучше использовать ulong, чтобы избежать ошибки переполнения.

Операции с указателями

Кроме операции разыменовывания к указателям применимы еще и некоторые арифметические операции(+, ++, -, --, +=, -=) и преобразования. Например, мы можем преобразовать число в указатель:

```
int* x; // определение указателя
int y = 10; // определяем переменную
x = &y; // указатель x теперь указывает на адрес переменной y
// получим адрес переменной y
```

```
uint addr = (uint)x;
Console.WriteLine("Адрес переменной y: {0}", addr);
byte* bytePointer = (byte*)(addr+4); // получить указатель на следующий байт после а
Console.WriteLine("Значение byte по адресу {0}: {1}", addr+4, *bytePointer);
// обратная операция
uint oldAddr = (uint)bytePointer - 4; // вычитаем четыре байта, так как bytePointer
int* intPointer = (int*)oldAddr;
Console.WriteLine("Значение int по адресу {0}: {1}", oldAddr, *intPointer);
// преобразование в тип double
double* doublePointer = (double*)(addr + 4);
Console.WriteLine("Значение double по адресу {0}: {1}", addr + 4, *doublePointer);
```

Так как у нас x - указатель на объект int, который занимает 4 байта, то мы можем получить следующий за ним байт с помощью выражения byte *chp = (byte)*addr+4;. Теперь указатель bytePointer указывает на следующий байт. Равным образом мы можем создать и другой указатель double *doublePointer = (double)*addr + 4;, только этот указатель уже будет указывать на следующие 8 байт, так как тип double занимает 8 байт.

Чтобы обратно получить исходный адрес, вызываем выражение bytePointer - 4. Здесь bytePointer - это указатель, а не число, и операции вычитания и сложения будут происходить в соответствии с правилами арифметики указателей. Например:

```
char* charPointer = (char*)123400;
charPointer += 4;// 123408
Console.WriteLine("Адрес {0}", (uint)charPointer);
```

Хотя мы к указателю прибавляем число 4, но итоговый адрес увеличится на 8, так как размер объекта char - 2 байта, а 2*4=8. Подобным образом действует сложение с другими типа указателей:

```
double* doublePointer = (double*)123000;
doublePointer = doublePointer+3; // 123024
Console.WriteLine("Адрес {0}", (uint)doublePointer);
```

Аналогично работает вычитание: doublePointer -=2 установит в указателе doublePointer в качестве адреса число 123008

Указатель на другой указатель

Объявление и использование указателя на указатель:

```
static void Main(string[] args)
{
    unsafe
    {
```

```
        int* x; // определение указателя
        int y = 10; // определяем переменную
        x = &y; // указатель x теперь указывает на адрес переменной y
        int** z = &x; // указатель z теперь указывает на адрес, который указывает и

        **z = **z + 40; // изменение указателя z повлечет изменение переменной y
        Console.WriteLine(y); // переменная y=50
        Console.WriteLine(**z); // переменная **z=50
    }
    Console.ReadLine();
}
```

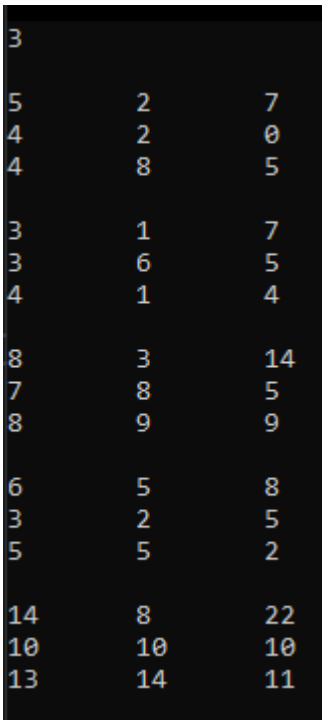
Ход работы:

8. Найти произведение k квадратных матриц A1, A2,..., Ak. Функция: вычисление произведения двух матриц.

```
int k = Convert.ToInt32(Console.ReadLine());
Console.WriteLine();
Random random = new Random();
int[,] matrix = new int[3, 3];

matrix = pullMatrix();

multiplyMatrix(ref matrix);

void multiplyMatrix(ref int[,] _matrix)
{
    int[,] arr = new int[3, 3];
    arr = pullMatrix();

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            _matrix[i, j] += arr[i, j];
            Console.Write($"{_matrix[i, j]}\t");
        }
        Console.WriteLine();
    }
    Console.WriteLine();
    k--;
    if (k <= 1)
        return;
    multiplyMatrix(ref _matrix);
} //Складывание матриц
```

```
int[,] pullMatrix()
{
    int[,] array = new int[3, 3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            array[i,j] = random.Next(0, 9);
            Console.Write($"{array[i, j]}\t");
        }
        Console.WriteLine();
    }
    Console.WriteLine();
    return array;
} //Заполнить матрицу и показать матрицу

Console.ReadKey();
```

выполнение:



Контрольные вопросы:

1. Как передать значение функции?

```
void foo(сюда)
```

2. Как получить значение от функции?

```
return
```

3. Для чего в языке C# используют указатели и ссылки?
   В языке C# **указатели** очень редко **используются**, однако в некоторых случаях можно прибегать к ним для оптимизации приложений.
4. Каким образом указатели передаются функции?

```
var foo(**z)
```