**Практическое занятие № 25**

**Использование указателей для организации связанных списков.**

**Цель:** получить навыки работы с указателями для организации связанных списков.

**Норма времени:** 2 часа.

**Оборудование:** Компьютер, среда программирования Visual Studio.

## Порядок выполнения работы

### Указатели

Указатели позволяют получить доступ к определенной ячейке памяти и произвести определенные манипуляции со значением, хранящимся в этой ячейке.

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.

Чтобы использовать небезопасный код в C#, надо первым делом указать проекту, что он будет работать с небезопасным кодом. Для этого надо установить в настройках проекта соответствующий флаг - в меню Project (Проект) найти Свойства проекта. Затем в меню Build установить флажок Allow unsafe code (Разрешить небезопасный код). Теперь мы можем приступать к работе с небезопасным кодом и указателями.

### 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.

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

**Указатели на типы и операция ->**

Кроме указателей на простые типы можно использовать указатели на структуры.

А для доступа к полям структуры, на которую указывает указатель, используется операция - >:

```
class Program
{    static void Main(string[] args)
    {        unsafe
        {
            Person person;
            person.age = 29;
            person.height = 176;
            Person* p = &person;
            p->age = 30;
            Console.WriteLine(p->age);
            // разыменовывание указателя
            (*p).height = 180;
            Console.WriteLine((*p).height);
        }
    }
}

public struct Person
{
    public int age;
    public int height;
}
```

Обращаясь к указателю **p->Age = 30;** мы можем получить или установить значение свойства структуры, на которую указывает указатель. Обратите внимание, что просто написать **p.Age=30** мы не можем, так как p - это не структура Person, а указатель на структуру.

Альтернативой служит операция разыменования: **(*p).height = 180;**

**Указатели на массивы и stackalloc**

С помощью ключевого слова stackalloc можно выделить память под массив в стеке. Смысл выделения памяти в стеке в повышении быстродействия кода. Посмотрим на примере вычисления факториала:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication4
{
    class Program
    {
        // Указатели на массивы и stackalloc
        static void Main(string[] args)
        {
            unsafe
            {
                const int size = 7;
                int* factorial = stackalloc int[size]; // выделяем память в стеке под
семь объектов int
                int* p = factorial;

                *(p++) = 1; // присваиваем первой ячейке значение 1 и
                // увеличиваем указатель на 1
                for (int i = 2; i <= size; i++, p++)
                {
                    // считаем факториал числа
                    *p = p[-1] * i;
```

```
            }
            for (int i = 1; i <= size; ++i)
            {
                Console.WriteLine(factorial[i - 1]);
            }
            Console.ReadKey();
        }

    }
}
}
```

Оператор stackalloc принимает после себя массив, на который будет указывать указатель. **int* factorial = stackalloc int[size];**.

Для манипуляций с массивом создаем указатель **p: int* p = factorial;**, который указывает на первый элемент массива, в котором всего 7 элементов

Далее начинаются уже сами операции с указателем и подсчет факториала. Так как факториал 1 равен 1, то присваиваем первому элементу, на который указывает указатель p, единицу с помощью операции разыменования: ***(p++)= 1**;

Из прошлой темы мы узнали, что чтобы вложить некоторое значение по адресу, который хранит указатель, надо использовать выражение: ***p=1**. Но кроме этого тут происходит также инкремент указателя p++. То есть сначала первому элементу массива присваивается единица, потом указатель p смещается и начинает указывать уже на второй элемент. Мы могли бы написать это так:

***p= 1;**
**p++;**

Чтобы получить предыдущий элемент и сместиться назад, можно использовать операцию декремента: **Console.WriteLine(*(--p));**. Обратите внимание, что операции ***(--p)** и ***(p--)** различаются, так как в первом случае сначала идет смещение указателя, а затем его разыменовывание. А во втором случае - наоборот.

Затем вычисляем факториал всех остальных шести чисел: ***p = p[-1] *i;**. Обращение к указателям как к массивам представляет альтернативу операции разыменовывания для получения значения. В данном случае мы получаем значение предыдущего элемента.

И в заключении, используя указатель factorial, выводим факториалы всех семи чисел.

**Оператор fixed и закрепление указателей**

Ранее мы посмотрели, как создавать указатели на типы значений, например, int или структуры. Однако кроме структур в C# есть еще и классы, которые в отличие от типов значений, помещают все связанные значения в куче. И в работу данных классов может в любой момент вмешаться сборщик мусора, периодически очищающий кучу. Чтобы фиксировать на все время работы указатели на объекты классов используется оператор fixed.

Допустим, у нас есть класс Person:

```
public class Person
{
 public int age;
 public int height;
}
```

Зафиксируем указатель с помощью оператора fixed:

```
unsafe
 {
       Person person = new Person();
       person.age = 28;
       person.height = 178;
       // блок фиксации указателя
       fixed(int* p = &person.age)
             {
```

```
                if (*p < 30)
                    {
                        *p = 30;
                    }
                }
            Console.WriteLine(person.age); // 30
    }
```

Оператор fixed создает блок, в котором фиксируется указатель на поле объекта person. После завершения блока fixed закрепление с переменных снимается, и они могут быть подвержены сборке мусора.

Кроме адреса переменной можно также инициализировать указатель, используя массив, строку или буфер фиксированного размера:

```
            unsafe
            {
                int[] nums = { 0, 1, 2, 3, 7, 88 };
                string str = "Привет мир";
                fixed (int* p = nums)
                {
                    int third = *(p + 2); // получим третий элемент
                    Console.WriteLine(third); // 2
                }
                fixed (char* p = str)
                {
                    char forth = *(p + 3); // получим четвертый элемент
                    Console.WriteLine(forth); // в
                }
            }
```

При инициализации указателей на строку следует учитывать, что указатель должен иметь тип char*.

Полный код программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication5
{
    class Program
    {
        //Оператор fixed и закрепление указателей
        static void Main(string[] args)
        {
            //Зафиксируем указатель с помощью оператора fixed:
            unsafe
            {
                Person person = new Person();
                person.age = 28;
                person.height = 178;
                // блок фиксации указателя
                fixed (int* p = &person.age)
                {
                    if (*p < 30)
                    {
                        *p = 30;
                    }
                }
                Console.WriteLine(person.age); // 30
            }
            Console.ReadKey();
```

```
            //инициализировать указатель, используя массив
            unsafe
            {
                int[] nums = { 0, 1, 2, 3, 7, 88 };
                string str = "Привет мир";
                fixed (int* p = nums)
                {
                    int third = *(p + 2); // получим третий элемент
                    Console.WriteLine(third); // 2
                }
                fixed (char* p = str)
                {
                    char forth = *(p + 3); // получим четвертый элемент
                    Console.WriteLine(forth); // в
                }
            }
            Console.ReadKey();


        }
        public class Person
        {
            public int age;
            public int height;
        }

    }
}
```

**Задание для самостоятельного выполнения:** Создать приложение, для решения задач.

**Вариант 1.** Сформировать очередь из 8 чисел. Найти произведение 3-го и 4-го чисел из очереди. Результат поместить в очередь.

**Вариант 2.** Заполнить очередь 10 случайными числами из интервала [10;80]. Найти количество чисел, которые при делении на 7 дают в остатке 4.

**Вариант 3.** Сформировать очередь из 8 чисел. Найти сумму 2-го и 4-го чисел из очереди. Результат поместить в очередь.

**Вариант 4.** Заполнить очередь 8 случайными числами из интервала [20; 70]. Найти среднее арифметическое 2-го и 4-го чисел очереди.

**Вариант 5.** Заполнить очередь 15 случайными числами из интервала [0;100]. Найти среднее арифметическое нечетных чисел. Результат поместить в очередь.

**Вариант 6.** Сформировать очередь из 18 чисел. Найти сумму 12-го и 14-го чисел из очереди. Результат поместить в очередь.

**Вариант 7.** Заполнить очередь 18 случайными числами из интервала [10; 50]. Найти среднее арифметическое 12-го и 13-го чисел очереди.

**Вариант 8.** Сформировать очередь из 12 чисел. Найти среднее арифметическое первых 8 чисел из очереди. Результат поместить в очередь.

**Вариант 9.** Сформировать очередь из 17 чисел. Найти модуль разности между 12-м и 13-м числом очереди.

**Вариант 10.** Сформировать очередь из 6 чисел. Найти произведение первых 3 чисел из очереди.

**Вариант 11.** Сформировать очередь из 8 чисел. Найти среднее арифметическое первых 4 чисел из очереди. Результат поместить в очередь.

**Вариант 12.** Заполнить очередь 10 случайными числами из интервала [-40; 50]. Найти количество положительных чисел. Результат поместить в очередь.

**Вариант 13.** Сформировать очередь из 10 чисел. Найти произведение 3-го, 4-го и 5-го чисел из очереди.

**Вариант 14.** Заполнить очередь 8 случайными числами из интервала [0;50]. Найти среднее арифметическое четных чисел. Результат поместить в очередь.

**Вариант 15.** Заполнить очередь 8 случайными числами из интервала [-20;50]. Найти среднее арифметическое 2-го и 3-го чисел очереди. Результат поместить в очередь.

**Вариант 16.** Сформировать очередь из 7 чисел. Найти модуль разности между 2-м и 3-м числом очереди.

**Вариант 17.** Сформировать очередь из 8 чисел. Утроить модуль разности между 2-м и 3-м числом очереди. Результат поместить в очередь.

**Вариант 18.** Сформировать очередь из 10 чисел. Найти сумму первых 5 чисел из очереди. Результат поместить в очередь.

**Вариант 19.** Сформировать очередь из 8 чисел. Найти произведение первых 4 чисел из очереди.

**Вариант 20.** Сформировать очередь из 8 чисел. Найти сумму 3-го, 4-го и 5-го чисел из очереди.

**Контрольные вопросы**

1. Дать понятие указателей для организации связанных списков.
2. Принцип работы указателей для организации связанных списков и.
3. Основные операции над указателями для организации связанных списков.