Тема 3. ПРОГРАММИРОВАНИЕС ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ. РАЗРАБОТКА РЕКУРСИВНЫХ АЛГОРИТМОВ. ИСПОЛЬЗОВАНИЕ УКАЗАТЕЛЕЙ НА ФУНКЦИИ.

Цель занятия: научиться создавать рекурсивные функции, передавать функцию в качестве аргумента другой функции.

Методические указания

При подготовке к занятию необходимо изучить: правила записи и способы передачи параметров рекурсивных функций и способы обращения к ним; правила записи и порядок выполнения программ, использующих рекурсивные функции; правила использования указателей на функции.

Теоретические сведения

Рекурсивные функции

Функции, вызывающие сами себя, называются рекурсивными.

Пример 3.1. Использование рекурсии для вывода стихотворения:

10 негритят пошли купаться в море,
10 негритят резвились на просторе.
Один из них пропал – и вот вам результат:
9 негритят пошли купаться в море,
9 негритят резвились на просторе.
Один из них пропал – и вот вам результат:
…
Нет больше негритят!

```
#include <conio.h>
#include <iostream>
#include <windows.h>
using namespace std;
void negr(int n)
{
if(n == 0) cout<<"Нет больше негритят!";
else
        {
          cout<<n<<" негритят пошли купаться в море \n";
          cout<<n<<" негритят резвились на просторе\n";
          cout<<"Один из них пропал – и вот вам результат:\n";
          cout<<"\n";
          negr(n-1);
        }
}
int main()
{
        SetConsoleOutputCP(1251);
        negr(10);
        getch();
}
```

Распространённым примером рекурсивной функции является функция, вычисляющая факториал:

Пример 3.2.

```
int fac(int n)
{
        if(n == 1)
        return 1;
        else
        return  n*fac(n -1);
}
// не рекурсивная функция
int facn(int n)
{
```

```
        int fact = 1;
        for(int m = 1; m <= n; m++)
        fact *= m;
        return fact;
}
```

В рекурсивной функции обязательно должно быть условие выхода из рекурсии, при котором функции не нужно вызывать саму себя. В примере с факториалом это

```
if(n == 1) return 1;
```

Рекурсивный вызов внутри функции должен изменять аргументы исходной функции (в приведённом примере вычисляется факториал числа, на 1 меньшего).

Реализация механизма рекурсии. Переменные функции хранятся в сегменте стека (локальной памяти). При каждом рекурсивном вызове функции все локальные данные тоже помещаются в стек (данные первого вызова функции остаются в сегменте стека). Этот процесс называется прямым ходом рекурсии или рекурсивным спуском.

Прямой ход рекурсии продолжается до достижения условия выхода из рекурсии, после чего запускается процесс обратного хода рекурсии (рекурсивного подъёма). При этом из стека вынимаются сохранённые ранее значения и используются для последующих вычислений.

Структура рекурсивной функции может принимать три разных формы:

- форма с выполнением действий до рекурсивного вызова (на рекурсивном спуске):

```
void Rec(void) { S; if (условие) Rec(); }
```

- форма с выполнением действий после рекурсивного вызова (на рекурсивном возврате — подъеме):

```
void Rec(void) { if (условие) Rec(); S; }
```

- форма с выполнением действий как до (на рекурсивном спуске), так и после рекурсивного вызова (на рекурсивном возврате):

```
void Rec(void) { S1; if (условие) Rec(); S2; }
```

Названия «рекурсивный спуск» и «рекурсивный подъем» связаны с понятием глубины рекурсии – функция спускается на глубину рекурсии и поднимается «оттуда». Глубина рекурсии – это максимальная степень вложенности рекурсивных вызовов.

В принципе, любой цикл можно заменить эквивалентной рекурсивной программой.

Рекурсивные варианты большинства функций выполняются медленнее, чем их итеративные (не рекурсивные) варианты, так как при рекурсии тратится время на вызовы функции самой себя.

Пример 3.3. Используя рекурсивную функцию, вычислить сумму S=1+1/2+1/3+...+1/n

```
#include <conio.h>
#include <iostream>
using namespace std;
void SumRec(int n, double& sum)
{
        if(n==1)
                sum= 1;
        else
                SumRec(n-1, sum);
           sum= sum + 1.0/n; // будет выполнено на обратном ходе рекурсии
}
int main()
{
        cout<<"Input n:\n";
        int n;
        cin>>n;
        double sum;
        SumRec(n, sum);
        cout<<"sum="<<sum;
        getch();
}
```

Указатели на функции

Пусть есть функция такого вида:

```
int Compare( const string &sl, const string &s2 );
```

Указатель на нее объявляется следующим образом:

```
int (*pf)( const string &, const string & );
```

pf объявлен как указатель на функцию с двумя параметрами, возвращающую значение типа int, т.е. такую, как Compare(). pf способен адресовать и приведенную ниже функцию, поскольку ее сигнатура совпадает с типом Compare():

```
int sizeCompare( const string &sl, const string &s2 );
```

Функции calc() и gcd() другого типа, поэтому pf не может указывать на них:

```
int calc( int , int );
int gcd( int , int );
```

Указатель, который адресует эти две функции, определяется так:

```
int (*pfi)( int, int );
```

Указатель на функцию инициализируется следующим образом:

```
int (*pfi)( const string &, const string & ) = lexicoCompare;  // или
int (*pfi2)( const string &, const string & ) = &lexicoCompare;
```

Ему можно присвоить значение:

```
pfi = lexicoCompare;
pfi2 = pfi;
```

Инициализация и присваивание корректны только тогда, когда список параметров и тип значения, которое возвращает функция, адресованная указателем в левой части операции присваивания, в точности соответствуют списку параметров и типу значения, возвращаемого функцией или указателем в правой части. В противном случае выдается сообщение об ошибке компиляции. Никаких неявных преобразований типов для указателей на функции не производится.

Указатель на функцию применяется для вызова функции, которую он адресует. Включать оператор разыменования при этом необязательно. И прямой вызов функции по имени, и косвенный вызов по указателю записываются одинаково.

Пример 3.4. Демонстрация прямого и косвенного вызова функции нахождения минимального значения в массиве.

```
int min( int*, int );
int (*pf)( int*, int ) = min;

const int iaSize = 5;
int ia[ iaSize ] = { 7, 4, 9, 2, 5 };

int main() {
   cout << "Прямой вызов: min: "
        << min( ia, iaSize ) << endl;

   cout << "Косвенный вызов: min: "
        << pf( ia, iaSize ) << endl;
           getch();
   return 0;
}
int min( int* ia, int sz ) {
   int minVal = ia[ 0 ];
   for ( int ix = 1; ix < sz; ++ix )
      if ( minVal > ia[ ix ] )
         minVal = ia[ ix ];
   return minVal;
}
```

Вызов

pf(ia, iaSize);

может быть записан также и с использованием явного синтаксиса указателя:

(*pf)(ia, iaSize);

Результат в обоих случаях одинаковый, но вторая форма говорит о том, что вызов осуществляется через указатель на функцию. Можно объявить массив указателей на функции:

```
int (*testCases[10])();
```

testCases – это массив из десяти элементов, каждый из которых является указателем на функцию, возвращающую значение типа int и не имеющую параметров.

Массив указателей на функции может быть инициализирован списком, каждый элемент которого является функцией. Например:

```
int lexicoCompare(  const string &, const string &);
int sizeCompare(  const string &, const string &);

int ( *PFI2S[2]  )( const string &, const string & ) = {lexicoCompare,   sizeCompare};
```

Пример 3.5. Написать функцию bitcount, считающую число единичных разрядов в целом беззнаковом аргументе. Вызвать функцию через указатель на неё.

```
#include <conio.h>
#include <iostream>
using namespace std;
int bitcount(unsigned int n)
{
        int b= 0;
// n >>= 1 - сдвиг числа вправо на 1 бит, освобождающиеся позиции заполнятся нулями
        for(; n != 0; n >>= 1)
                if(n  & 1 != 0) //& - побитовое умножение
                        b++;
        return b;
}
int main()
{
        cout<<"Input n:\n";
        unsigned int n;
        cin>>n;
        int k = bitcount(n);
        cout<<k<<"  digits = \'1\'\n";
        // указатель на функцию bitcount()
        int (*p)(unsigned int n);
        p = &bitcount;
        cout<<"Input n:\n";
        cin>>n;
        //вызвать функцию через указатель
        k = p(n);
        cout<<k<<"  digits = \'1\'\n";
        getch();
        return 0;
}
```

Пример 3.6. Напишите функцию printArray(), которая меняет местами элементы числового массива и выводит массив на экран. В качестве параметров в функцию передавать массив, количество элементов массива и указатель на функцию, меняющую порядок следования элементов массива.

```
#include <conio.h>
#include <time.h>
#include <iostream>
using namespace std;
void printArray(int arr[],  int length, void (*change_array)(int m[],  int len))
{
```

```
	cout<<"Old array:\n";
	for(int i = 0; i<length; i++)
		cout<<arr[i]<<"  ";
	cout<<"\n";
	//вызвать через указатель функцию, которая поменяет порядок следования элементов в массиве
	change_array(arr, length);
	cout<<"Changed array:\n";
	for(int i = 0; i<length; i++)
		cout<<arr[i]<<"  ";
	cout<<"\n";
}
//меняет порядок следования элементов массива на обратный
void reverse(int m[], int size)
{
	for(int i = 0; i<size/2; i++)
	{
		int tmp  = m[i];
		m[i]  = m[size-i-1];
		m[size-i-1]=tmp;
	}
}
//меняет элементы массива случайным образом
void shuffle(int m[], int size)
{
	for(int i = 0; i<size; i++)
	{
		srand(time(0));  //инициализировать генератор псевдослучайных чисел любым числом
		int pos = rand()%size;//новый номер i-го элемента массива
		int tmp  = m[i];
		m[i]  = m[pos];
		m[pos]=tmp;
	}
}
int main()
{
	int m[] = {1,2,3,4,5,6,7,8,9,10};
	cout<<"Reversed array:\n";
	printArray(m, 10, reverse);
	cout<<"Random order:\n";
	printArray(m, 10, shuffle);
	getch();
	return 0;
}
```

Методические указания

При подготовке к занятию необходимо изучить:

- способы реализации механизма рекурсии;
- правила использования указателей на функции.

Аудиторные и домашние задания

1. Вывести на экран в порядке возрастания все четные числа из диапазона, границы которого вводятся с клавиатуры. Границы могут вводиться в произвольном порядке. Вывод должен осуществляться рекурсивной функцией.

2. Найти сумму цифр целого числа, вводимого с клавиатуры. Суммирование должно осуществляться рекурсивной функцией.

3. Разработать рекурсивную функцию, которая проверяет, является ли введенное с клавиатуры натуральное число простым.

4. Найти наибольший общий делитель двух натуральных чисел, вводимых с клавиатуры, с помощью алгоритма Евклида (Большее из чисел заменяется разностью этих чисел. Этот процесс повторяется до тех

пор, пока не останется одно ненулевое число. Это число и будет наибольшим общим делителем). Использовать рекурсивный алгоритм.

5. Возвести целое число в целую степень. Число и степень вводятся с клавиатуры. Все умножения должны выполняться рекурсивной функцией.

6. Разработать функцию, которая находит наименьшее общее кратное двух натуральных чисел, вводимых с клавиатуры (НОК(a,b)=(a*b)/НОД(a,b)). НОД(a,b) – наибольший общий делитель; найти с помощью алгоритма Евклида (Большее из чисел заменяется разностью этих чисел. Этот процесс повторяется до тех пор, пока не останется одно ненулевое число. Это число и будет наибольшим общим делителем). Объявить указатель на функцию, определяющую НОК. Вызвать функцию через этот указатель.

7. Написать функцию, которая вводит строку и определяет, является ли она палиндромом. Палиндром – это число или текст, который одинаково читается слева направо и справа налево. Например, 12321, 55555. Объявить указатель на функцию. Вызвать функцию через этот указатель.

8. Даны функции:

```
void swap(int& left, int& right)
{
// поменять значения аргументов
	int buffer = left;
	left = right;
	right = buffer;
}

// 2 функции сравнения:
bool compare_less(int left, int right)
{
	if (left<right)
		return true;
	else
		return false;
}

bool compare_greater(int left, int right)
{
	if (left>right)
		return true;
	else
		return false;
}
```

Напишите функцию sort() для сортировки числового массива. Для сравнения элементов массива в ней должна быть вызвана одна из приведенных выше функций. Чтобы поменять элементы массива местами – вызывать функцию swap(). Передавать в функцию sort() в качестве параметров массив для сортировки, его размер, указатель на функцию сравнения и указатель на функцию swap().

9. Написать функцию multiple, которая определяет для пары целых чисел, кратно ли второе число первому. Функция должна воспринимать два целых аргумента и возвращать 1 (истина), если второе число кратно первому, и 0 (ложь), если нет. Напишите объявление указателя на нее. Вызовите функцию через этот указатель.

10. Функции из библиотеки C, определенные в заголовочном файле <cmath>:

```
double abs(double);
	double sin(double);
	double cos(double);
	double sqrt(double);
```

Объявите массив указателей на функции такого вида и инициализируйте его этими четырьмя функциями. Вызовите эти функции по очереди через указатели.