МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

Отчет

Практическая работа №6

Дисциплина Структуры и алгоритмы обработки данных

Тема. Рекурсивные алгоритмы и их реализация

Выполнил студент Черномуров С. А.

Фамилия И.О.

Группа ИКБО-13-21

Номер группы

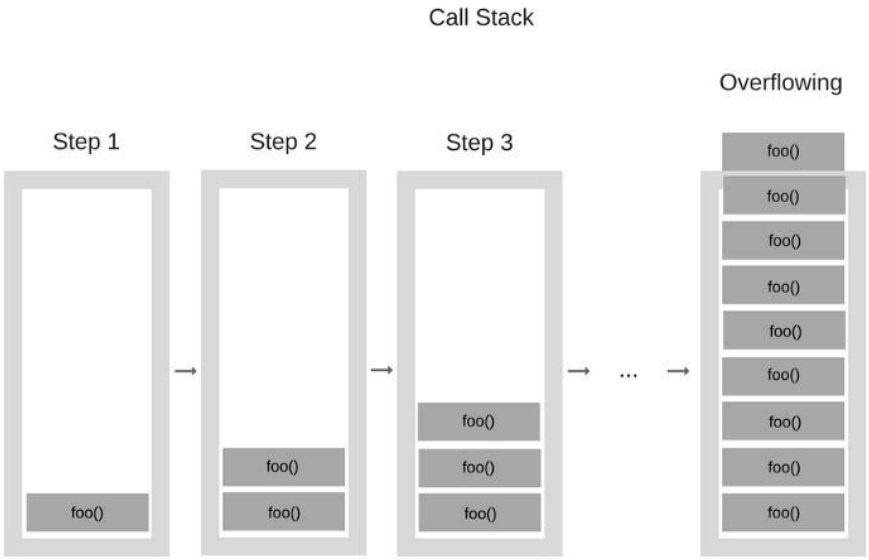
Москва 2022

Ответы на вопросы:

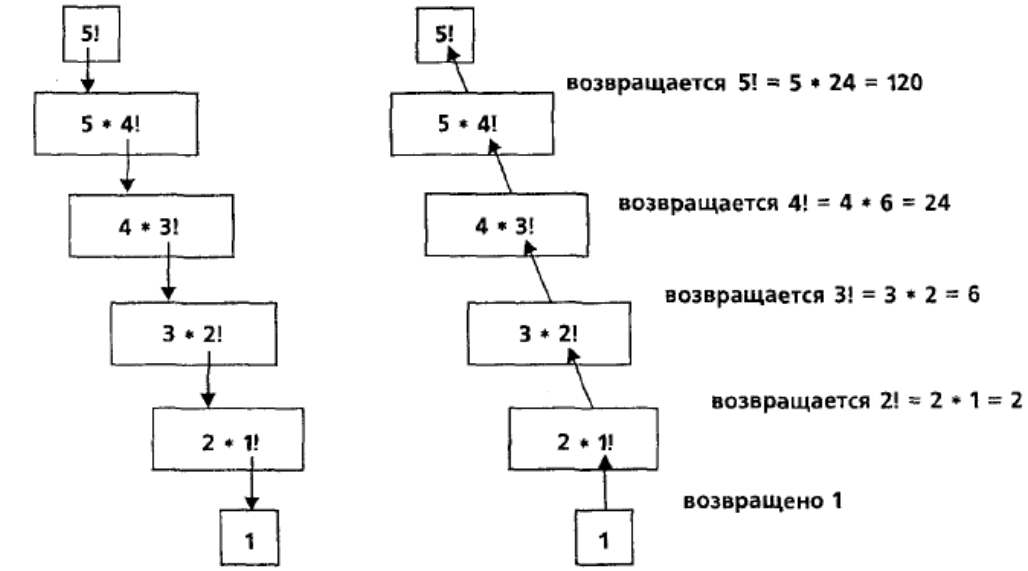
1) **Определение рекурсивной функции:**
   Рекурсивная функция – это числовая функция f(n) числового аргумента, которая в своей записи содержит себя же.
   Нисходящая рекурсия последовательно разбивает рассматриваемую задачу на все более простые, пока не доходит до терминальной ситуации. Под терминальной ситуацией принято понимать ситуацию, когда не требуется продолжения рекурсии. В этом случае значение определяемой функции получается без использования обращения к ней (применительно к другим значениям аргумента), характерного для рекурсивных определений. Только после этого начинается формирование ответа, а промежуточные результаты передаются обратно - вызывающим функциям.
   В восходящей рекурсии промежуточные результаты вычисляются на каждой стадии рекурсии, так что ответ строится постепенно и передается в виде параметра рабочей памяти до тех пор, пока не будет достигнута терминальная ситуация. К этому времени ответ уже готов, и нужно только передать его вызывающей функции верхнего уровня.
2) **Шаг рекурсии** – это правило, в теле которого обязательно содержится вызов определяемого предиката (вызов рекурсивной функцией самой себя, но от других параметров).
3) **Глубина рекурсии** – наибольшее одновременное количество рекурсивных обращений функции, определяющее максимальное количество слоев рекурсивного стека, в котором осуществляется хранение отложенных вычислений.
4) **Условие завершения рекурсии** – это условие, при выполнении которого рекурсивная функция перестает вызывать саму себя.
5) **Виды рекурсии**:
   Линейная рекурсия – это рекурсия, при которой исполнение программы приводит только к одному вызову этой же самой подпрограммы.
   Каскадная рекурсия – это рекурсия, при которой вызов рекурсивной функции по любой из всех возможных ветвей алгоритма встречается более одного раза.
6) **Прямая и косвенная рекурсия**:
   Прямая рекурсия – это вид рекурсии, при котором происходит непосредственное обращение рекурсивной функции к самой себе, но с иным набором входных данных.
   Косвенная рекурсия – это последовательность взаимных вызовов нескольких функций, организованная в виде

циклического замыкания на тело первоначальной функции, но с иным набором параметров.

7) **Организация стека рекурсивных вызовов**:
   Вся иерархия вызовов хранится в специальной области памяти, называемой *стеком вызовов*. Элементы в этот участок памяти добавляются по принципу LIFO: последний добавленный элемент должен быть извлечён первым. Когда метод вызывает сам себя, новым локальным переменным и параметрам выделяется место в стеке и код метода выполняется с этими новыми начальными значениями. При каждом возврате из рекурсивного вызова старые локальные переменные и параметры удаляются из стека, и выполнение продолжается с момента вызова внутри метода.

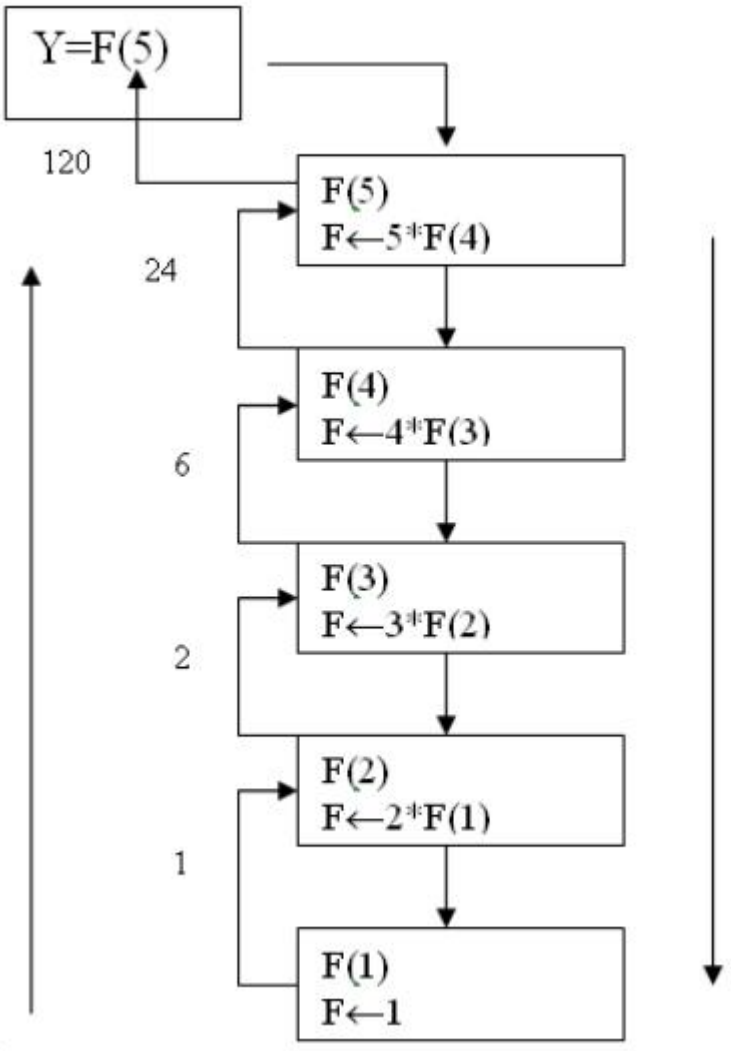


8) **Стек рекурсивных вызовов. Модель формирования для алгоритма вычисления n!:**



а) Процесс рекурсивных вызовов **б) Значения, возвращаемые после каждого рекурсивного вызова**

Рекурсивное вычисление 5!

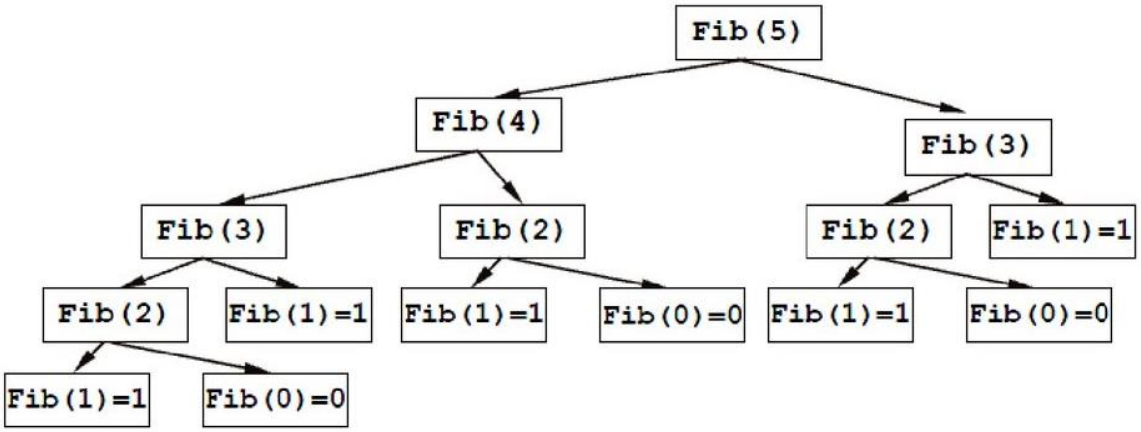



```
int F(int n){
  if(n<2) return 1;
     else return F(n-1)*n;
```

```
}
```

9) **Дерево рекурсии вычисления 5! (приведено выше) и пятого числа Фибоначчи:**

```
//вычисление n-го члена ряда Фибоначчи
int F(int n){
if (n==0) return 0;
  if(n>0 && n<3) return 1;
     else return F(n-1)+F(n-2);
}
```



Задание 1

Вариант №23

1. Условие задачи: разработать рекурсивную функцию, которая реализует синтаксический анализатор для понятия идентификатор.

$$identifier ::= \left\{ \begin{matrix} letter \\ identifier \left\{ \begin{matrix} digit \\ letter \end{matrix} \right. \end{matrix} \right\}$$

2. Постановка задачи:
   Дано. s – строка, которую нужно проанализировать на то, является ли она идентификатором
   Результат. true – если строка является идентификатором, false - в противном случае.
3. Рекуррентная зависимость:

$$identifier ::= \left\{ \begin{matrix} letter \\ identifier \left\{ \begin{matrix} digit \\ letter \end{matrix} \right. \end{matrix} \right\}$$

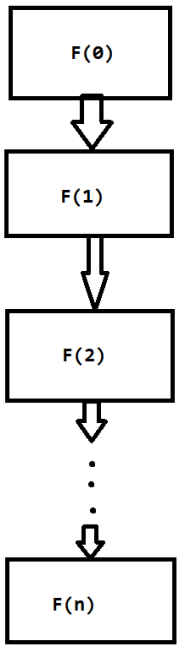
Первым символом идентификатора обязательно является буква латинского алфавита, последующие символы могут быть как буквой, так и цифрой.

4. Функция, используемая для решения задачи:
   Проверка строки на пригодность в качестве идентификатора:
   Предусловие. s – строка, которую нужно проверить, pos – текущая позиция строки, которая проверяется (по умолчанию равна нулю).
   Постусловие. true – если строка является идентификатором, false – в противном случае.
   ```
   bool is_identifier(string s, int pos);
   ```
   Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Bsa7FG	true
2	5A	false

```
bool is_identifier(string s, int pos) {
	if (pos == 0) {
		char symbol = s[pos];
		if ((symbol >= 'a') && (symbol <= 'z') || (symbol >= 'A') &&
(symbol <= 'Z'))
			return is_identifier(s, pos + 1);
		else return false;
	}
	else if (pos < s.size()) {
		char symbol = s[pos];
		if ((symbol >= 'a') && (symbol <= 'z') || (symbol >= 'A') &&
(symbol <= 'Z') || (symbol >= '0') && (symbol <= '9'))
			return is_identifier(s, pos + 1);
		else return false;
	}
	else return true;
}
```

5. Определение глубины рекурсии:
   Так как рекурсия восходящая, то для расчета следующего значения используется лишь одно предыдущее, значит глубина рекурсии m=1.
6. Определение временной сложности рекурсивного алгоритма методом подстановки:
   $T_n=T_0+T_1+\ldots+T_{n-1}=O(1)+O(1)+\ldots+O(1)=n*O(1)=O(n)$
7. Определение сложности рекурсивного алгоритма с помощью дерева рекурсии:



Временная сложность алгоритма – O(n)

Сложность по памяти – O(1)

8. Схема рекурсивных вызовов для одного из значений:
   s=”5A”

F(s, 0) – return false (первый символ не является буквой латинского алфавита)

Далее рекурсивных вызовов не происходит.

Тестирование программы

```
Лабораторная работа №6 ИКБО-13-21 Черномуров С.А. Вариант 23

Выберите номер задания:
1) Разработать рекурсивную функцию, которая реализует синтаксический анализатор для понятия идентификатор
2) Сформировать новый список из значений узлов исходного, занеся в него значения в обратном порядке
0) Закончить программу
1
```

```
Введите строку, которую нужно проанализировать:
5A
Строка не является идентификатором
```

```
Введите строку, которую нужно проанализировать:
Bsa7FG
Строка является идентификатором
```

Задание 2

Вариант №23

1. Условие задачи: дан линейный однонаправленный список. Сформировать новый список из значений узлов исходного, занеся в него значения в обратном порядке
2. Постановка задачи:
   Дано. Список l – исходный список.
   Результат. Список m – новый сформированный список, в который занесены значения списка l в обратном порядке
3. Рекуррентная зависимость:
   l[n-1]=m[0]
   l[n-2]=m[1]
   .
   .
   .
   l[0]=m[n-1]
   l[n-k]=m[k-1]
4. Функции, используемые для решения задачи:

   1) Проверка наличия узлов в списке:

   Предусловие. Список.

   Постусловие. true – если список пустой, false – если список не пустой.

```
bool is_empty();
```

   Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Список с указателем first=nullptr	true
2	Список с указателем first!=nullptr	false

```
bool list::is_empty() { //
	if (first == nullptr) return true;
	else return false;
}
```

   2) Присоединение узла в конец списка:

   Предусловие. Список list, _data – числовое значение, которое будет записано в узел.

   Постусловие. Список с присоединенным в конец узлом.

```
void push_back(int _data);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Пустой список, _data=5, size=0	Список, first=указатель на первый узел, first->data=5, size=1
2	Список с указателем first!=nullptr, _data=10, size=3	Список, first=указатель на первый узел, последний узел списка->data=10, size=4

```
void list :: push_back(int _data) {
	Node* pointer = new Node(_data);
	if (is_empty()) {
		first = pointer;
		size++;
	}
	else {
		Node* temp = first;
		while (temp != nullptr) {
			if (temp->next == nullptr) { temp->next = pointer; break; }
			else temp = temp->next;
		}
		temp = new Node(_data);
		size++;
	}

}
```

3) Вывод списка на экран:

Предусловие. Список list.

Постусловие. Список list, выведенный на экран.

```
void print();
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Список из 5 узлов	Выведенный на экран список из 5 узлов
2	Пустой список	void

```
void list :: print() {
	if (is_empty()) return;
	Node* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2)<<pointer->data << " ";
		pointer = pointer->next;
	}
```

4) Генерация списка определенной длины:

Предусловие. len – длина генерируемого списка.

Постусловие. Сгенерированный список list длины len.

```
void generate(int len);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	len=5	Список из 5 узлов, size=5
2	len=1	Список из 1 узла, size=1

```
void list::generate(int len) { //

	for (int i = 0; i < len; i++) {
		int value = rand() % 100;
		this->push_back(value);
	}
}
```

5) Заполнение нового списка значениями исходного в обратном порядке:

Предусловие. l – исходный список, n – длина исходного списка, m – заполняемый список.

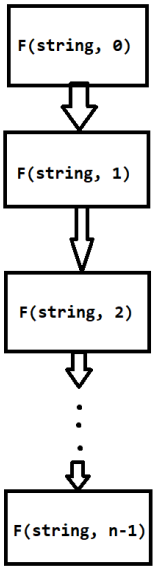
Постусловие. заполненный список m значениями списка l в обратном порядке.

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	l = {1, 2, 3, 4}, n = 4, m	m = {4, 3, 2, 1}
2	l = {6, 5, 4, 9, 5}, n = 5, m	m = {5, 9, 4, 5, 6}

```
void reverse2(list l, int n, list& m) {
	if (n <= 0) return;
	else {
		Node* pointer = l.first;
		for (int i = 0; i < n-1; i++) {
			pointer = pointer->next;
		}
		m.push_back(pointer->data);
		reverse2(l, n - 1, m);
	}
}
```

5. Определение глубины рекурсии:
   Так как алгоритм рекурсии не имеет ветвлений, рекурсия восходящая, и при каждом вызове функции значение pos просто инкрементируется, то глубина рекурсии m=1.
6. Определение временной сложности рекурсивного алгоритма методом подстановки:
   $T_n=T_{pos=0}+T_{pos=1}+\ldots+T_{pos=n-1}=O(1)+O(1)+\ldots+O(1)=n*O(1)=O(n)$
7. Определение сложности рекурсивного алгоритма с помощью дерева рекурсии:



   Временная сложность алгоритма – O(S(n)), где S(n)=(1+n)*n/2 (сумма арифметической прогрессии от 1 до n)

   Сложность по памяти – O(1)

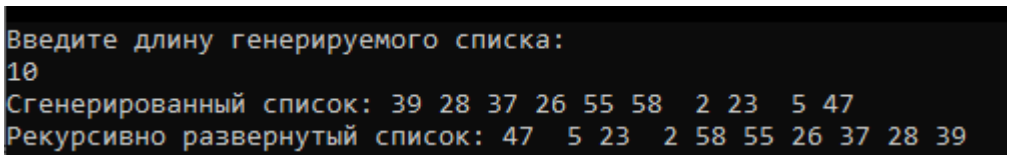
8. Схема рекурсивных вызовов для одного из значений:
   l = {1, 2}, n=2;
   reverse2(l, 2, m)
   l[1]=m[0] reverse2(l, 1, m)
   l[0]=m[1] reverse2(l, 0, m)
   return void; - программа прошла по всему списка от конца и до начала, новый список сформировался

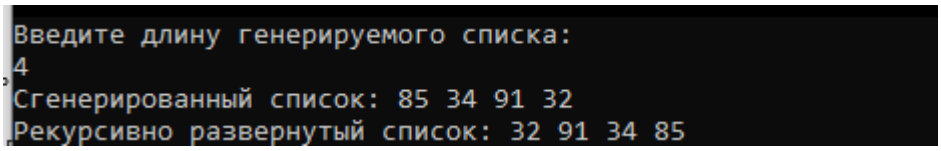
Далее рекурсивных вызовов не происходит.

Тестирование программы

```
Лабораторная работа №6 ИКБО-13-21 Черномуров С.А. Вариант 23

Выберите номер задания:
1) Разработать рекурсивную функцию, которая реализует синтаксический анализатор для понятия идентификатор
2) Сформировать новый список из значений узлов исходного, занеся в него значения в обратном порядке
0) Закончить программу
2
```





Полный код программы на языке C++

Файл functions.h (описаны функции и их тела, а также структура списка и узла списка)

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

bool is_identifier(string s, int pos) {
	if (pos == 0) {
		char symbol = s[pos];
		if ((symbol >= 'a') && (symbol <= 'z') || (symbol >= 'A') && (symbol <= 'Z'))
			return is_identifier(s, pos + 1);
		else return false;
	}
	else if (pos < s.size()) {
		char symbol = s[pos];
		if ((symbol >= 'a') && (symbol <= 'z') || (symbol >= 'A') && (symbol <= 'Z') || (symbol >= '0') && (symbol <= '9'))
			return is_identifier(s, pos + 1);
		else return false;
	}
	else return true;
}

struct Node {
	Node* next = nullptr;
	int data;
	Node(int _data) : data(_data), next(nullptr) {}
};

struct list {
	Node* first;
	int size = 0;
	list() : first(nullptr) {}

	bool is_empty();
	void push_back(int _data);
	void print();
	void generate(int len);
};

bool list::is_empty() { //
	if (first == nullptr) return true;
	else return false;
}

void list::push_back(int _data) { //
```

```
	Node* pointer = new Node(_data);
	if (is_empty()) {
		first = pointer;
		size++;
	}
	else {
		Node* temp = first;
		while (temp != nullptr) {
			if (temp->next == nullptr) { temp->next = pointer; break; }
			else temp = temp->next;
		}
		temp = new Node(_data);
		size++;
	}

}

void list::print() { //
	if (is_empty()) return;
	Node* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2) << pointer->data << " ";
		pointer = pointer->next;
	}
}

void list::generate(int len) { //

	for (int i = 0; i < len; i++) {
		int value = rand() % 100;
		this->push_back(value);
	}
}

void reverse2(list l, int n, list& m) {
	if (n <= 0) return;
	else {
		Node* pointer = l.first;
		for (int i = 0; i < n-1; i++) {
			pointer = pointer->next;
		}
		m.push_back(pointer->data);
		reverse2(l, n - 1, m);
	}
}
```

Файл main.cpp (основной алгоритм программы)

```
#include <iostream>
#include <string>
#include "functions.h"
#include "random"
using namespace std;

int main() {
	srand(time(0));
	setlocale(LC_ALL, "");

	cout << "Лабораторная работа №6 ИКБО-13-21 Черномуров С.А. Вариант 23" << endl
<< endl;
	cout << "Выберите номер задания:\n1) Разработать рекурсивную функцию, которая
реализует синтаксический анализатор для понятия идентификатор\n2) Сформировать новый
```

```
список из значений узлов исходного, занеся в него значения в обратном порядке \n0)
Закончить программу\n";

	int choice1;

	do {
		cin >> choice1;

		if (choice1 != 1 && choice1 != 2 &&  choice1 != 0) cout << "Введено
неверное значение, попробуйте снова.\n";
	} while (choice1 != 1 && choice1 != 2  && choice1 != 0);

	system("cls");

	switch (choice1) {

	case 1: {
		cout << "Введите строку, которую нужно проанализировать:\n";

		string s;
		cin.ignore(32767, '\n');
		getline(cin, s);

		if (is_identifier(s, 0)) cout << "Строка является идентификатором\n\n";
		else cout << "Строка не является идентификатором\n\n";
		break; }

	case 2: {
		list l, m;
		cout << "Введите длину генерируемого списка:\n";

		int list_len;
		cin >> list_len;

		l.generate(list_len);

		cout << "Сгенерированный список: ";
		l.print();
		cout << "\n";

		reverse2(l, list_len, m);

		cout << "Рекурсивно развернутый список: ";
		m.print();
		cout << "\n";

		break; }

	case 0:
		return 0;
	}
	main();
}
```

Вывод

В ходе работы было выполнено задание в соответствии с поставленным вариантом. Были получены знания и практические навыки по разработке и реализации рекурсивных процессов.