# Практическая работа № 10

**АЛГОРИТМ ХАФФМАНА**

**Постановка задачи**

Составить программу, реализующую кодирование строки с помощью алгоритма Хаффмана

1. **Описание алгоритма**

Алгоритм реализован с использованием классов Haffman и Los и одной структуры, являющейся узлом перевернутого дерева.

Методы класса Huffman:

**void set\_string** - функция записывает строку в память.

**void make\_tree** - функция создает дерево шифрования.

**void compress** – функция зашифровывает строку

**string get\_code** – функция возвращает полученный код

Методы класса Los:

**void add\_ell** – функция добавляет элемент в список

**Node \*find\_ell** – функция возвращает указатель на элемент по ключу

**void inc\_ell** – функция инкрементирует поле number структуры

**int count\_true** – функция возвращает количество записей, в которых поле is\_last\_layer является истиной.

**int \*pare\_min** – возвращает массив из 2-х id записей с наименьшими полями number.

**int get\_id** – функция возвращает id записи

**void inc\_id** – функция инкрементирует поле counter\_id в классе Los

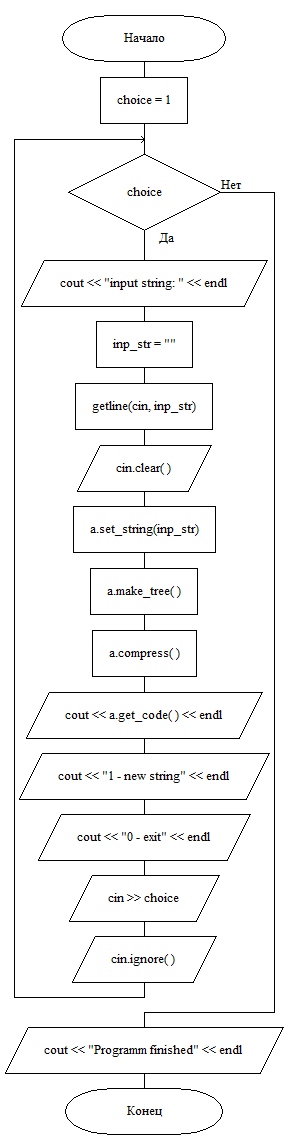


Рис.1 Схема алгоритма функции main

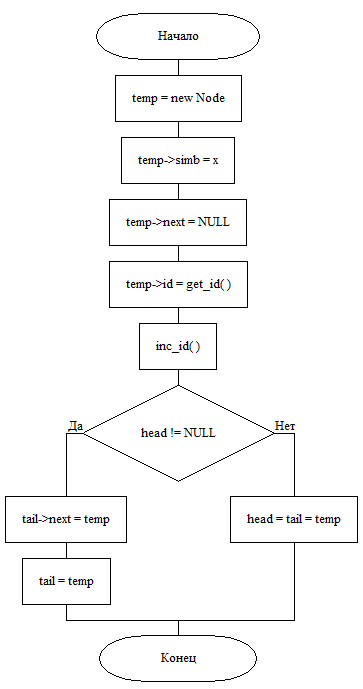


Рис.2 Схема алгоритма функции add\_ell(char x)

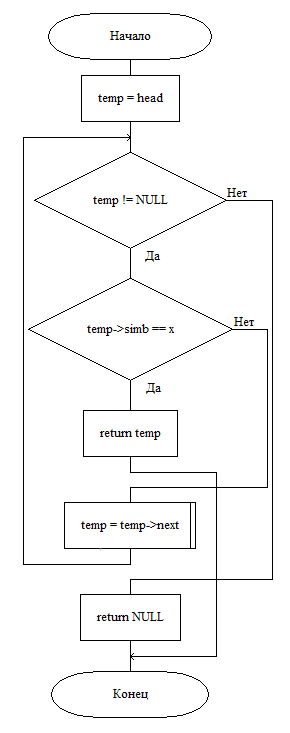


Рис.3 Схема алгоритма функции find\_ell(char x)

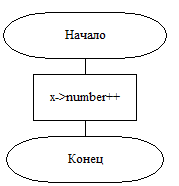


Рис.4 Схема алгоритма функции inc\_ell(Node \*x)

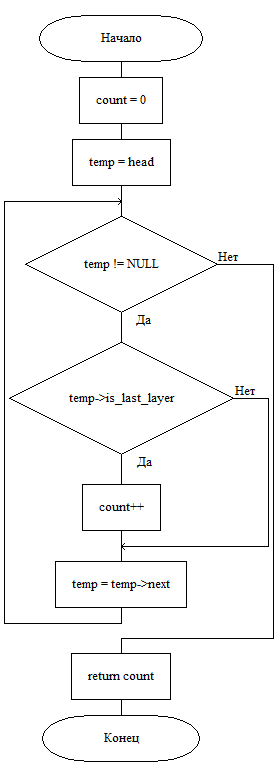


Рис.5 Схема алгоритма функции count\_true()

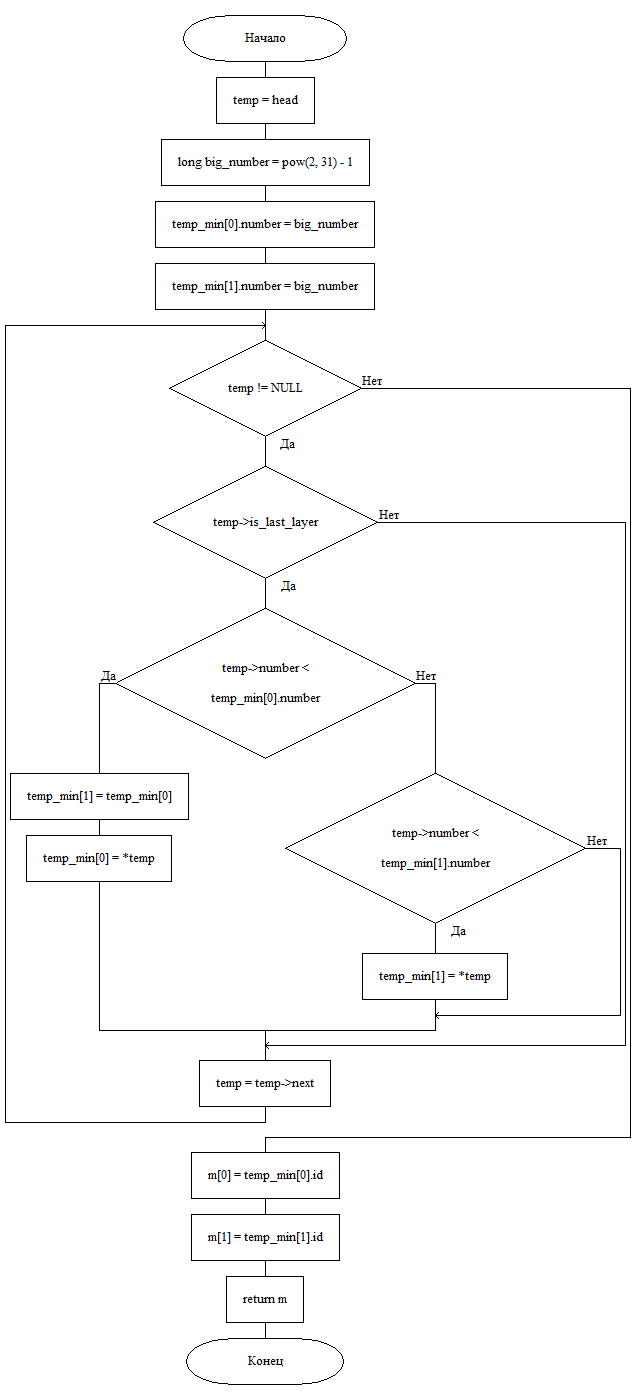


Рис.6 Схема алгоритма функции pare\_min()

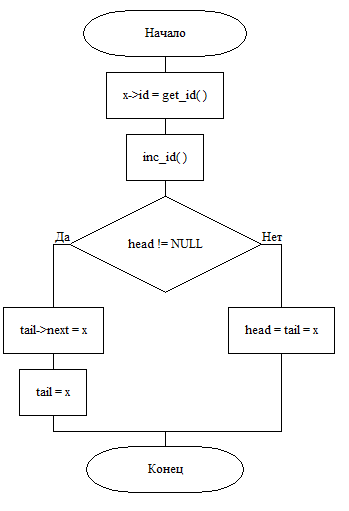


Рис.7 Схема алгоритма функции add\_ell(Node \*x)

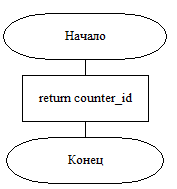


Рис.8 Схема алгоритма функции get\_id()

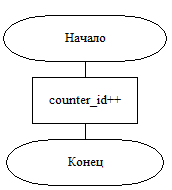


Рис.9 Схема алгоритма функции inc\_id()

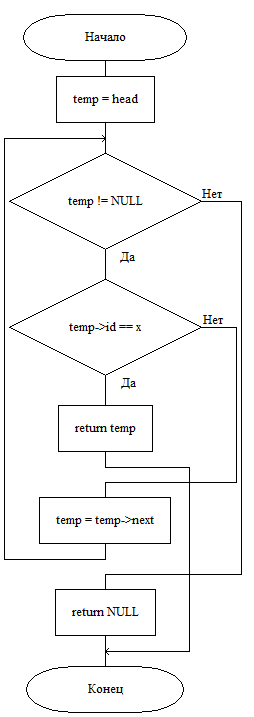


Рис.10 Схема алгоритма функции find\_ell(int x)

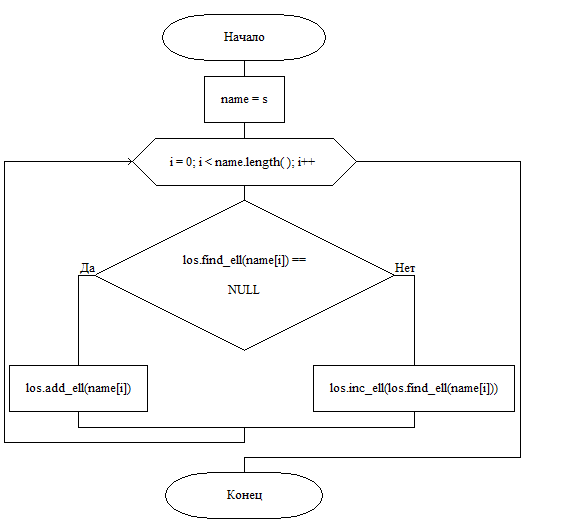


Рис.11 Схема алгоритма функции set\_string(string s)

Рис.12 Схема алгоритма функции make\_tree()

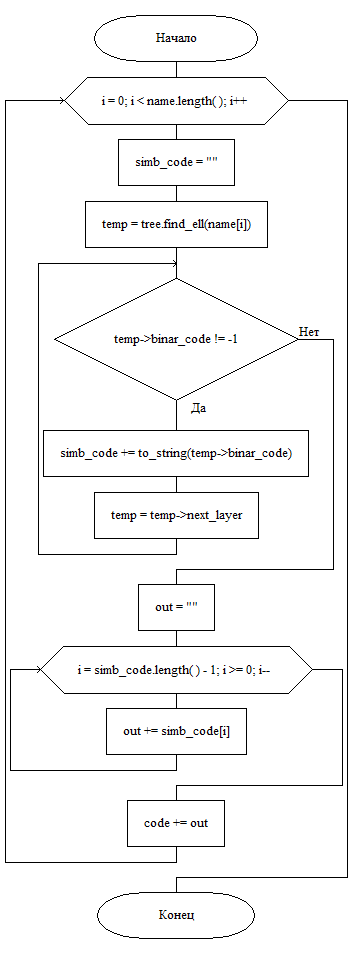
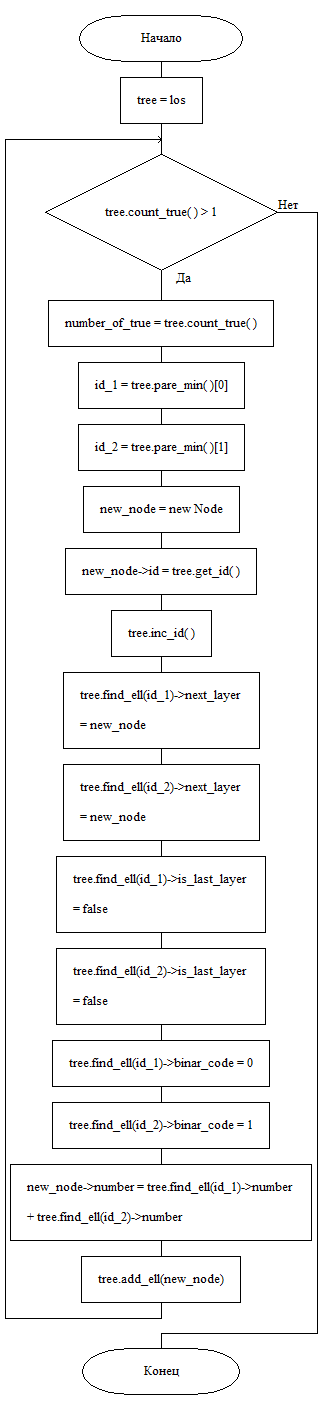


Рис.13 Схема алгоритма функции compress()

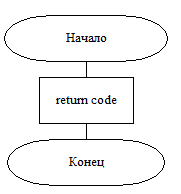


Рис.14 Схема алгоритма функции get\_code()

1. **Реализация алгоритма**

**Текст исходного кода программы**

**main.cpp**

#include <iostream>  
#include "Huffman.h"  
#include <string>  
  
  
int main() {  
  
 int choice = 1;  
 while (choice) {  
 Huffman a;  
 cout << "input string: " << endl;  
 string inp\_str = "";  
 getline(cin, inp\_str);  
 cin.clear();  
 a.set\_string(inp\_str);  
 a.make\_tree();  
 a.compress();  
 cout << a.get\_code() << endl;  
 cout << "1 - new string" << endl;  
 cout << "0 - exit" << endl;  
 cin >> choice;  
 cin.ignore();  
  
  
 }  
 cout << "Programm finished" << endl;  
  
  
}

**Los.h**

#ifndef PR\_10\_LOS\_H  
#define PR\_10\_LOS\_H  
  
#include <iostream>  
  
struct Node {  
 Node \*next = NULL;  
 int id = NULL;  
 char simb = '\_';  
 int number = 1;  
 Node \*next\_layer = NULL;  
 int binar\_code = -1;  
 bool is\_last\_layer = true;  
  
};  
  
class Los {  
private:  
 Node \*head, \*tail;  
 int counter\_id = 0;  
public:  
  
 Los() : head(NULL), tail(NULL) {};  
  
 void add\_ell(char x);  
  
 void add\_ell(Node \*x);  
  
 Node \*find\_ell(char x);  
  
 Node \*find\_ell(int x);  
  
 void inc\_ell(Node \*x);  
  
 int count\_true();  
  
 int \*pare\_min();  
  
 int get\_id();  
  
 void inc\_id();  
  
  
};  
  
  
#endif

**Los.cpp**

#include "Los.h"  
#include <iostream>  
  
  
void Los::add\_ell(char x) {  
 Node \*temp = new Node;  
 temp->simb = x;  
 temp->next = NULL;  
 temp->id = get\_id();  
 inc\_id();  
  
 if (head != NULL) {  
 tail->next = temp;  
 tail = temp;  
 } else head = tail = temp;  
};  
  
Node \*Los::find\_ell(char x) {  
 Node \*temp = head;  
 while (temp != NULL) {  
 if (temp->simb == x) {  
 return temp;  
 }  
 temp = temp->next;  
 }  
 return NULL;  
}  
  
void Los::inc\_ell(Node \*x) {  
 x->number++;  
}  
  
int Los::count\_true() {  
 int count = 0;  
 Node \*temp = head;  
 while (temp != NULL) {  
 if (temp->is\_last\_layer) count++;  
 temp = temp->next;  
  
 }  
 return count;  
}  
  
  
int \*Los::pare\_min() {  
 int m[2];  
 Node \*temp = head;  
 static Node temp\_min[2];  
 unsigned long big\_number = pow(2, 31) - 1;  
 temp\_min[0].number = big\_number;  
 temp\_min[1].number = big\_number;  
  
 while (temp != NULL) {  
 if (temp->is\_last\_layer) {  
 if (temp->number < temp\_min[0].number) {  
 temp\_min[1] = temp\_min[0];  
 temp\_min[0] = \*temp;  
 } else if (temp->number < temp\_min[1].number) { temp\_min[1] = \*temp; }  
  
 }  
 temp = temp->next;  
 }  
 m[0] = temp\_min[0].id;  
 m[1] = temp\_min[1].id;  
  
  
 return m;  
  
  
}  
  
void Los::add\_ell(Node \*x) {  
 x->id = get\_id();  
 inc\_id();  
 if (head != NULL) {  
 tail->next = x;  
 tail = x;  
 } else head = tail = x;  
}  
  
int Los::get\_id() {  
 return counter\_id;  
}  
  
void Los::inc\_id() {  
 counter\_id++;  
}  
  
Node \*Los::find\_ell(int x) {  
 Node \*temp = head;  
 while (temp != NULL) {  
 if (temp->id == x) {  
 return temp;  
 }  
 temp = temp->next;  
 }  
 return NULL;  
}

**Haffman.h**

#ifndef PR\_10\_HUFFMAN\_H  
#define PR\_10\_HUFFMAN\_H  
  
#include <iostream>  
#include <list>  
#include "Los.h"  
  
using namespace std;  
  
class Huffman {  
private:  
 Los los;  
 Los tree;  
 string name = "A";  
 string code = "";  
  
public:  
 void set\_string(string s);  
  
 void make\_tree();  
  
 void compress();  
  
 string get\_code();  
  
  
};  
  
  
#endif

**Haffman.cpp**

#include "Huffman.h"  
  
#include <string>  
  
void Huffman::set\_string(string s) {  
 name = s;  
 for (int i = 0; i < name.length(); i++) {  
 if (los.find\_ell(name[i]) == NULL) {  
 los.add\_ell(name[i]);  
 } else {  
 los.inc\_ell(los.find\_ell(name[i]));  
 }  
 }  
  
}  
  
void Huffman::make\_tree() {  
 tree = los;  
 while (tree.count\_true() > 1) {  
 int number\_of\_true = tree.count\_true();  
  
 int id\_1 = tree.pare\_min()[0];  
 int id\_2 = tree.pare\_min()[1];  
 Node \*new\_node = new Node;  
 new\_node->id = tree.get\_id();  
 tree.inc\_id();  
 tree.find\_ell(id\_1)->next\_layer = new\_node;  
 tree.find\_ell(id\_2)->next\_layer = new\_node;  
 tree.find\_ell(id\_1)->is\_last\_layer = false;  
 tree.find\_ell(id\_2)->is\_last\_layer = false;  
 tree.find\_ell(id\_1)->binar\_code = 0;  
 tree.find\_ell(id\_2)->binar\_code = 1;  
 new\_node->number = tree.find\_ell(id\_1)->number + tree.find\_ell(id\_2)->number;  
 tree.add\_ell(new\_node);  
  
 }  
}  
  
  
void Huffman::compress() {  
  
 for (int i = 0; i < name.length(); i++) {  
 string simb\_code = "";  
 Node \*temp = tree.find\_ell(name[i]);  
 while (temp->binar\_code != -1) {  
 simb\_code += to\_string(temp->binar\_code);  
 temp = temp->next\_layer;  
 }  
 string out = "";  
 for (int i = simb\_code.length() - 1; i >= 0; i--)  
 out += simb\_code[i];  
 code += out;  
 }  
  
  
}  
  
string Huffman::get\_code() {  
 return code;  
}

1. **Тестирование программы**

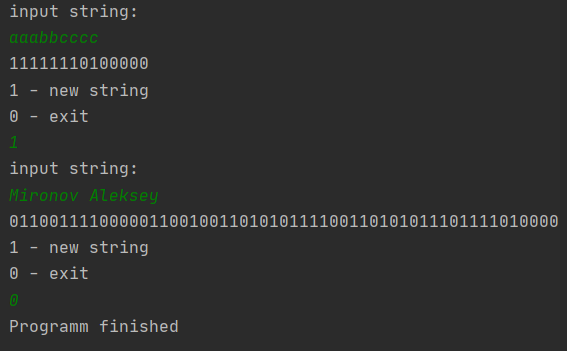


Рис.15 Скриншот шифрования двух строк (1 – произвольная, 2 - Фамилия, имя)

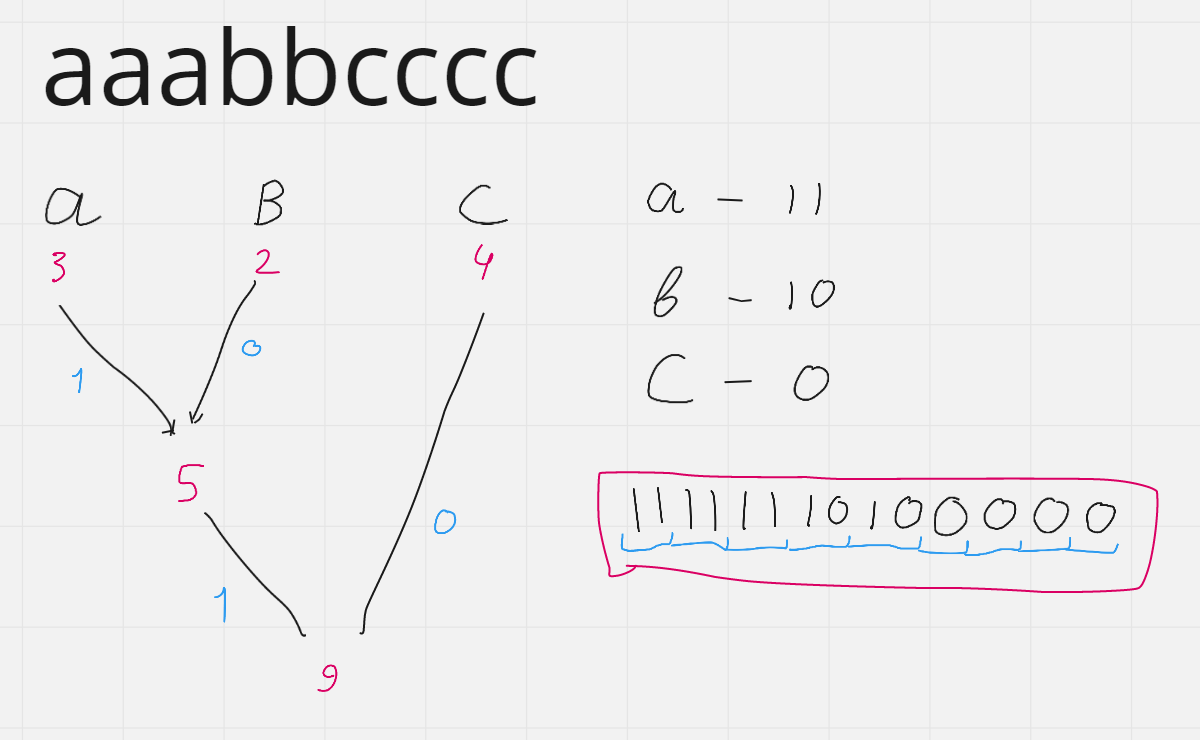
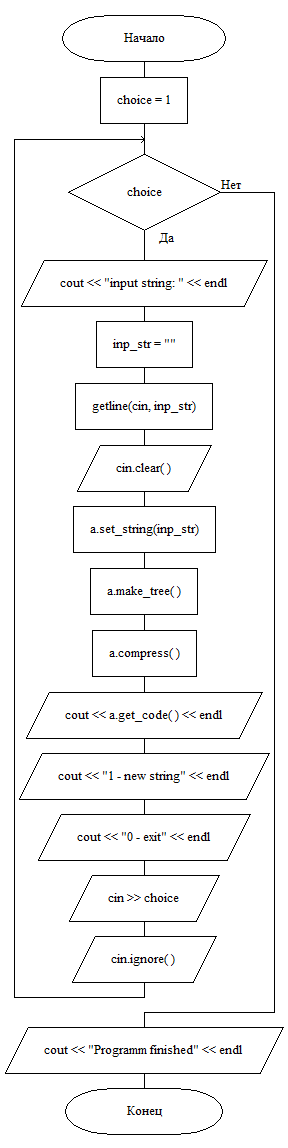


Рис.16 Скриншот ручной проверки алгоритма на строке из рис.1

1. **Выводы**
2. В ходе данной работы были изучены принципы работы сжатия данных
3. На практике изучен и реализован алгоритм Хаффмана.

**Список используемых информационных источников**

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования С++, интернет-ресурс: <https://en.cppreference.com/w/> (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: <https://www.jetbrains.com/clion/learning-center/> (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: <http://docs.cntd.ru/document/gost-19-701-90-espd> (Дата обращения – 22.04.2020).



1. Cреда для создания схем алгоритмов, интернет-ресурс: <https://www.draw.io> (Дата обращения – 22.04.2020).