МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»
Кафедра вычислительных систем

КУРСОВАЯ РАБОТА
по дисциплине «Технологии разработки программного обеспечения»
на тему
«Программа символы: проверка сбалансированности скобок в программе на C++»

Выполнил: Ст.гр.ИП-211
Сократов М.В.
Проверил:
Ст. преподаватель Токмашева Е. И

Новосибирск, 2023

Содержание

Введение и постановка задачи

Нами выбрана тема “Программа символы: проверка сбалансированности скобок в программе на C++”, потому что мне интересен процесс создания приложения, и я хотел бы лучше разобраться в технологии создания функций. В работе будут реализованы следующие функции:

1. Функция «меню».
2. Функция записи из файла расширения .cpp в файл расширения .txt.
3. Функция нахождения скобок в файле
4. Функция обнаружения не сбалансированных скобок
5. Вывод информации о правильности кода или о том, в каких строках не сбалансированные скобки

Для этого нужно решить следующие задачи:

1. Изучить язык.
2. Реализовать вышеперечисленные функции.

После реализации можно провести тесты приложения.

Техническое задание

Реализовать программу.

Функционал: Ввод пользователем полного пути проверяемого файла или же ввод текста проверяемой программы самостоятельно.

Клиентская часть. Для всего использовать язык программирования C++.

1. Выбор пользователем одной из 3 вариантов: ввод полного пути к проверяемому файлу, ввод текста проверяемой программы самостоятельно, выход из программы.

    1.1 Если был выбран первый вариант, то пользователь вводит в консоль полный путь к файлу.

    1.2 Если был выбран второй вариант, то пользователь вводит в консоль код программы, которую он хочет проверить.

    1.3 Если был выбран третий вариант, то программа завершает работу.

    1.4 Если были введены некорректные данные, то программа выдаст ошибку и попросит ввести данные повторно.

2. Запись из файла расширения .cpp в файл расширения .txt «если был выбран 1 вариант».

3. Считать файл .txt и найти в нём все скобки.

4. Повторное считывание файла и проверка на корректные скобки.

    4.1 Вывод о сбалансированности всех скобок.

    4.2 Вывод строк, в которых есть несбалансированные скобки.

5. Написать тесты.

Проверяемая программа не рассматривается с точки зрения правильности кода.

Описание выполненного проекта

```
int main()
{
    cout << "Privet hozain, now i work" << endl;
    window1();
    char choicep = choice();
    string sf = window(choicep);
    if (sf == "-1") {
        cout << "Bye Bye" << endl;
        return 0;
    } else if (sf != "0") {
        wrote(sf);
    }
    int countmemory = cheking(fille);
    cout << "Quantity '{}[]': " << countmemory << endl;
    searching(countmemory);
    remove("file.txt");
}
```

В данном фрагменте кода описана заглавная функция main, в которой происходит вызов всех вторичных функций. Сначала идет вызов функции “window”, для вывода меню. Затем в зависимости от возвращаемого значения идет или закрытие программы, или запись из файла .cpp с заданным пользователем адресом в файл file.txt. Далее в переменную countmemory записывается возвращаемое значение из функции cheking и в конце происходит вызов функции searching с последующим удаление вспомогательного файла «file.txt».

```
void window1()
{
    cout << "\nМеню :";
    cout << "\nВведите 1 для ввода полного адреса файла ";
    cout << "\nВведите 2 для ввода кода в консоль ";
    cout << "\nВведите 0 для выхода из программы\n";
}
char choice()
{
    char ch;
    cin >> ch;
    if (ch != '0' and ch != '1' and ch != '2') {
        cout << "\nВвод неверных данных, повторите попытку снова ";
        choice();
    } else {
        return ch;
    }
}
```

```
string window(char pok)
{
    string st;
    string str;
    ofstream file("file.txt");
    if (pok == '1') {
        cout << "\nВведите полный адрес файла(не забутьте заменить 1'\\' "
                "на 2'\\\\' и в конце нажать Enter -> ctlr+x -> Enter для "
                "Windows или Enter -> ctlr+d -> Enter) : ";
        while (getline(cin, str)) {
            st = st + str;
        }
        cout << st;
        file.close();
        return st;
    }
    if (pok == '2') {
        cout << "\nВведите код в консоль(для окончания ввода нажмите Enter "
                "-> ctlr+x -> Enter для Windows или Enter -> ctlr+d -> "
                "Enter): \n";
        while (getline(cin, st)) {
            file << st << endl;
        }
        file.close();
        return "0";
    }
    if (pok == '0') {
        return "-1";
    }
}
```

В данной части кода описывается все, что связанной с меню и окном ввода стартовых данных. Функция window отвечает за вывод меню на экран. В функции choic пользователь вводит цифру, которая отвечает за выбор пункта из меню и эти данные записываются в переменную ch типа char которая потом передается в функцию window. В этой функции, в зависимости от переменной ch, выполняется запись в переменную st полного пути до проверяемого файла (если пользователь ввел “1”) или выполняется запись во вспомогательный файл file.txt данных, которые пользователь ввел с клавиатуры в консоль (если пользователь ввел “2”) или выход из программы (если пользователь ввел “0”) или просьба о повторном вводе данных при других значениях ch.

```
int wrote(string str)
{
    std::string line;
    std::ofstream out;
    std::ifstream in(str);
    out.open("file.txt");
    if (in.is_open()) {
        while (std::getline(in, line)) {
            out << line << std::endl;
        }
    }
    out.close();
    std::cout << "File C++ is writting in file.txt" << std::endl;
    in.close();
    return 0;
}
```

В данной части программы описана функция “wrote” отвечающая за запись из файла .cpp в file.txt, в качестве передаваемого параметра идет адрес файла .cpp записанный в переменную str. Переменная line типа string отвечает за перезапись текста, переменная out отвечает за работу с файлом “file.txt”, переменная in отвечает за работу с файлом .cpp. Если файл открылся, то происходит считывание текста из in в строку line, а потом запись line в out. Далее оба файла закрываются.

```
char* readFile(char files[])
{
    char ch;
    FILE* pf;
    pf = fopen(files, "r");
    int count = 0;
    while ((ch = getc(pf) != EOF)) {
        count++;
    }

    rewind(pf);
    char* chh = new char[count];
    int i = 0;
    while ((ch = getc(pf) != EOF)) {
        chh[i] = ch;
        i++;
    }
    fclose(pf);
    return chh;
}
```

```
char* readFile(char files[])
{
    char ch;
    FILE* pf;
    pf = fopen(files, "r");
    int count = 0;
    while ((ch = getc(pf) != EOF)) {
        count++;
    }

    rewind(pf);
    char* chh = new char[count];
    int i = 0;
    while ((ch = getc(pf) != EOF)) {
        chh[i] = ch;
        i++;
    }
    fclose(pf);
    return chh;
}
```

```
int numbersumbols1(char* chh, int cols)
{
    int count = 0;
    int ch;
    int ct = 0;
    int ct1 = 0;
    for (int i = 0; i < cols; i++) {
        ch = chh[i];
        if (ch == char(0x22)) {
            if (ct1 == 0)
                ct1++;
            else {
                ct1 = 0;
            }
        }
        if (ch == char(0x27)) {
            if (ct == 0)
                ct++;
            else {
                ct = 0;
            }
        }
        // Searching for brackets and entering them into the Skobki array
        if (ch == '{' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '}' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '[' and ct == 0 and ct1 == 0) {
            count++;
        }
```

```
        if (ch == ']' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '(' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == ')' and ct == 0 and ct1 == 0) {
            count++;
        }
    }
    return count;
}
```

```
char* counting(char* arr, int count)
{
    for (int i = 0; i < count - 1; i++) {
        for (int j = i; j < count; j++) {
            if (arr[i] != '0') {
                if (arr[i] == '{' and arr[j] == '}') {
                    arr[i] = '0';
                    arr[j] = '0';
                    break;
                }
                if (arr[i] == '[' and arr[j] == ']') {
                    arr[i] = '0';
                    arr[j] = '0';
                    break;
                }
                if (arr[i] == '(' and arr[j] == ')') {
                    arr[i] = '0';
                    arr[j] = '0';
                    break;
                }
            }
        }
    }
    return arr;
}
```

```
unsigned char* promo(char* arr, int* ind, int count, int& chetTrue)
{
    unsigned char* answer = new unsigned char[count];
    for (int i = 0; i < count; i++) {
        if (arr[i] != '0') {
            chetTrue++;
            answer[chetTrue] = ind[i];
        }
    }
    return answer;
}
```

```
int searching(int n)
{
    char filess[] = "file.txt";
    ifstream file(filess); // opening a file

    if (!file.is_open()) {
        cerr << "file can't be open" << endl;
        return 1;
    }
    char ch;
    //  "{"
    int chetstr = 0;
    int chet = 0;
    int ct = 0;
    int ct1 = 0;
    // counting brackets not contained between the characters " and '

    string line;
    char* Skobki = new char[n];
    int* indexi = new int[n];
    while (file.get(ch)) {
        // Checking for the closeness of quotation marks
        if (ch == char(0x22)) {
            if (ct1 == 0)
                ct1++;
            else {
                ct1 = 0;
            }
        }
        if (ch == char(0x27)) {
            if (ct == 0)
                ct++;
            }
        }
        if (ch == '\n') {
            ++chetstr;
        }
        if (ch == '{' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '{';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == '}' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '}';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == '[' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '[';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == ']' and ct == 0 and ct1 == 0) {
            Skobki[chet] = ']';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == '(' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '(';
            indexi[chet] = chetstr;
            chet++;
        }
```

```
        if (ch == ')' and ct == 0 and ct1 == 0) {
            Skobki[chet] = ')';
            indexi[chet] = chetstr;
            chet++;
        }
    }
    // Search for paired brackets and their pairwise removal from the Skobki
    // array
    for (int i = 0; i < n - 1; i++) {
        for (int j = i; j < n; j++) {
            if (Skobki[i] != '0') {
                if (Skobki[i] == '{' and Skobki[j] == '}') {
                    Skobki[i] = '0';
                    Skobki[j] = '0';
                    break;
                }
                if (Skobki[i] == '[' and Skobki[j] == ']') {
                    Skobki[i] = '0';
                    Skobki[j] = '0';
                    break;
                }
                if (Skobki[i] == '(' and Skobki[j] == ')') {
                    Skobki[i] = '0';
                    Skobki[j] = '0';
                    break;
                }
            }
        }
    }
```

```
    int chetTrue = 0;
    for (int i = 0; i < n; i++) {
        if (Skobki[i] != '0') {
            chetTrue++;
            if (chetTrue == 1) {
                cout << "Your code is bad" << endl;
            }
            cout << "Bad line:" << indexi[i] << endl;
        }
    }
    if (chetTrue == 0) {
        cout << "Your code is good" << endl;
    }
    file.close();
}
```

В данной части кода описаны функции по нахождению скобок в файле и с последующей проверкой их на правильное местоположение. Основных функций 2: cheking и searching. В функции cheking идет проверка на открытие файла и в последствии подсчет скобок в файле за счет посимвольного обхода по файлу и сравнению каждого символа со всеми видами скобок. В функции searching идет создание массивов водном из которых будут хранится сами скобки, а в другом их адреса. Далее идет зануление всех корректных скобок и, если имеются некорректные, то на экран выводится в какой строке они находятся. Все остальные функции служат для тестов.

Тесты:

Тесты делятся на 4 вида и затронули правильность работы алгоритма:

```
CTEST(searching, test1)
{
    char mas[] = "[]]{}}()9]";
    int exp = numbersumbols1(mas, 11);
    int real = 9;
    ASSERT_EQUAL(exp, real);
}

CTEST(searching, test2)
{
    char mas[] = "597gjdie8473yrhbwi4i6nvzxc3215";
    int exp = numbersumbols1(mas, 31);
    int real = 0;
    ASSERT_EQUAL(exp, real);
}
CTEST(searching, test3)
{
    char mas[] = "'{' []";
    int exp = numbersumbols1(mas, 7);
    int real = 2;
    ASSERT_EQUAL(exp, real);
}
```

В первом типе идет проверка алгоритма на нахождение количества скобок в файле.

```
CTEST(searching, test4)
{
    char mas[] = "{}";
    char* exp = writtingSupportList(mas, 3);
    char real[] = "{}";
    ASSERT_STR(exp, real);
}
CTEST(searching, test5)
{
    char mas[] = "'{'}'['";
    char* exp = writtingSupportList(mas, 8);
    char real[] = "}";
    ASSERT_STR(exp, real);
}
CTEST(searching, test6)
{
    char mas[] = "{[]sry[][]4hgd(){(6))}({)bc}()}jkl[][]}";
    char* exp = writtingSupportList(mas, 40);
    char real[] = "{[][][](){())}({)}()}[][]}";
    ASSERT_STR(exp, real);
}
CTEST(searching, test7)
{
    char mas[] = "";
    char* exp = writtingSupportList(mas, 1);
    char real[] = "";
    ASSERT_STR(exp, real);
}
```

Во втором типе происходит переписывание скобок во вспомогательный массив без лишних символов.

```
CTEST(searching, test8)
{
    char mas[] = "{}]";
    char* exp = counting(mas, 4);
    char real[] = "00]";
    ASSERT_STR(exp, real);
}
CTEST(searching, test9)
{
    char mas[] = "";
    char* exp = counting(mas, 1);
    char real[] = "";
    ASSERT_STR(exp, real);
}
CTEST(searching, test10)
{
    char mas[] = "{}[]({[)}]";
    char* exp = counting(mas, 11);
    char real[] = "0000000000";
    ASSERT_STR(exp, real);
}
CTEST(searching, test11)
{
    char mas[] = "{qw({}][{)])8){[([]0-}]";
    char* exp = counting(mas, 24);
    char real[] = "0qw000]0{00)8){0(000-00";
    ASSERT_STR(exp, real);
}
```

В третьем типе идет проверка на зануление всех корректных скобок.

```
CTEST(searching, test12)
{
    char mas[] = "000";
    int in[] = {3, 5, 8};
    int expsize = 0;
    unsigned char* exp = promo(mas, in, 4, expsize);
    unsigned char* real = new unsigned char[1];
    int realsize = 1;
    ASSERT_DATA(exp, expsize, real, realsize);
}

CTEST(searching, test13)
{
    char mas[] = "0000001000";
    int in[] = {3, 5, 8, 34, 67, 78, 98, 123, 565, 786};
    int expsize = 0;
    unsigned char* exp = promo(mas, in, 10, expsize);
    unsigned char* real = new unsigned char[10];
    int realsize = 1;
    ASSERT_DATA(exp, expsize, real, realsize);
}
```

В четвертом типе идет сравнение 2 массивов и замена не 0 на нужный индекс строки.

Программа проходит проверку на все тесты.

Личный вклад в проект:

Написал функцию “window”, частично участвовал в написании остальных функций. Работа с документацией.

Приложения. Текст программы.

Consol.cpp:

```
#include "../app_lib/String.h"
#include <fstream>
#include <iostream>


using namespace std;
void window1()
{
   cout << "\nМеню :";
   cout << "\nВведите 1 для ввода полного адреса файла ";
   cout << "\nВведите 2 для ввода кода в консоль ";
   cout << "\nВведите 0 для выхода из программы\n";
}
char choice()
{
   char ch;
   cin >> ch;
   if (ch != '0' and ch != '1' and ch != '2') {
      cout << "\nВвод неверных данных, повторите попытку снова ";
      choice();
   } else {
      return ch;
   }
}
string window(char pok)
```

```
{
    string st;
    string str;
    ofstream file("file.txt");
    if (pok == '1') {
        cout << "\nВведите полный адрес файла(не забутьте заменить 1'\\' "
                "на 2'\\\\' и в конце нажать Enter -> ctlr+x -> Enter для "
                "Windows или Enter -> ctlr+d -> Enter) : ";
        while (getline(cin, str)) {
            st = st + str;
        }
        cout << st;
        file.close();
        return st;
    }
    if (pok == '2') {
        cout << "\nВведите код в консоль(для окончания ввода нажмите Enter "
                "-> ctlr+x -> Enter для Windows или Enter -> ctlr+d -> "
                "Enter): \n";
        while (getline(cin, st)) {
            file << st << endl;
        }
        file.close();
        return "0";
    }
    if (pok == '0') {
        return "-1";
    }
}
```

main.cpp:

```
#include "../app_lib/String.h"
#include "../app_lib/functions.h"
#include <fstream>
#include <iostream>
using namespace std;
char fille[] = "file.txt";
std::string window(char pok);
int wrote(string str);
int main()
{
   cout << "Privet hozain, now i work" << endl;
   window1();
   char choicep = choice();
   string sf = window(choicep);
   if (sf == "-1") {
      cout << "Bye Bye" << endl;
      return 0;
   } else if (sf != "0") {
      wrote(sf);
   }
   int countmemory = cheking(fille);
   cout << "Quantity '{}[]': " << countmemory << endl;
   searching(countmemory);
   remove("file.txt");
}
```

Search.cpp:

```
#include "../app_lib/String.h"
#include <fstream>
#include <iostream>
using namespace std;

char* readFile(char files[])
{
   char ch;
   FILE* pf;
   pf = fopen(files, "r");
   int count = 0;
   while ((ch = getc(pf) != EOF)) {
      count++;
   }

   rewind(pf);
   char* chh = new char[count];
   int i = 0;
   while ((ch = getc(pf) != EOF)) {
      chh[i] = ch;
      i++;
   }
   fclose(pf);
   return chh;
```

```
}
/*This is a special function for test coverage based on the main algorithm of
 * the "checking" function. It is here that edits are made first and, with
 * successful tests, the algorithm in the “checking” function is changed.
 */
int numbersumbols1(char* chh, int cols)
{
    int count = 0;
    int ch;
    int ct = 0;
    int ct1 = 0;
    for (int i = 0; i < cols; i++) {
        ch = chh[i];
        if (ch == char(0x22)) {
            if (ct1 == 0)
                ct1++;
            else {
                ct1 = 0;
            }
        }
        if (ch == char(0x27)) {
            if (ct == 0)
                ct++;
            else {
                ct = 0;
            }
        }
        // Searching for brackets and entering them into the Skobki array
        if (ch == '{' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '}' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '[' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == ']' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '(' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == ')' and ct == 0 and ct1 == 0) {
            count++;
        }
    }
    return count;
}

int cheking(char files[])
{
```

```
    ifstream file(files); // opening a file

    if (!file.is_open()) {
        cerr << "file can't be open" << endl;
        return 1;
    }

    int count = 0;
    char ch;
    //  "{"
    int ct = 0;
    int ct1 = 0;
    // counting brackets not contained between the characters " and '
    while (file.get(ch)) {
        // Checking for the closeness of quotation marks
        if (ch == char(0x22)) {
            if (ct1 == 0)
                ct1++;
            else {
                ct1 = 0;
            }
        }
        if (ch == char(0x27)) {
            if (ct == 0)
                ct++;
            else {
                ct = 0;
            }
        }
        // Searching for brackets and entering them into the Skobki array
        if (ch == '{' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '}' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '[' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == ']' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == '(' and ct == 0 and ct1 == 0) {
            count++;
        }
        if (ch == ')' and ct == 0 and ct1 == 0) {
            count++;
        }
    }
    file.close();
    return count;
}
```

```
/*This is a special function for test coverage based on the main algorithm of
 * the "searching" function. It is here that edits are made first and, with
 * successful tests, the algorithm in the “searching” function is changed.
 */
char* writtingSupportList(char* arr, int count)
{
    char ch;
    int chetstr = 0;
    int chet = 0;
    int ct = 0;
    int ct1 = 0;
    char* Skobki = new char[count];
    // counting brackets not contained between the characters " and '
    for (int i = 0; i < count; i++) {
        ch = arr[i];
        // Checking for the closeness of quotation marks
        if (ch == char(0x22)) {
            if (ct1 == 0)
                ct1++;
            else {
                ct1 = 0;
            }
        }
        if (ch == char(0x27)) {
            if (ct == 0)
                ct++;
            else {
                ct = 0;
            }
        }
        if (ch == '\n') {
            ++chetstr;
        }
        if (ch == '{' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '{';
            chet++;
        }
        if (ch == '}' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '}';
            chet++;
        }
        if (ch == '[' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '[';
            chet++;
        }
        if (ch == ']' and ct == 0 and ct1 == 0) {
            Skobki[chet] = ']';
            chet++;
        }
        if (ch == '(' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '(';
            chet++;
```

```
        }
        if (ch == ')' and ct == 0 and ct1 == 0) {
            Skobki[chet] = ')';
            chet++;
        }
    }
    return Skobki;
}
/*This is a special function for test coverage based on the main algorithm of
 * the "searching" function. It is here that edits are made first and, with
 * successful tests, the algorithm in the “searching” function is changed.
 */
char* counting(char* arr, int count)
{
    for (int i = 0; i < count - 1; i++) {
        for (int j = i; j < count; j++) {
            if (arr[i] != '0') {
                if (arr[i] == '{' and arr[j] == '}') {
                    arr[i] = '0';
                    arr[j] = '0';
                    break;
                }
                if (arr[i] == '[' and arr[j] == ']') {
                    arr[i] = '0';
                    arr[j] = '0';
                    break;
                }
                if (arr[i] == '(' and arr[j] == ')') {
                    arr[i] = '0';
                    arr[j] = '0';
                    break;
                }
            }
        }
    }
    return arr;
}
/*This is a special function for test coverage based on the main algorithm of
 * the "searching" function. It is here that edits are made first and, with
 * successful tests, the algorithm in the “searching” function is changed.
 */
unsigned char* promo(char* arr, int* ind, int count, int& chetTrue)
{
    unsigned char* answer = new unsigned char[count];
    for (int i = 0; i < count; i++) {
        if (arr[i] != '0') {
            chetTrue++;
            answer[chetTrue] = ind[i];
        }
    }
    return answer;
}
```

```
int searching(int n)
{
    char filess[] = "file.txt";
    ifstream file(filess); // opening a file

    if (!file.is_open()) {
        cerr << "file can't be open" << endl;
        return 1;
    }
    char ch;
    //  "{"
    int chetstr = 0;
    int chet = 0;
    int ct = 0;
    int ct1 = 0;
    // counting brackets not contained between the characters " and '

    string line;
    char* Skobki = new char[n];
    int* indexi = new int[n];
    while (file.get(ch)) {
        // Checking for the closeness of quotation marks
        if (ch == char(0x22)) {
            if (ct1 == 0)
                ct1++;
            else {
                ct1 = 0;
            }
        }
        if (ch == char(0x27)) {
            if (ct == 0)
                ct++;
            else {
                ct = 0;
            }
        }
        if (ch == '\n') {
            ++chetstr;
        }
        if (ch == '{' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '{';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == '}' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '}';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == '[' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '[';
```

```
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == ']' and ct == 0 and ct1 == 0) {
            Skobki[chet] = ']';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == '(' and ct == 0 and ct1 == 0) {
            Skobki[chet] = '(';
            indexi[chet] = chetstr;
            chet++;
        }
        if (ch == ')' and ct == 0 and ct1 == 0) {
            Skobki[chet] = ')';
            indexi[chet] = chetstr;
            chet++;
        }
    }
    // Search for paired brackets and their pairwise removal from the Skobki
    // array
    for (int i = 0; i < n - 1; i++) {
        for (int j = i; j < n; j++) {
            if (Skobki[i] != '0') {
                if (Skobki[i] == '{' and Skobki[j] == '}') {
                    Skobki[i] = '0';
                    Skobki[j] = '0';
                    break;
                }
                if (Skobki[i] == '[' and Skobki[j] == ']') {
                    Skobki[i] = '0';
                    Skobki[j] = '0';
                    break;
                }
                if (Skobki[i] == '(' and Skobki[j] == ')') {
                    Skobki[i] = '0';
                    Skobki[j] = '0';
                    break;
                }
            }
        }
    }
    int chetTrue = 0;
    for (int i = 0; i < n; i++) {
        if (Skobki[i] != '0') {
            chetTrue++;
            if (chetTrue == 1) {
                cout << "Your code is bad" << endl;
            }
            cout << "Bad line:" << indexi[i] << endl;
        }
    }
```

```
    if (chetTrue == 0) {
        cout << "Your code is good" << endl;
    }
    file.close();
}
```

Writing.cpp:

```
#include "../app_lib/String.h"
#include <fstream>
#include <iostream>
// the file should not be modified!!!!!

using namespace std;

int wrote(string str)
{
    std::string line;
    std::ofstream out;
    std::ifstream in(str);
    out.open("file.txt");
    if (in.is_open()) {
        while (std::getline(in, line)) {
            out << line << std::endl;
        }
    }
    out.close();
    std::cout << "File C++ is writting in file.txt" << std::endl;
    in.close();
    return 0;
}
```

Main1.cpp:

```
#define CTEST_MAIN
#include "../src/app_lib/ctest.h"
int cheking(char fill[]);
int main(int argc, const char** argv)
{
    return ctest_main(argc, argv);
}
```

Test.cpp:

```
#include "../src/app_lib/String.h"
#include "../src/app_lib/ctest.h"
#include "../src/app_lib/functions.h"

// testing numbersumbols1
CTEST(searching, test1)
{
    char mas[] = "[]]{}}()9]";
    int exp = numbersumbols1(mas, 11);
    int real = 9;
    ASSERT_EQUAL(exp, real);
```

```
}

CTEST(searching, test2)
{
   char mas[] = "597gjdie8473yrhbwi4i6nvzxc3215";
   int exp = numbersumbols1(mas, 31);
   int real = 0;
   ASSERT_EQUAL(exp, real);
}
CTEST(searching, test3)
{
   char mas[] = "'{' []";
   int exp = numbersumbols1(mas, 7);
   int real = 2;
   ASSERT_EQUAL(exp, real);
}
// testing writtingSupportList - algoritm #1/3 "searching"
CTEST(searching, test4)
{
   char mas[] = "{}";
   char* exp = writtingSupportList(mas, 3);
   char real[] = "{}";
   ASSERT_STR(exp, real);
}
CTEST(searching, test5)
{
   char mas[] = "'{'}'['";
   char* exp = writtingSupportList(mas, 8);
   char real[] = "}";
   ASSERT_STR(exp, real);
}
CTEST(searching, test6)
{
   char mas[] = "{[]sry[][]4hgd(){(6))}({)bc}()}jkl[][]}";
   char* exp = writtingSupportList(mas, 40);
   char real[] = "{[][][](){())}({)}()}[][]}";
   ASSERT_STR(exp, real);
}
CTEST(searching, test7)
{
   char mas[] = "";
   char* exp = writtingSupportList(mas, 1);
   char real[] = "";
   ASSERT_STR(exp, real);
}
// testing counting - algoritm #2/3 "searching"
CTEST(searching, test8)
{
   char mas[] = "{}]";
   char* exp = counting(mas, 4);
   char real[] = "00]";
   ASSERT_STR(exp, real);
```

```
}
CTEST(searching, test9)
{
   char mas[] = "";
   char* exp = counting(mas, 1);
   char real[] = "";
   ASSERT_STR(exp, real);
}
CTEST(searching, test10)
{
   char mas[] = "{}[]({[)}]";
   char* exp = counting(mas, 11);
   char real[] = "0000000000";
   ASSERT_STR(exp, real);
}
CTEST(searching, test11)
{
   char mas[] = "{qw({}][{)])8){[([]0-}]";
   char* exp = counting(mas, 24);
   char real[] = "0qw000]0{00)8){0(000-00";
   ASSERT_STR(exp, real);
}
// testing counting - algoritm #3/3 "searching"
CTEST(searching, test12)
{
   char mas[] = "000";
   int in[] = {3, 5, 8};
   int expsize = 0;
   unsigned char* exp = promo(mas, in, 4, expsize);
   unsigned char* real = new unsigned char[1];
   int realsize = 1;
   ASSERT_DATA(exp, expsize, real, realsize);
}

CTEST(searching, test13)
{
   char mas[] = "0000001000";
   int in[] = {3, 5, 8, 34, 67, 78, 98, 123, 565, 786};
   int expsize = 0;
   unsigned char* exp = promo(mas, in, 10, expsize);
   unsigned char* real = new unsigned char[10];
   int realsize = 1;
   ASSERT_DATA(exp, expsize, real, realsize);
}
```

Makefile:

```
all: main testing

main: src/app/main.o src/app/console.o src/app/search.o src/app/writting.o
	g++ -o src/appp src/app/main.o src/app/console.o src/app/search.o src/app/writting.o
```

```
main.o: src/app/main.cpp
	g++ -Wall -Wextra -c src/app/main.cpp -MMD

console.o: src/app/console.cpp
	g++ -Wall -Wextra -c src/app/console.cpp -MMD

search.o: src/app/search.cpp
	g++ -Wall -Wextra -c src/app/search.cpp -MMD

writting.o: src/app/writting.cpp
	g++ -Wall -Wextra -c src/app/writting.cpp -MMD

testing: tests/main1.o tests/test.o src/app/main.o src/app/console.o src/app/search.o
src/app/writting.o
	g++ -o tests/testing tests/main1.o tests/test.o src/app/console.o src/app/search.o
src/app/writting.o

main1.o: tests/main1.cpp
	g++ -Wall -Wextra -c test/main1.cpp -MMD

test.o: tests/test.cpp
	g++ -Wall -Wextra -c tests/test.cpp -MMD

clean:
	rm src/appp src/app/*.o src/app/*.d -f
	rm tests/appp tests/*.o tests/*.d tests/testing -f

run: testing
	./tests/testing

runap: main
	./src/appp

start:
	make
	make run
	make runap

start1:
	make clean
	make
	make run
	make runap
```