МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Студент гр. 9304 _____ Борисовский В.Ю.

Преподаватель _____ Фиалковский М.С.

Санкт-Петербург
2020

Цель работы.

Изучить рекурсию, написать синтаксический анализатор.

Задание.

Вариант 24

Построить синтаксический анализатор для понятия текст_со_скобками.

текст_со_скобками ::= элемент || элемент текст_со_скобками

элемент ::= A || B || (текст_со_скобками) || [текст_со_скобками]

||{ текст_со_скобками }

Выполнение работы.

Для выполнения задачи были разработаны три побочные функции, по одной для каждого типа скобок и одна основная, в которой происходит анализ переданной строки.

Побочные функции имеют одинаковый функционал за исключением того, что данный функционал применяется к разному типу скобок, следовательно я опишу работу алгоритма для одного типа скобок, например, для «(», для остальных типов скобок алгоритм аналогичен.

Выполнение работы я разбил на 4 части.

1) Сперва я написал функцию *round_brackets(int &index, string &str, string &result).* Функция *round_brackets(int &index, string &str, string &result)* принимает три параметра, первый хранит индекс обрабатываемого в данный момент символа, второй - это обрабатываемая строка, а третий — строка, в которую записывается результат (если, что - то пошло не так, результат записываться перестанет, и можно будет увидеть место ошибки при выводе строки *result).* Мы идем по строке до тех пор, пока не встретим закрывающую скобку такого же типа, либо пока не возникнет какая - либо ошибка. В строке мы можем встретить: символ элемента «A» или «B», символ пробела либо же символ другой открывающей скобки, все остальные символы будут распознаны как неверные. Если встречается символ «A» или

«В» или же пробела выполняется проверка некоторых условий и если все в порядке, символ добавляется в строку *result*, иначе же будет напечатан лог ошибки и функция вернет *false*. Если будет встречена открывающая скобка, то рекурсивно будет вызвана функция для соответствующего типа скобки. Если закрывающая скобка не будет встречена, произойдет ошибка, которая будет отловлена.

2) После этого я написал аналогчиный код для скобок типа «{» и «[» и назвал данные функции соответственно *bool braces (int &index, string &str, string &result)* и *bool square_brackets(int &index, string &str, string &result)* данные функции имеют полностью аналогичный функционал как и функция *round_brackets(int &index, string &str, string &result)* за исключением того, что мы идем по строке пока не встретим скобку закрывающего типа «}» и «]» соответственно, либо пока не словим какую - либо ошибку (невалидный символ, лишний пробел и тд.).

3) Затем я объеденил все три написанные функции для каждого из типа скобок в одну общую функцию, которая могла бы обрабатывать уже абсолютно любую строку. То есть работа этой функции начиналась с 0 индекса строки переданной ей, а не со скобки открывающего типа как в 3 функциях для работы со скобками. Принцип работы данной функции *bracketed_text(stirng &str, string &result)* аналогичен принципу работы функций обрабатывающих выражения начинающиеся с открывающих скобок. Отличие в том, что обработка строки происходит по условию «пока индекс меньше длины строки», а не до того как нам встретится скобка закрывающего типа.

4) Последним этапом было написание функции *main*. Строка для обработки принимается при реализации Command Line Interface, то есть через аргументы командной строки. Такая реализация сделана для удобства тестирования скриптом. Затем для строки вызывается функция *bracketed_text(stirng &str, string &result)* и если она вернет *true*, то будет

выведен текст "Oh, YES, it's bracketed_text\n", а также результирующая строка, которая должна быть равна исходной. В ином же случае сама функция *bracketed_text(stirng &str, string &result)* выведет причину ошибки, а в функции *main* уже будет выведена результирующая строка, в которой можно будет увидеть место, в котором данная ошибка произошла и прервала дальнейшее выполнение функции.

Тестирование.

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции программы и созданию исполняемого файла lab1. Запуск программы производится командой “./lab1” и последующим вводом строки, содержащей логическое выражение. Тестирование производится с помощью скрипта test_skript.py. Запуск скрипта производится командой “python3 test_skript.py” в директории lab1.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	{A B} B [(B)] A {([B A])}	Oh, YES, it's bracketed_text {A B} B [(B)] A {([B A])}	Отработано успешно.
2.	{A B B B [B B A {A}]}	Oh, YES, it's bracketed_text {A B B B [B B A {A}]}	Отработано успешно.
3.	{A{B}}	Error: space expected! {A{	Ошибка: ожидался пробел.
4.	(A B) {B}	Error: odd space! (A	Ошибка: лишний пробел.
5.	{A B]	Error: wrong type of closing brackets! {A B]	Ошибка: неверный тип закрывающей скобки.
6.	[A A B	Error: missing closing brackets! [A A B	Ошибка: не обнаружена закрывающая скобка.
7.	{A B [B] x}	Error: invalid character! {A B [B] x	Ошибка: обнаружен неверный символ.
8.	()	Error: the contents of the brackets must not be empty or end with a space! ()	Ошибка: скобка не может быть пустой.
9.	{A B (B) }	Error: the contents of the brackets must not be empty or end with a space! {A B (B) }	Ошибка: скобка не может заканчиваться пробелом
10.	({{[A B]} B})	Oh, YES, it's bracketed_text ({{[A B]} B})	Отработано успешно.

Выводы.

Изучили рекурсию и применили ее в написании синтаксического анализатора скобок.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  bool round_brackets(int &index, string &str, string &result);
6.  bool braces (int &index, string &str, string &result);
7.  bool square_brackets(int &index, string &str, string &result);
8.  bool bracketed_text(string &str, string &result);
9.
10. bool braces (int &index, string &str, string &result){
11.     bool k;
12.     result += str[index];
13.     index++;
14.     while (str[index] != '}'){
15.         if ((str[index] == 'A' || str[index] == 'B') && (str[index - 1] == '{' ||
    str[index - 1] == ' ')){
16.             result += str[index];
17.         }
18.
19.         else if (str[index] == ' ' && (str[index - 1] == 'A' || str[index - 1] == 'B'
    || str[index - 1] == '}' || str[index - 1] == ']' || str[index - 1] == ')')){
20.             result += str[index];
21.         }
22.
23.         else if (str[index] == '{' && (str[index - 1] == '{' || str[index - 1] == ' '))
    {
```

```
24.             k = braces(index, str, result);
25.             if (!k){
26.                 return false;
27.             }
28.         }
29.
30.         else if (str[index] == '(' && (str[index - 1] == '{' || str[index - 1] == ' ')){
31.             k = round_brackets(index, str, result);
32.             if (!k){
33.                 return false;
34.             }
35.         }
36.
37.         else if (str[index] == '[' && (str[index - 1] == '{' || str[index - 1] == ' ')){
38.             k = square_brackets(index, str, result);
39.             if (!k){
40.                 return false;
41.             }
42.         }
43.
44.         else {
45.             result += str[index];
46.              if (str[index] == 'A' || str[index] == 'B' || str[index] == '{' || str[index]
    == '(' || str[index] == '['){
47.                 cout << "Error: space expected!\n";
48.             }
49.
50.             else if (str[index] == ' '){
51.                 cout << "Error: odd space!\n";
```

```
52.             }
53.
54.             else if (str[index] == ']' || str[index] == ')'){
55.                 cout << "Error: wrong type of closing brackets!\n";
56.             }
57.
58.             else if (index == str.length()){
59.                 cout << "Error: missing closing brackets!\n";
60.             }
61.
62.             else {
63.                 cout << "Error: invalid character!\n";
64.             }
65.
66.             return false;
67.         }
68.
69.         index++;
70.     }
71.
72.     if(str[index - 1] != '{' && str[index - 1] != ' '){
73.         result += str[index];
74.         return true;
75.     } else {
76.         result += str[index];
77.          cout << "Error: the contents of the brackets must not be empty or end
    with a space!\n";
78.         return false;
79.     }
```

```
80.}
81.
82.
83.bool round_brackets(int &index, string &str, string &result){
84.    bool k;
85.    result += str[index];
86.    index++;
87.    while (str[index] != ')'){
88.          if ((str[index] == 'A' || str[index] == 'B') && (str[index - 1] == '(' ||
    str[index - 1] == ' ')){
89.          result += str[index];
90.       }
91.
92.       else if (str[index] == ' ' && (str[index - 1] == 'A' || str[index - 1] == 'B'
    || str[index - 1] == '}' || str[index - 1] == ']' || str[index - 1] == ')')){
93.          result += str[index];
94.       }
95.
96.       else if (str[index] == '(' && (str[index - 1] == '(' || str[index - 1] == ' ')){
97.          k = round_brackets(index, str, result);
98.          if (!k){
99.             return false;
100.                }
101.             }
102.
103.              else if (str[index] == '{' && (str[index - 1] == '(' || str[index - 1]
    == ' ')){
104.                k = braces(index, str, result);
105.                if (!k){
```

```
106.                 return false;
107.             }
108.         }
109.
110.          else if (str[index] == '[' && (str[index - 1] == '(' || str[index - 1]
    == ' ')){
111.             k = square_brackets(index, str, result);
112.             if (!k){
113.                 return false;
114.             }
115.         }
116.
117.         else {
118.             result += str[index];
119.                 if (str[index] == 'A' || str[index] == 'B' || str[index] == '{' ||
    str[index] == '(' || str[index] == '['){
120.                 cout << "Error: space expected!\n";
121.             }
122.
123.             else if (str[index] == ' '){
124.                 cout << "Error: odd space!\n";
125.             }
126.
127.             else if (str[index] == '}' || str[index] == ']'){
128.                 cout << "Error: wrong type of closing brackets!\n";
129.             }
130.
131.             else if (index == str.length()){
132.                 cout << "Error: missing closing brackets!\n";
```

```
133.                 }
134.
135.                 else {
136.                     cout << "Error: invalid character!\n";
137.                 }
138.
139.                 return false;
140.             }
141.
142.             index++;
143.         }
144.
145.         if(str[index - 1] != '(' && str[index - 1] != ' '){
146.             result += str[index];
147.             return true;
148.         } else {
149.             result += str[index];
150.             cout << "Error: the contents of the brackets must not be empty or
    end with a space!\n";
151.             return false;
152.         }
153.     }
154.
155.
156.
157.     bool square_brackets(int &index, string &str, string &result){
158.         bool k;
159.         result += str[index];
160.         index++;
```

```
161.            while (str[index] != ']'){
162.                if ((str[index] == 'A' || str[index] == 'B') && (str[index - 1] == '['
    || str[index - 1] == ' ')){
163.                    result += str[index];
164.                }
165.
166.                else if (str[index] == ' ' && (str[index - 1] == 'A' || str[index - 1]
    == 'B' || str[index - 1] == '}' || str[index - 1] == ']' || str[index - 1] == ')')){
167.                    result += str[index];
168.                }
169.
170.                else if (str[index] == '[' && (str[index - 1] == '[' || str[index - 1]
    == ' ')){
171.                    k = square_brackets(index, str, result);
172.                    if (!k){
173.                        return false;
174.                    }
175.                }
176.
177.                else if (str[index] == '{' && (str[index - 1] == '[' || str[index - 1]
    == ' ')){
178.                    k = braces(index, str, result);
179.                    if (!k){
180.                        return false;
181.                    }
182.                }
183.
184.                else if (str[index] == '(' && (str[index - 1] == '[' || str[index - 1]
    == ' ')){
```

```
185.                 k = round_brackets(index, str, result);
186.                 if (!k){
187.                     return false;
188.                 }
189.             }
190.
191.             else {
192.                 result += str[index];
193.                     if (str[index] == 'A' || str[index] == 'B' || str[index] == '{' ||
    str[index] == '(' || str[index] == '['){
194.                     cout << "Error: space expected!\n";
195.                 }
196.
197.                 else if (str[index] == ' '){
198.                     cout << "Error: odd space!\n";
199.                 }
200.
201.                 else if (str[index] == '}' || str[index] == ')'){
202.                     cout << "Error: wrong type of closing brackets!\n";
203.                 }
204.
205.                 else if (index == str.length()){
206.                     cout << "Error: missing closing brackets!\n";
207.                 }
208.
209.                 else {
210.                     cout << "Error: invalid character!\n";
211.                 }
212.
```

```
213.                 return false;
214.             }
215.
216.             index++;
217.         }
218.
219.
220.
221.         if(str[index - 1] != '[' && str[index - 1] != ' '){
222.             result += str[index];
223.             return true;
224.         } else {
225.             result += str[index];
226.             cout << "Error: the contents of the brackets must not be empty or
    end with a space!\n";
227.             return false;
228.         }
229.     }
230.
231.     bool bracketed_text(string &str, string &result){
232.         int index = 0;
233.         bool k;
234.         if (index == str.length()){
235.             cout << "Error: string must be init";
236.             return false;
237.         }
238.
239.         while (index < str.length()){
240.             if (str [index] == '{' && (str[index - 1] == ' ' || index == 0)){
```

```
241.                 k = braces(index, str, result);
242.                 if (!k){
243.                     return false;
244.                 }
245.             }
246.
247.             else if (str [index] == '(' && (str[index - 1] == ' ' || index == 0)){
248.                 k = round_brackets(index, str, result);
249.                 if (!k){
250.                     return false;
251.                 }
252.             }
253.
254.             else if (str [index] == '[' && (str[index - 1] == ' ' || index == 0)){
255.                 k = square_brackets(index, str, result);
256.                 if (!k){
257.                     return false;
258.                 }
259.             }
260.
261.              else if ((str[index] == 'A' || str[index] == 'B') && (str[index - 1]
    == '[' || str[index - 1] == ' ' || index == 0)){
262.                 result += str[index];
263.             }
264.
265.              else if (str[index] == ' ' && (str[index - 1] == 'A' || str[index - 1]
    == 'B' || str[index - 1] == '}' || str[index - 1] == ']' || str[index - 1] == ')') &&
    (index + 1) != str.length()){
266.                 result += str[index];
```

```
267.             }
268.
269.             else {
270.                 result += str[index];
271.                     if (str[index] == 'A' || str[index] == 'B' || str[index] == '{' ||
    str[index] == '(' || str[index] == '['){
272.                     cout << "Error: space expected!\n";
273.                 }
274.
275.                 else if (str[index] == ' '){
276.                     cout << "Error: odd space!\n";
277.                 }
278.
279.                 else {
280.                     cout << "Error: invalid character!\n";
281.                 }
282.
283.                 return false;
284.
285.             }
286.             index++;
287.         }
288.
289.         return true;
290.     }
291.
292.     int main(int argc, char **argv) {
293.         string result;
294.         string str(argv[1]);
```

```
295.            if (bracketed_text(str, result)){
296.                cout << "Oh, YES, it's bracketed_text\n" << result;
297.            } else {
298.                cout << result;
299.            }
300.            return 0;
301.        }
```