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

ОТЧЁТ

по лабораторной работе №1
по дисциплине «Алгоритмы и струкруры данных»
Тема: Рекурсивные алгоритмы

Студент гр. 9303 _____ Эйсвальд М.И.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Познакомиться с основными понятиями и приемами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций.

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*}

Основные теоретические положения.

Рекурсивным называется объект, содержащий сам себя или определенный с помощью самого себя. То же определение применимо и к рекурсивным алгоритмам: рекурсивный алгоритм описан с помощью себя же. Рекурсивные алгоритмы лучше всего использовать, когда решаемая задача, вычисляемая функция или обрабатываемая структура данных определены с помощью рекурсии.

Если процедура (функция) `P` содержит явное обращение к самой себе, она называется *прямо рекурсивной*. Если `P` содержит обращение к процедуре (функции) `Q`, которая содержит (прямо или косвенно) обращение к `P`, то `P` называется *косвенно рекурсивной*.

Выполнение работы.

Поскольку понятия текста со скобками и элемента текста определены взаимно-рекурсивно, было решено проверять корректность введённых данных с помощью взаимно-рекурсивных функций `int isText(const char** ptr, char expected, int, FILE*)` и `int isElement(const char** ptr, int, FILE*)` соответственно для проверки корректности текста и корректности элемента. Последние два параметра каждой функции нужны лишь для соблюдения дополнительных условий к программе: вывода глубины рекурсии и печати протокола работы программы в файл. Итак, раз-

бирая алгоритм анализа текста, можно считать, что функция `isText` принимает аргументы `const char**` и `char`, а функция `isElement` — `const char**`. Возвращаемые значения обеих функций одинаковы: 0 — неверно, 1 — верно. Алгоритм проверки использует только функционал чистого Си.

Аргументы `isText` представляют собой указатель на C-style строку, с которой идёт работа, и ожидаемый символ — скобку, парную к открытой в данный момент. Функция считает, что обрабатываемая последовательность является текстом, если встречает ожидаемую скобку (если бы во вводе ранее была ошибка, эта функция не была бы вызвана), обнаруживает пустую строку при условии, что скобок не ожидается (случай полностью пустого ввода обработан отдельно) или если удостоверяется, что текст представляет собой элемент, за которым идёт текст (исходя из того, что полностью пустая строка является текстом).

Функция `isElement` также принимает указатель на строку, с которой сейчас работает программа. Если текущий символ в строке — атомарный элемент, то подлинность элемента может быть установлена сразу. Сложнее дело обстоит, если `isElement` обнаруживает скобку. По определению текст не может быть пустым, но если `isText` встречает пустую строчку, она вернёт `true`. Поэтому приходится «заглядывать» в следующий символ (он может быть и нулём-терминатором), чтобы убедиться, что за открывающей скобкой не следует закрывающая. Если проверка пройдена, вызывается функция `isText` для содержимого скобок.

Возможно, нуждается в пояснении тот факт, что функции обработки ввода принимают указатель на строчку, а не саму строчку. Дело в том, что все функции на всех уровнях глубины рекурсии пользуются одной и той же строкой. После отработки более глубинной функции и возврата в вызвавшую её функцию последующая обработка ввода должна проходить с того места, где её завершила предыдущая функция. Итак, хоть функции анализа текста и взаимно-рекурсивные, входные данные они должны обрабатывать последовательно.

Остальной код программы представляет собой реализацию ввода-вывода и к бизнес-логике не относится.

Тестирование.

В листинге ниже представлен вывод программы, включающий в себя введённые в консоль входные данные (тесты) и протокол их обработки согласно требованиям к лабораторной работе.

```
Failed to open file ”input.txt”. Expecting input from console...
Enter a line to analyze. Enter EOF (i.e. Ctrl+D) to exit.
A
Checking line ”A”
.isText() processing ”A”
..isElement() processing ’A’
..isElement() finished
..isText() processing ””
..isText() finished
.isText() finished
OK:
A

AB
Checking line ”AB”
.isText() processing ”AB”
..isElement() processing ’A’
..isElement() finished
..isText() processing ”B”
...isElement() processing ’B’
...isElement() finished
...isText() processing ””
...isText() finished
..isText() finished
.isText() finished
OK:
AB

[A{A}]B
Checking line ”[A{A}]B”
.isText() processing ”[A{A}]B”
..isElement() processing ’[’
...isText() processing ”A{A}]B”
....isElement() processing ’A’
....isElement() finished
....isText() processing ”{A}]B”
.....isElement() processing ’{’
......isText() processing ”A}]B”
.......isElement() processing ’A’
.......isElement() finished
.......isText() processing ”}]B”
.......isText() finished
......isText() finished
.....isElement() finished
.....isText() processing ”]B”
.....isText() finished
....isText() finished
...isText() finished
..isElement() finished
..isText() processing ”B”
...isElement() processing ’B’
...isElement() finished
...isText() processing ””
```

```
...isText() finished
..isText() finished
.isText() finished
OK:
[A{A}]B


Checking line ”“
Error: empty input
A{}
Checking line ”A{}”
.isText() processing ”A{}”
..isElement() processing ’A’
..isElement() finished
..isText() processing ”{}”
...isElement() processing ’{’
...isElement() finished
Error: empty brackets:
..isText() finished
.isText() finished
A{}
  ^
[A)B
Checking line ”[A)B”
.isText() processing ”[A)B”
..isElement() processing ’[’
...isText() processing ”A)B”
....isElement() processing ’A’
....isElement() finished
....isText() processing ”)B”
.....isElement() processing ’)’
.....isElement() finished
Error: extra closing bracket:
....isText() finished
...isText() finished
..isElement() finished
.isText() finished
[A)
  ^
ACCA
Checking line ”ACCA”
.isText() processing ”ACCA”
..isElement() processing ’A’
..isElement() finished
..isText() processing ”CCA”
...isElement() processing ’C’
...isElement() finished
Error: illegal symbol:
..isText() finished
.isText() finished
AC
 ^
(A[
Checking line ”(A[”
.isText() processing ”(A[”
..isElement() processing ’(’
...isText() processing ”A[”
....isElement() processing ’A’
....isElement() finished
....isText() processing ”[”
```

```
.....isElement() processing '['
......isText() processing ""
......isText() finished
Error: unexpected end of input:
.....isElement() finished
....isText() finished
...isText() finished
..isElement() finished
.isText() finished
(A[
   ^
```

Вывод.

Были изучены основные понятия и приёмы рекурсивного программирования, получены навыки создания рекурсивных процедур и функций. Результатом работы стал синтаксический анализатор для определённого в задании понятия *текст-со-скобками*, анализирующий входные данные на предмет соответствия определению текста со скобками с помощью рекурсивного алгоритма. Также было в очередной раз подтверждено, что со строками на чистом Си лучше не работать.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: input.h

```
#ifndef INPUT_H
#define INPUT_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define BUFFER_BLOCK 20
#define INPUT_FILE ”input.txt”
#define OUTPUT_FILE ”output.txt”

int readLine(char** string, const char* breakers, int read_breakers, FILE* stream);

#endif
```

Название файла: input.c

```
nclude ”input.h”

int readLine(char** string, const char* breakers, int read_breakers, FILE* stream){
    int num = 0;
    int size = 0;
    char chr = 0;
    char* tmp;

    *string = calloc(BUFFER_BLOCK, sizeof(char));
    if(!(*string)) return 1;
    size += BUFFER_BLOCK;

    while ((!strchr(breakers, (chr = fgetc(stream)))) && (chr != EOF)){
        if(num + 2 >= size){
            tmp = realloc(*string, (size + BUFFER_BLOCK) * sizeof(char));
            if(!tmp){
               free(*string);
               return 1;
            }
            *string = tmp;
            size += BUFFER_BLOCK;
        }
        (*string)[num] = chr;
        (*string)[num+1] = 0;
        num++;
    }
    if(read_breakers && (chr != EOF)){
       (*string)[num] = chr;
       (*string)[num+1] = 0;
    }
    if (chr == EOF) return -1;
    return 0;
}
```

Название файла: business.h

```
fndef BUSINESS_H
#define BUSINESS_H

#include <string.h>
#include <stdio.h>

#include ”output.h”

#define false 0
#define true 1

int isText(const char** ptr, char expected, int, FILE*);
int isElement(const char** ptr, int, FILE*);

#endif // BUSINESS_H
```

Название файла: business.c

```
#include ”business.h”

int isText(const char** ptr, char expected, int depth, FILE* out){
int result;
depthtab(depth+1, out);
printf(”isText() processing \”%s\”\n”, *ptr);
fprintf(out, ”isText() processing \”%s\”\n”, *ptr);
if(!strlen(*ptr)){
depthtab(depth+1, out);
printf(”isText() finished\n”);
fprintf(out, ”isText() finished\n”);
if(!(expected)) return true;
error(UNEXPECTED_EOL, out);
return false; // unexpected end of input
}
if(**ptr == expected) {
expected = 0;
depthtab(depth+1, out);
printf(”isText() finished\n”);
fprintf(out, ”isText() finished\n”);
return true;
}
if(isElement(ptr, depth+1, out)){
++(*ptr);
result = isText(ptr, expected, depth+1, out);
depthtab(depth+1, out);
                printf(”isText() finished\n”);
                fprintf(out,”isText() finished\n”);
return result;
}
        depthtab(depth+1, out);
        printf(”isText() finished\n”);
        fprintf(out,”isText() finished\n”);
return false;
}


int isElement(const char** ptr, int depth, FILE* out){
int result;
char tmp;
depthtab(depth+1, out);
printf(”isElement() processing \’%c\’\n”, **ptr);
```

```
fprintf(out,”isElement() processing \’%c\’\n”, **ptr);
switch(**ptr){
case(’A’):
case(’B’):
                depthtab(depth+1, out);
                printf(”isElement() finished\n”);
fprintf(out,”isElement() finished\n”);
return true;
case(’(’):
tmp = ’)’;
break;
case(’{’):
tmp = ’}’;
break;
case(’[’):
tmp = ’]’;
break;
case(’]’):
case(’)’):
case(’}’):
                depthtab(depth+1, out);
                printf(”isElement() finished\n”);
                fprintf(out,”isElement() finished\n”);
error(EXTRA_BRACKET, out);
return false; // extra closing bracket
default:
                depthtab(depth+1, out);
                fprintf(out,”isElement() finished\n”);
                printf(”isElement() finished\n”);
error(ILLEGAL_SYMBOL, out);
return false; // inappropriate symbol
}
++(*ptr);
if(**ptr == tmp){
depthtab(depth+1, out);
printf(”isElement() finished\n”);
fprintf(out,”isElement() finished\n”);
error(EMPTY_BRACKETS, out);
return false; // empty brackets
}
result = isText(ptr, tmp, depth+1, out);
depthtab(depth+1, out);
printf(”isElement() finished\n”);
fprintf(out, ”isElement() finished\n”);
        return result;

}
```

Название файла: output.h

```
#ifndef OUTPUT_H
#define OUTPUT_H

#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define UNEXPECTED_EOL 0
#define EXTRA_BRACKET 1
#define ILLEGAL_SYMBOL 2
```

```
#define EMPTY_BRACKETS 3

#define NORMAL ”\033[0m”
#define RED ”\033[0;31m”

void depthtab(int, FILE*);
void error(int, FILE*);

#endif
```

Название файла: output.c

```
#include ”output.h”

void depthtab(int depth, FILE* out){
for(int i = 0; i < depth; ++i){
printf(”.”);
putc(’.’, out);
}
}


void error(int index, FILE* out){
char errors[][100] = {”unexpected end of input”,
 ”extra closing bracket”,
 ”illegal symbol”,
”empty brackets”};
printf(RED”Error: %s:\n”NORMAL, errors[index]);
fprintf(out, ”Error: %s:\n”, errors[index]);
}
```

Название файла: main.c

```
nclude ”input.h”
#include ”business.h”

int main(){
char* string = NULL;
char* ptr;
FILE* f = fopen(INPUT_FILE, ”r”);
FILE* out = fopen(OUTPUT_FILE, ”w”);
if(!out){
puts(”Cannot create output file. Exiting.”);
return 0;
}
char tmp = 0;
int text;
if(!f){
printf(”Failed to open file \”%s\”. Expecting input from console...\n”, INPUT_FILE);
puts(”Enter a line to analyze. Enter EOF (i.e. Ctrl+D) to exit.”);
fprintf(out,”Failed to open file\”%s\”. Expecting input from console...\n”,INPUT_FILE);
fputs(”Enter a line to analyze. Enter EOF (i.e. Ctrl+D) to exit.\n”,out);
        }
int result;
do{
result = readLine(&string, ”\n”, 0, (f ? f : stdin));
         if(result > 0){
                puts(”Failed to read text. Exiting...”);
                fputs(”Failed to read text. Exiting...\n”, out);
```

```
if(string) free(string);
                exit(0);
        }
if(result == EOF){
if(string) free(string);
string = NULL;
break;
}
printf("Checking line \"%s\"\n", string);
fprintf(out,"Checking line \"%s\"\n", string);
if(!strlen(string)){
puts(RED"Error: empty input"NORMAL);
fputs("Error: empty input\n", out);
}
else{
ptr = string;
text = isText((const char**)(&ptr), tmp, 0, out);
if(text){
puts("OK:");
fputs("OK:\n", out);
printf("%s\n\n", string);
fprintf(out, "%s\n\n", string);
}
else{
for(char* p = string; p < ptr; ++p){
putchar(*p);
putc(*p, out);
}
printf(RED);
putchar(*ptr);
if(*ptr) putc(*ptr, out);
printf(NORMAL);
puts("");
putc('\n', out);
for(int i = 0; i < (int)(ptr - string); ++i){
putchar(' ');
putc(' ', out);
}
puts("^");
fputs("^\n", out);
}
}
free(string);
string = NULL;
}while(result == 0);
if(f) fclose(f);
if(out) fclose(out);
return 0;
}
```