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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Структуры данных

Студент гр. 7382 _____ **Глазунов С.А.**

Преподаватель _____ **Фирсов М. А.**

Санкт-Петербург
2018

Описание алгоритма: (вар 13)

В лабораторной работе требуется написать программу: вычислить глубину (число уровней вложения) иерархического списка как максимальное число одновременно открытых левых скобок в сокращённой скобочной записи списка; принять, что глубина пустого списка и глубина атомарного S-выражения равны нулю; например, глубина списка (a (b ( ) c) d) равна двум;

В работе используется язык программировая C++;

исходный код: файл main.cpp и list.cpp

Файл компиляции и запуска программы из файла на тестирование — runtest2.sh:

Компилирует программу лежащую в папке Sourse и запускает её подавая данные на вход программы файлы из папки: tests.

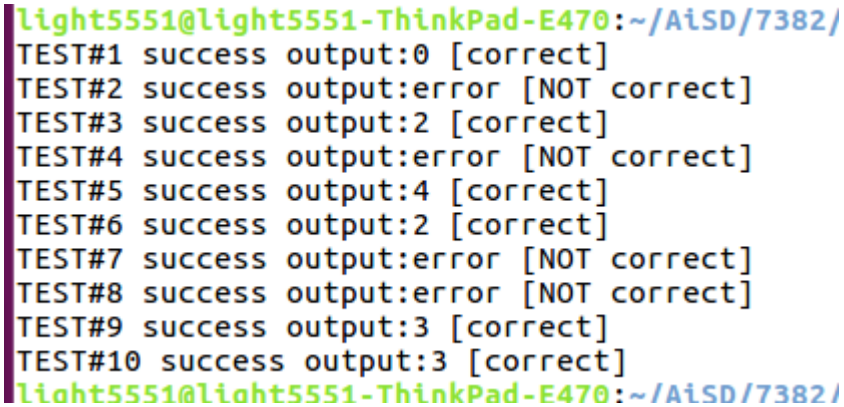
Программа может как и печатать ход работы алгоритма,так и просто выводить решение примера на консоль.

Алгоритм проходит по всей строчке ,занося скобки в список с флагом «is_atom».Если в скобке что то есть,значит true;

После функция count считает сколько раз is_atom был true в списке(реализовано рекурсивно).

Тестирование программы:

Программа выдает ошибку «error»,если на вход подается строка ,где есть неправильная последовательность скобок(то есть случай «)(»);или же выдает ошибку ,когда кол-во закрывающих скобок больше чем открывающих . Тестирование программы осуществляется скриптом написанным на bashe в файле runtest.sh.



Рассмотрим конкретный тест #3 : (a()(ab)). 1-ый символ это открывающая скобка- вставляем ее в список,после проверяем атомарная ли это скобка или нет. Дальше пропускаем все символы кроме скобок , когда встречаются скобки мы складываем или вычитаем чтобы проверять правильность строки, если переменная становиться отрицательной ,значит исход код неверен. Во-второй скобке происходит проверка на атомарность , т.к. после скобки идет сразу закрывающая скобка,то значит она ею и является. После мы встречаем следующую скобку и опять идет проверка на атомарность(в данном случае она не атомарна). В конце программы идет подсчет не атомарных скобок-это и является результатом работы программы.

Описание структур данных

В данной программе структура данных это иерархический список.

Так как список не имеет иерархии,то он превращается в бинарное дерево,так получается из за того,что фигурирует только «(» и «)», и т. к. элементов всего 2,значит оно превращается в бинарное дерево.

Список состоит из 3 структур:

left_bracket- содержит информацию является ли скобка частью атомарной скобки или нет.

right_bracket-содержит указатель на следующую структуру(то есть на следующий возможный элемент)

el_of_list-содержит указатели на левую и правую ветку,если существует левая,то правая становится NULL,и если обратное тоже верно.

Описание функций:

void push(char symbol,bool next_symbol)

Вставляет элемент в список ,перед этим проверив является ли скобочка атомарной или нет при помощи функции check_next_symbol.

Аргументы:symbol — это символ,который мы засовываем в список.

next_symbol-это переменная ,которая показывает атомарная скобка или нет .

void delete_list(struct el_of_list*cur_el)

cur_el-указатель на начало списка.

Вызывается в деструкторе ,чтобы очистить всю динамическую память.

Она рекурсивно доходит до конца списка и начинает удалять элементы (то есть 3 структуры:left_bracket,right_bracket,el_of_list).Код в приложении А.

int count()

Функция считает глубину скобок,а точнее рекурсивно проходит по списку и считает сколько раз встретилась is_atom=true;

bool istextcorrect(std::string text)

Проверяет текст на правильность,используя алгоритм,при котором считается сумма скобок. Если в конце функции сумма равна 0,значит все хорошо-проверка пройдена,а также если в течении работы функции сумма становиться отрицательной- значит неправильный исходный текст,а именно рассматривается случай: ())((сумма конечная равна 0,но на 3 итерации сумма была отрицательной,что как видно неправильно).

bool check_next_symbol(std::string &text,int shift)

Функция проверяет открывающую скобку на атомарность, а именно смотрит какие символы находятся после скобки. Пробелы игнорируются. Если следующий символ равен закрывающей скобке значит скобка атомарна и она не будет включена в глубину скобок.

Выводы.

В ходе выполнения лабораторной работы получены знания по теме «рекурсия» и «структуры данных» ,а также узнали что такое иерархический список и дерево,а также было закрепленны знания синтаксиса языка C++.Код исходной программы лежит в приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STRING>
#INCLUDE <CSTDIO>
#INCLUDE "LIST.CPP"//CLASS,WHERE THERE IS LIST :)
#DEFINE SCRIPT_TEST
//#DEFINE TEST_ALG
BOOL CHECK_NEXT_SYMBOL(STD::STRING &TEXT,INT SHIFT)
        //TRUE- ( A )
{                                                                       //FALSE- ( )
        FOR(INT I=SHIFT;I<TEXT.SIZE()&&((TEXT[I])!=')');I++)
                {
                        IF((TEXT[I])!=' ')
                        RETURN TRUE;
                }
        RETURN FALSE;
}

STD::STRING INPUT(){
        STD::STRING TEXT;
        GETLINE(STD::CIN,TEXT);
        RETURN TEXT;
}

BOOL ISTEXTCORRECT(STD::STRING TEXT)
{
        INT SUM=0;
        FOR(INT I=0;I<TEXT.SIZE();I++)
        {
                IF(SUM<0)                                               //IF SITUATION: ))(( -
TOTAL SUM=0,BUT
                        RETURN FALSE;                                           // IT'S
WRONG ,THAT WHY SUM IS CHECKED
                IF(TEXT[I]=='(')                                        //IN CYCLE
                        SUM++;
                IF(TEXT[I]==')')
```

```
                SUM--;
        }
        IF(SUM==0)
        RETURN TRUE;

RETURN FALSE;
}

INT MAIN(){
MYLIST STK;
BOOL FIRST_CORRECT_BRACKET=FALSE;
#IFNDEF SCRIPT_TEST
STD::COUT<<"HELLO,I AM CHECKER OF BRACKETS.ENTER..."<<STD::ENDL;
#ENDIF
STD::STRING TEXT=INPUT();
INT PAD=0;
IF(FIRST_CORRECT_BRACKET=ISTEXTCORRECT(TEXT))
FOR(INT I=0;I<TEXT.SIZE();I++)
{
        #IFDEF TEST_ALG
        STD::COUT
        <<"DEPTH OF BRACKETS:"<<PAD<<STD::ENDL
        <<"SYMBOL=
"<<TEXT[I]<<STD::ENDL<<"_____"<<STD::ENDL;
        #ENDIF
        IF(TEXT[I]=='(')
        {
                STK.PUSH('(',CHECK_NEXT_SYMBOL(TEXT,I+1) );

                        #IFDEF TEST_ALG
                                IF(CHECK_NEXT_SYMBOL(TEXT,I+1))
                                        PAD++;
                        #ENDIF
        }
        IF(TEXT[I]==')')
        {
                STK.PUSH(')',FALSE);
```

```
        }
}

        IF(FIRST_CORRECT_BRACKET)
                {
                        #IFDEF SCRIPT_TEST
                        FILE *F;
                F = FOPEN("OUTPUT_OF_LAB2.TXT","WT");
                FPRINTF(F, "%D",STK.COUNT());
                FCLOSE(F);
                        RETURN 0;
                        #ELSE
                        STD::COUT<<"RESULT="<<STK.COUNT()<<STD::ENDL;
                        RETURN 0;
                        #ENDIF
                }

        #IFDEF SCRIPT_TEST
                        FILE *F;
                F = FOPEN("OUTPUT_OF_LAB2.TXT","WT");
                FPRINTF(F, "ERROR");
                FCLOSE(F);
                        RETURN 0;
                        #ELSE
                        STD::COUT<<"ERROR"<<STD::ENDL;
                        #ENDIF

        RETURN 0;
}
```

```
#INCLUDE <IOSTREAM>
STRUCT LEFT_BRACKET{                                    //IT'S LEFT SON
        BOOL IS_ATOM;
        STRUCT EL_OF_LIST*ELEMENT;
};
STRUCT RIGHT_BRACKET{                                   //IT'S RIGHT SON
        STRUCT EL_OF_LIST*ELEMENT;
};
STRUCT EL_OF_LIST{                                              //NO BROTHERS
STRUCT LEFT_BRACKET*LEFT;
STRUCT RIGHT_BRACKET*RIGHT;
};
CLASS MYLIST
{
PRIVATE:
        STRUCT EL_OF_LIST*EL;
        STRUCT EL_OF_LIST* LIST;
        STRUCT EL_OF_LIST*EL1;
        INT SIZE_;
PUBLIC:
        MYLIST()
        {
                LIST = NEW STRUCT EL_OF_LIST;
                LIST->LEFT=NULL;
                LIST->RIGHT=NULL;
                EL = LIST;
                EL1=LIST;
                SIZE_=0;
        }
        ~MYLIST()
        {
                        DELETE_LIST(EL1);
        }
        INT SIZE()
        {
                RETURN SIZE_;
        }
```

```
VOID PUSH(CHAR SYMBOL,BOOL NEXT_SYMBOL)
{
        IF(SYMBOL=='(')
        {
                EL->LEFT=NEW STRUCT LEFT_BRACKET;
                EL->RIGHT=NULL;
                EL->LEFT->ELEMENT=NEW STRUCT EL_OF_LIST;
                EL->LEFT->IS_ATOM=NEXT_SYMBOL;
                EL=EL->LEFT->ELEMENT;
                EL->LEFT=NULL;
                EL->RIGHT=NULL;
        }
        IF(SYMBOL==')')
        {
                EL->RIGHT=NEW STRUCT RIGHT_BRACKET;
                EL->LEFT=NULL;
                EL->RIGHT->ELEMENT=NEW STRUCT EL_OF_LIST;
                EL=EL->RIGHT->ELEMENT;
                EL->LEFT=NULL;
                EL->RIGHT=NULL;
        }
}
INT COUNT()
{
        IF(LIST->LEFT==NULL&&LIST->RIGHT==NULL)
                RETURN 0;

                IF(LIST->LEFT!=NULL)
                {
                        IF(LIST->LEFT->IS_ATOM)
                                {
                                        LIST=LIST->LEFT->ELEMENT;
                                        RETURN COUNT()+1;
                                }
                        ELSE
                                {
                                        LIST=LIST->LEFT->ELEMENT;
                                        RETURN COUNT();
```

```
                    }
                }

                IF(LIST->RIGHT!=NULL)
                {
                    LIST=LIST->RIGHT->ELEMENT;
                    RETURN COUNT();
                }

        }

        VOID DELETE_LIST(STRUCT EL_OF_LIST*CUR_EL)
        {
            IF(CUR_EL->LEFT==NULL&&CUR_EL->RIGHT==NULL)
                {
                    IF(CUR_EL)
                        DELETE CUR_EL;
                }
            IF(CUR_EL->LEFT!=NULL)
                {
                    DELETE_LIST(CUR_EL->LEFT->ELEMENT);
                    IF(CUR_EL->LEFT)
                        DELETE CUR_EL->LEFT;
                }
            IF(CUR_EL->RIGHT!=NULL)
                {
                    DELETE_LIST(CUR_EL->RIGHT->ELEMENT);
                    IF(CUR_EL->RIGHT)
                        DELETE CUR_EL->RIGHT;
                }
        }
};
```