МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

КАФЕДРА МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 7383 _____ Ласковенко Е.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Оглавление

Цель работы

Познакомиться с основными понятиями и приемами работы с иерархическими списками, получить навыки программирования рекурсивных функций на языке программирования C++.

Задача

Подсчитать число различных атомов в иерархическом списке; сформировать из них линейный список;

Реализация задачи

Описание функций, классов и переменных. Описание работы алгоритма

Функция **int main()**:

Является головной функцией. В ней происходит считывание данных с консоли или из файла (по выбору пользователя), объявляются необходимые переменные, описанные ниже, происходит вызов функций построения списка, функции вывода на экран построенного списка, функции построения линейного списка на основе иерархического в соответствии с заданием, функции вывода линейного списка на экран и количества атомов этого списка, вызов деструкторов используемых объектов классов. Все вызовы находятся в цикле, выходом из которого является условие ввода пользователем обозначенной цифры в консоли. Также в функции реализована обработка исключений — проверка на валидность входных данных, проверка наличия файла с данными.

Объявляемые переменные:

- int int_temp — целочисленная переменная, необходимая для перемещения по кейсам условий switch/case.
- string file_name — строковая переменная, хранящая имя файла, с которого необходимо считать данные.
- string temp_str — строковая переменная, хранящая входные данные.
- filebuf file — файловый буфер, используемый при считывании входных данных с файла.
- stringbuf str_buf — строковый буфер, используемый при считывании входных данных с консоли.
- istream is_str — входной поток, в который помещаются входные данные.
- auto head — объект класса Pair, голова иерархического списка.
- auto result — объект класса Atoms, голова линейного списка.

- char istr_null — символьная переменная, необходимая для проверки входных данных на валидность.
- auto size — переменная, хранящая размер (количество символов) файла с входными данными.

Функция **Pair* set_right_list(istream& is_str, Pair* c_pair, base prev_ch):**

Рекурсивная функция, принимающая на вход входной поток с исходными данными и объект класса — головы иерархического списка. Использует методы классов Pair и List. Обеспечивает промежуточную проверку входных данных и строит иерархический список по мере прохождения по символам входных данных. Состоит из вложенных условий. Внешним условием является проверка на конец входного потока. Внутренним условием является проверка на то, является ли первым элементом — открывающая скобка. Если это так, то происходит рекурсивный вызов без инициализации какого-либо объекта класса (пропуск первой скобки). Далее объявляются 2 объекта класса List — голова и хвост объекта класса Pair. Следом происходят проверки следующих условий: является ли текущий символ буквой (если да, голова инициализируется этим атомом, происходит рекурсивный вызов, возвращающий объект класса Pair в хвост), является ли текущий символ открывающей скобкой (если да, это означает, что следующий атом находится на нижнем уровне, при этом совершается 2 последовательных рекурсивных вызова, которые вернут объекты класса Pair в голову и в хвост), является ли текущий символ закрывающей скобкой (если да, текущий рекурсивный вызов возвращает нулевой указатель, что свидетельствует о завершении текущего уровня). Если ни одно из вышеперечисленных условий не выполняется, то это свидетельствует о невалидности входных данный, происходит выброс исключения и его последующая интерпретация в головной функции. При этом промежуточные

объекты классов уничтожаются, тем самым освобождая память. В конце алгоритма происходит проверка последнего элемента - он должен являться закрывающей скобкой. Если это не так, то это свидетельствует о невалидности данных.

Объявляемые переменные:

- char cur_ch — символьная переменная, текущий просматриваемый символ.
- auto c_hd — объект класса List, голова в текущем объекте класса Pair.
- auto c_tl — объект класса List, хвост в текущем объекте класса Pair.
- auto n_hd_pair — объект класса Pair, передаваемый в новый рекурсивный вызов в качестве следующей пары указателей (класс Pair). На этот объект будет указывать голова текущей пары.
- auto n_tl_pair — объект класса Pair, передаваемый в новый рекурсивный вызов в качестве следующей пары указателей (класс Pair). На этот объект будет указывать хвост текущей пары.
- auto n_pair — объект класса Pair, передаваемый в новый рекурсивный вызов в качестве следующей пары указателей (класс Pair). На этот объект будет указывать хвост текущей пары.

Функция **void cout_list(Pair* head):**

Рекурсивная функция, которая проходит по иерархическому списку и выводит его на экран. Использует методы классов Pair и List. Алгоритм функции следующий: происходит проверка текущего элемента списка, является ли он непустым. Если да, происходит проверка, является ли голова текущей пары атомом. Если да, то происходит вывод этого атома на экран. Если нет, то происходит рекурсивный вызов, причем в качестве новой рассматриваемой пары берется пара из головы текущей (понижение уровня списка). После происходит рекурсивный вызов, в качестве новой пары в этом

случае берется пара из хвоста. Алгоритм заканчивает работу при обнаружении конца списка.

Функция **Atoms* diff_atoms(Pair* head, Atoms* h_res, Atoms* c_res):**

Рекурсивная функция, которая, проходя по списку, обнаруживает различные атомы и составляет новый список — линейный, состоящий из различных атомов. Основа алгоритма похожа на алгоритм предыдущей функции, однако в условии, проверяющем на принадлежность головы пары к атому, добавлен следующий цикл — в нем происходит проверка каждого атома в текущем линейном списке на уникальность. Если рассматриваемый атом уникален — он добавляется в конец текущего линейного списка (если текущий список пустой, создается первый элемент). В результате функция возвращает указатель на объект класса Atoms — голову линейного списка.

Объявляемые переменные:

- base cur_atom — символьная переменная, текущий атом.
- bool diff — булевая переменная, является флагом, указывающим на то, является ли текущий атом уникальным.
- Atoms* tmp_c — объект класса Atoms, используется для проходжения по текущему линейному списку для проверки атома на уникальность.

Функция **void cout_diff_list(Atoms* head):**

Функция, осуществляющая вывод на экран линейного списка и подсчитывающая количество элементов в нем с выводом на экран. Использует целочисленную переменную int count, которая является счетчиком элементов. Функция проверяет, является ли список пустым. Если да, то выводит соответствующее сообщение на экран. Основа функции — цикл, который осуществляет проход по списку через указатели элемента списка на следующие.

Также были реализованы функции **bool syn_analyzer(string str, unsigned int i, unsigned int opened_br, bool flag)** и **Pair* set_list(istream& is_str, bool st_flag)**. Первая функция является синтаксическим анализатором, который проверяет строку на валидность. Вторая составляет иерархический список без проверки на валидность. Однако, функции оказались не востребованы, так как была реализована функция, совмещающая эти две, по указанию преподавателя.

Класс **Atoms:**

Класс линейного списка. Состоит из следующих полей: символьной переменной atom и указателя next на следующий элемент. Для класса были реализованы следующие методы:

- Atoms() - конструктор класса. Инициализирует поля нулевыми значениями.
- void set_atom(base am) — метод, принимающий на вход символьную переменную и инициализирующий поле атома.
- base get_atom() - метод, возвращающий значение символьной переменной.
- void set_next(Atoms* nxt) - метод, принимающий на вход указатель на объект класса Atoms и инициализирующий поле указателя.
- Atoms* get_next() - метод, возвращающий значение указателя.
- ~Atoms() - деструктор класса.

Класс **Pair:**

Класс, состоящий из полей, которые являются указателями на объекты класса List. Первый указатель — голова объекта, второй — хвост. Для класса были реализованы следующие методы:

- Pair() - конструктор класса, инициализирующий поля нулевыми значениями.

- void set(class List* head, class List* tail) — метод, принимающий на вход 2 указателя на объекты класса List и инициализирующий поля объекта текущего класса .
- class List* get_head() - метод, возвращающий значение указателя-головы.
- class List* get_tail() - метод, возвращающий значение указателя-хвоста.
- bool is_null() - метод, возвращающий значение true, если текущий объект является нулевым указателем. Иначе возвращает false.
- ~Pair() - деструктор класса.

Класс **List:**

Класс, состоящий из следующих полей: флаг, указывающий, является ли объект атомом и объединение, в котором может храниться либо символьная переменная — атом, либо указатель на объект класса Pair. Для класса были реализованы следующие методы:

- List() - конструктор класса, инициализирующий поля нулевыми значениями.
- void set_atom(base atom) - метод, принимающий на вход символьную переменную и инициализирующий поле атома.
- base get_atom() - метод, возвращающий значение символьной переменной.
- void set_pair(class Pair* ptr) - метод, принимающий на вход указатель на объект класса Pair и инициализирующий поле объединения этим указателем.
- class Pair* get_pair() - метод, возвращающий значение указателя на объект класса Pair.
- bool is_atom() - метод, возвращающий значение true, если текущий объект является атомом. Иначе возвращает false.
- ~List() - деструктор класса.

Тестирование программы

Процесс тестирования

Программа собрана в операционной системе Linux Mint 19, с использованием компилятора G++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования

Тестовые случаи представлены в Приложении А.

По результатам тестирования было показано, что поставленная задача была выполнена.

Выводы

В ходе лабораторной работы были получены навыки работы с иерархическими списками. Иерархические списки позволяют упростить написание и повысить читабельность кода, если требуется выполнить задачу, подразумевающую работу с повторяющимися типами данных. Рекурсия, использованная при реализации списков, позволяет облегчить выполнение поставленной задачи.

ПРИЛОЖЕНИЕ А

Тестовые случаи

Входные данные	Вывод	Верно?
(a a a b (c d d e f f g) b b f h (f g h j k (u y (t r e (u t r) y) l j g n (o)) e h q w) z x (o p l) q w e r t y u)	List is set. Result: (a a a b (c d d e f f g) b b f h (f g h j k (u y (t r e (u t r) y) l j g n (o)) e h q w) z x (o p l) q w e r t y u) Linear list of different atoms: a -- b -- c -- d -- e -- f -- g -- h -- j -- k -- u -- y -- t -- r -- l -- n -- o -- q -- w -- z -- x -- p -- Number of different atoms: 22	Да
(abcvbrgnbfgbgfbbgf)	List is set. Result: (a b c v b r g n b f g b g f b b g f) Linear list of different atoms: a -- b -- c -- v -- r -- g -- n -- f -- Number of different atoms: 8	Да
fgregrewv)	It isn't an expression!	Да
(a()b)	List is set. Result: (a () b) Linear list of different	Да

	atoms: a -- b -- Number of different atoms: 2	
()	List is set. Result: (()) Linear list of different atoms: EMPTY.	Да
(erg)(tehddtrwteg)	It isn't an expression!	Да

ПРИЛОЖЕНИЕ Б

Код программы

1. Код **main.cpp:**

```
#include <fstream>
#include <exception>
#include "header.h"
#include "atoms.h"
#include "list.h"
#include "pair.h"

using namespace std;

int main()
{
    int int_temp = -1;
    //case 1, 2
    string file_name;
    string temp_str;
    filebuf file;
    //case 3
    stringbuf str_buf;
    istream is_str(&str_buf);
    while(int_temp)
    {
        auto head = new Pair;
        auto result = new Atoms;
        char istr_null ='\0';
        switch(int_temp)
        {
        case 1:
            cout << "Enter a name of the data-file:"
                 << endl;
            getline(cin, file_name);
            if(!file.open(file_name, ios::in))
                cout << "Input file isn't opened."
                     << endl;
            else
            {
                auto size = file.in_avail();
                for(auto c_size = 0; c_size<size; c_size++)
                        temp_str.append(1, file.sbumpc());
                cout << "File contains: "
                     << temp_str
                     << endl;
                file.close();
                int_temp = 3;
            }
            break;
        case 2:
            cout << "Enter an expression:"
                 << endl;
            try
            {
                getline(cin, temp_str);
                if(temp_str=="")
                    throw 0;
            }
            catch (...)
            {
```

```
            cout << "Empty string!"
                 << endl;
            break;
        }
        int_temp = 3;
        break;
    case 0:
        break;
    case 3:
        cout << "============================================="
             << endl;
        str_buf.str(temp_str);
        try
        {
            set_right_list(is_str, head);
            if(is_str>>istr_null)
                throw 0;
        }
        catch (...)
        {
            cout << "It isn't an expression!"
                 << endl
                 << "============================================="
                 << endl;
            delete head;
            delete result;
            temp_str.clear();
            is_str.clear();
            int_temp = -1;
            break;
        }
        cout << "List is set. Result: "
             << endl
             << '(';
        cout_list(head);
        cout << ')'
             << endl;
        diff_atoms(head, result, result);
        cout_diff_list(result);
        cout << "============================================="
             << endl;
        delete head;
        delete result;
        temp_str.clear();
        is_str.clear();
        int_temp = -1;
        break;
    default:
        cout << "Enter \"1\" to input data from file."
             << endl
             << "Enter \"2\" to input data from console."
             << endl
             << "Enter \"0\" to exit."
             << endl;
        try
        {
            string t_str;
            getline(cin, t_str);
            int_temp = stoi(t_str);
            if(int_temp == 3)
                int_temp = -1;
        }
```

```
            catch(...)
            {
                cout << "It isn't an integer, retard!"
                     << endl;
            }
            break;
        }
    }
    return 0;
}
```

2. Код **functions.cpp:**

```
#include "header.h"
#include "atoms.h"
#include "list.h"
#include "pair.h"

Pair* set_right_list(istream& is_str, Pair* c_pair, base prev_ch)
{
    char cur_ch;
    if(is_str>>cur_ch)
    {
        if(cur_ch=='(' && prev_ch=='\0')
        {
            set_right_list(is_str, c_pair, cur_ch);
        }
        else
        {
            auto c_hd = new List;
            auto c_tl = new List;
            c_pair->set(c_hd, c_tl);
            if(prev_ch>=97 && prev_ch<=122 && cur_ch=='(')
            {
                auto n_hd_pair = new Pair;
                c_hd->set_pair(set_right_list(is_str, n_hd_pair, cur_ch));
                auto n_tl_pair = new Pair;
                c_tl->set_pair(set_right_list(is_str, n_tl_pair, cur_ch));
                return c_pair;
            }
            else if(prev_ch!='\0' && cur_ch>=97 && cur_ch<=122)
            {
                c_hd->set_atom(cur_ch);
                auto n_pair = new Pair;
                c_tl->set_pair(set_right_list(is_str, n_pair, cur_ch));
                return c_pair;
            }
            else if(cur_ch==')')
            {
                return nullptr;
            }
            else
            {
                throw 0;
            }
        }
    }
    else if(prev_ch!=')')
        throw 0;
    return nullptr;
}
```

```
bool syn_analyzer(string str, unsigned int i, unsigned int opened_br, bool
flag) {
    if(str.empty())
        return false;
    if(i<str.length()) {
        if(str[i]==' ')
            i++;
        if(!flag && str[i]=='(')
            flag = syn_analyzer(str, ++i, ++opened_br, true);
        else if(opened_br>0 && (str[i]>=97 && str[i]<=122))
            flag = syn_analyzer(str, ++i, opened_br, false);
        else if(str[i]==')')
            flag = syn_analyzer(str, ++i, --opened_br, true);
        else
            return false;
    }
    else if(opened_br==0)
        return true;
    else
        return false;
    return flag;
}
Pair* set_list(istream& is_str, bool st_flag) {
    char ch;
    if(st_flag)
        is_str >> ch;
    if(is_str >> ch) {
        auto c_pair = new Pair;
        auto c_hd = new List;
        auto c_tl = new List;
        c_pair->set(c_hd, c_tl);
        if(ch>=97 && ch<=122) {
            c_hd->set_atom(ch);
            c_tl->set_pair(set_list(is_str, false));
        }
        else if(ch=='(') {
            c_hd->set_pair(set_list(is_str, false));
            c_tl->set_pair(set_list(is_str, false));
        }
        else if(ch==')') {
            return nullptr;
        }

        return c_pair;
    }
    return nullptr;
}
void cout_list(Pair* head) {
    if(!head->is_null()) {
        if(head->get_head()->is_atom()) {
            cout << head->get_head()->get_atom();
            if(!head->get_tail()->get_pair()->is_null() &&
                head->get_tail()->get_pair()->get_head()->is_atom())
                cout << ' ';
        }
        else {
            cout << ' ' << '(';
            cout_list(head->get_head()->get_pair());
            cout << ')' << ' ';
        }
        cout_list(head->get_tail()->get_pair());
    }
```

```
}
Atoms* diff_atoms(Pair* head, Atoms* h_res, Atoms* c_res) {
    if(!head->is_null()) {
        if(head->get_head()->is_atom()) {
            base cur_atom = head->get_head()->get_atom();
            bool diff = true;
            Atoms* tmp_c = h_res;
            while(tmp_c!=nullptr){
                if(tmp_c->get_atom() == cur_atom)
                    diff = false;
                tmp_c = tmp_c->get_next();
            }
            if(diff) {
                c_res->set_atom(cur_atom);
                c_res->set_next(new Atoms);
                c_res = c_res->get_next();
            }
        }
        else
            c_res = diff_atoms(head->get_head()->get_pair(), h_res, c_res);
        c_res = diff_atoms(head->get_tail()->get_pair(), h_res, c_res);
    }
    return c_res;
}
void cout_diff_list(Atoms* head) {
    int count = 0;
    cout << "Linear list of different atoms: "
         << endl;
    while(head!=nullptr){
        if(head->get_next()!= nullptr) {
            cout << head->get_atom() << " -- ";
            count++;
        }
        head = head->get_next();
    }
    if(count) {
        cout << endl
             << "Number of different atoms: "
             << count
             << endl;
    }
    else
        cout << "EMPTY."
             << endl;
}
```

3. Код **atoms.cpp:**

```
#include "atoms.h"

Atoms::Atoms()
{
    atom = '\0';
    next = nullptr;
}
void Atoms::set_atom(base am)
{
    atom = am;
}
base Atoms::get_atom()
{
    return atom;
```

```
}
void Atoms::set_next(Atoms* nxt)
{
    next = nxt;
}
Atoms* Atoms::get_next()
{
    return next;
}
Atoms::~Atoms()
{
    delete next;
}
```

4. Код **pair.cpp:**

```
#include "pair.h"
#include "list.h"

Pair::Pair()
{
    hd = nullptr;
    tl = nullptr;
}
void Pair::set(List* head, List* tail)
{
    hd = head;
    tl = tail;
}
List* Pair::get_head()
{
    return hd;
}
List* Pair::get_tail()
{
    return tl;
}
bool Pair::is_null()
{
    return this == nullptr;
}
Pair::~Pair()
{
    delete hd;
    delete tl;
}
```

5. Код **list.cpp:**

```
#include "list.h"
#include "pair.h"

List::List() {
    flag = false;
}
void List::set_atom(base atom) {
    flag = true;
    node.atom = atom;
}
base List::get_atom() {
    return node.atom;
```

```
}
void List::set_pair(Pair* ptr) {
    flag = false;
    node.p_ptr = ptr;
}
Pair* List::get_pair() {
    return node.p_ptr;
}
bool List::is_atom() {
    return this->flag;
}
List::~List() {
    if(!flag && node.p_ptr!=nullptr)
        delete node.p_ptr;
}
```

6. Код **header.h:**

```
#ifndef HEADER_H
#define HEADER_H

#include <iostream>
#include <sstream>

//Namespace:
using namespace std;

//Types definition:
typedef char base;

//Signatures of functions:
class Pair* set_right_list(istream& is_str, Pair* c_pair, base prev_ch='\0');
bool syn_analyzer(string str, unsigned int i=0, unsigned int opened_br=0,
bool flag=false);
class Pair* set_list(istream& is_str, bool st_flag=true);
void cout_list(class Pair* head);
class Atoms* diff_atoms(class Pair* head, class Atoms* h_res, class Atoms*
c_res);
void cout_diff_list(class Atoms* head);

#endif
```

7. Код **atoms.h:**

```
#ifndef ATOMS_H
#define ATOMS_H

typedef char base;

class Atoms
{
public:
    Atoms();
    void set_atom(base am);
    base get_atom();
    void set_next(Atoms* nxt);
    Atoms* get_next();
    ~Atoms();
private:
    base atom;
    Atoms* next;
};
```

```
#endif
```

8. Код **pair.h:**

```
#ifndef PAIR_H
#define PAIR_H

class Pair
{
public:
    Pair();
    void set(class List* head, class List* tail);
    class List* get_head();
    class List* get_tail();
    bool is_null();
    ~Pair();
private:
    class List* hd;
    class List* tl;
};

#endif
```

9. Код **list.h:**

```
#ifndef LIST_H
#define LIST_H

typedef char base;

class List
{
public:
    List();
    void set_atom(base atom);
    base get_atom();
    void set_pair(class Pair* ptr);
    class Pair* get_pair();
    bool is_atom();
    ~List();
private:
    bool flag; //true: atom, false: pair
    union {
        base atom;
        class Pair* p_ptr;
    }node{};
};

#endif
```

10. Код **Makefile:**

```
all: main.o functions.o atoms.o pair.o list.o
	g++ main.o functions.o atoms.o pair.o list.o -o start


main.o: main.cpp header.h atoms.h pair.h list.h
	g++ -c main.cpp
functions.o: functions.cpp header.h atoms.h pair.h list.h
	g++ -c functions.cpp
```

```
atoms.o: atoms.cpp atoms.h
	g++ -c atoms.cpp
pair.o: pair.cpp pair.h list.h
	g++ -c pair.cpp
list.o: list.cpp list.h pair.h
	g++ -c list.cpp

clean:
	rm *.o start
```