# ВВЕДЕНИЕ

На сегодняшний день всё большее количество людей сталкивается с компьютером, прогресс неумолимо движет нас вперёд. В данное время это обусловлено большими информационными потоками, и необходимо обеспечивать эту отрасль специалистами информационных технологий.

Одно из наиболее мощных свойств языка программирования – указатели. Указатели – одна из наиболее трудных для освоения возможностей С. Указатели предоставляют программам возможность моделировать передачу по ссылке и создавать и манипулировать динамическими структурами данных, т.е. структурами данных, которые могут нарастать и сокращаться, например, такими как связные списки, очереди, стеки и деревья.

Переменные-указатели содержат в качестве своих значений адреса памяти. Обычно переменная содержит определенное значение. С другой стороны, указатель содержит адрес переменной, которая содержит определенное значение. В этом смысле имя переменной отсылает к значению прямо, а указатель – косвенно. Ссылка на значение посредством указателя называется косвенной адресацией.

Указатель – это переменная, содержащая адрес в памяти компьютера. Если удастся осознать смысл этого простого предложения, то это и все, что необходимо знать об указателях. Указатель – это переменная, которая содержит адрес оперативной памяти.

Чтобы лучше понять указатели, рассмотрим устройство оперативной памяти компьютера. Оперативная память разделена на последовательно пронумерованные ячейки. Каждая переменная размещается в одной или нескольких последовательно расположенных отдельных ячейках памяти, адрес первой из них называется адресом переменной. Каждая ячейка в оперативной памяти занимает 1 байт или 8 бит.

## 1. ПОСТАНОВКА ЗАДАЧИ

Задание №1:

Описать функцию, которая меняет местами первый и предпоследний элемент непустой очереди.

Входные данные:

Запись{inf}: цел – элементы очереди;

Промежуточные данные:

kol: цел – счетчик(номера элементов очереди);

key: сим – клавиша события;

tmp: сим – временный массив символов;

Ограничения:

нет

Задание №2:

Определить количество изолированных вершин неориентированного графа, вывести их список. Сделать выбранную вершину неизолированной.

Входные данные:

Запись {v1, v2}: цел – 1-я и 2-я вершины одного ребра;

n: цел – общее количество вершин в графе.

Промежуточные данные:

i: цел – счетчик(номера элементов 1-ой очереди);

f: цел – счетчик(проверка на существование висячих вершин);

ch: сим – клавиша события;

s,s1: сим – строки, которые нужны для проверки введенных результатов;

v [10]: сим – показатель степени данной вершины.

Ограничения:

2<=n<=5;

1<=v1<=n;

1<=v2<=n;

v1<>v2.

Выходные данные:

Запись {v1, v2}: цел – граф после удаления одной из висячих вершин.

## 2. ОБРАБОТКА СПИСКОВ

### 2.1. Описание структуры списка

Линейный список - это конечная последовательность однотипных элементов (узлов), возможно, с повторениями. Количество элементов в последовательности называется длиной списка, причем длина в процессе работы программы может изменяться. Линейный список F, состоящий из элементов D1,D2,...,Dn, записывают в виде последовательности значений заключенной в угловые скобки F=, или представляют графически (рисунок 2.1).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 |  | D2 |  | D3 |  | ... |  | Dn |

Рисунок 2.1. - Изображение линейного списка

Например, F1=< 2,3,1>,F2=< 7,7,7,2,1,12 >, F3=< >. Длина списков F1, F2, F3 равна соответственно 3,6,0. В реальных языках программирования нет какой-либо структуры данных для представления линейного списка. Поэтому при работе с линейными списками важным является представление используемых в программе линейных списков таким образом, чтобы была обеспечена максимальная эффективность и по времени выполнения программы, и по объему требуемой памяти. Методы хранения линейных списков разделяются на методы последовательного и связанного хранения. При последовательном хранении элементы линейного списка размещаются в массиве d фиксированных размеров, например, 100, и длина списка указывается в переменной l, т.е. в программе необходимо иметь объявления вида

float d [100] ; int l;

Размер массива 100 ограничивает максимальные размеры линейного списка. Список F в массиве d формируется так:

d [0] =7; d [1] =10; l=2;

Полученный список хранится в памяти согласно схеме на рисунке 2.2.

|  |  |
| --- | --- |
| l: | 2 |
| d: | 7 | 10 |  |  | ... |  |  |
|  | [0] | [1] | [2] | [3] |  | [98] | [99] |
| Рисунок 2.2. – Последовательное хранение линейного списка | | | | | | | |

При связанном хранении в качестве элементов хранения используются структуры, связанные по одной из компонент в цепочку, на начало которой (первую структуру) указывает указатель dl. Структура образующая элемент хранения, должна кроме соответствующего элемента списка содержать и указатель на соседний элемент хранения.

Описание структуры и указателя в этом случае может имееть вид:

typedef struct snd /\* структура элемента хранения \*/

{ float val; /\* элемент списка \*/

struct snd \*n; /\* указатель на элемент хранения \*/

} DL;

DL \*p; /\* указатель текущего элемента \*/

DL \*dl; /\* указатель на начало списка \*/

Для выделения памяти под элементы хранения необходимо пользоваться функцией malloc(sizeof(DL)) или calloc(l,sizeof(DL)). Формирование списка в связанном хранении может осуществляется операторами:

p=malloc(sizeof(DL));

p->val=10; p->n=NULL;

dl=malloc(sizeof(DL));

dl->val=7; dl->n=p;

В последнем элементе хранения (конец списка) указатель на соседний элемент имеет значение NULL. Получаемый список изображен на рисунке 2.3.



Рисунок 2.3. – Связное хранение линейного списка

### 2.2. Операции над элементами списка

При простом связанном хранении каждый элемент списка представляет собой структуру graf, состоящую из двух элементов: v - предназначен для хранения элемента списка, next - для указателя на структуру, содержащую следующий элемент списка. На первый элемент списка указывает указатель head. Для всех операций над списком используется описание:

typedef struct graf

{ int v;

struct graf\* next;

} Graf;

Graf \*g, \*head, \*teil;

Для реализации операций могут использоваться следующие фрагменты программ:

1) определить первый элемент в линейном списке (рисунок 2.4):

printf("v=");

scanf("%d",&head->v);

head->next=0;

teil->next=0;

teil=head;

head: NULL

Рисунок 2.4. – Создание первого элемента в списке

2) вставить дополнительный элемент после указанного узла (рисунок 2.5):

printf("v=");

scanf("%d",&g->v);

teil->next=g;

teil=teil->next;

new

head: NULL

Рисунок 2.5. – Вставка дополнительного элемента

3) печать всех элементов списка:

g=head;

i=0;

while(g! =NULL)

{

i++;

printf("Эллемент%d:%d\n", i,g->v1);

g=g->next;

}

### 2.3. Описание программной реализации

Данная программа реализована с помощью пяти функций, которые частично демонстрируют работу с такой структурой данных как очередь.

Очередь аналогична очереди людей в супермаркете: первый клиент в ней обслуживается первым, а другие клиенты занимают очередь с конца и ожидают, когда их обслужат. Узлы очереди удаляются только из головы очереди, а помещаются в очередь только в ее хвост. По этой причине, очередь – это структура данных типа "первым вошел – первым вышел" (first-in, first-out – FIFO). У очередей имеется множество применений в вычислительных системах. У большинства компьютеров имеется только один процессор, поэтому в один и тот же момент времени может быть обслужен только один пользователь. Запросы остальных пользователей помещаются в очередь. Каждый запрос постепенно продвигается в очереди вперед по мере того, как происходит обслуживание пользователей. Запрос в начале очереди является очередным кандидатом на обслуживание.

Первые четыре функции для формирования очереди похожи между собой: первые две из них: vvod1 и vvod2 нужны для созданий первых элементов двух очередей. Две остальные: dobavl1 и dobavl2, для добавления новых элементов в уже созданные нами очереди. Последняя функция: proga, содержит первые четыре и ряд новых возможностей, в который входят: возможность вывода на экран всех элементов очередей, а также возможность проверки на вхождение всех элементов первой очереди во вторую. Так как структуры можно сравнивать только поэлементно в данной функции в цикле каждый элемент сравнивается с отдельными элементами второй очереди, если элемент первой очереди встречается во второй, значит что этот элемент входит во вторую очередь. В конце расчетов на экран пользователю выводится соответствующее сообщение о том, есть ли вхождение одной очереди в другую или его нет.

#### 2.3.1. Описание процедур и функций языка

void \*malloc(size\_t size) – данная функция используется для выделения памяти из кучи в байтах. Для таких информационных структур, таких как деревья и списки выделение памяти происходит таким же способом

void free(void \*block) – данная функция очищает память, которая была выделена такими функциями как calloc, malloc или realloc.

#### 2.3.2. Описание созданных функций для работы со списками

void vvod1() – данная функция создает первый элемент очереди под номером 1 и, используя стандартную функцию malloc, выделят определенное количество памяти под этот первый элемент;

void dobavl1() – функция в цикле, до нажатия клавиши Esc, каждый раз добавляет новый элемент 1-ой очереди и выделяет под него память.

void vvod2() – данная функция создает первый элемент очереди под номером 2 и, используя стандартную функцию malloc, выделят определенное количество памяти под этот первый элемент;

void dobavl2() – функция в цикле, до нажатия клавиши Esc, каждый раз добавляет новый элемент 2-ой очереди и выделяет под него память.

void proga() – эта функция содержит в себе все выше перечисленные, поэтому она создает первые элементы 1-ой и 2-ой очередей и добавляет в них новые элементы. Также эта функция выводит на экран полученные очереди, то есть выводит на экран все элементы двух очередей, после чего функция производит сравнение: входит ли 1-я очередь во 2-ю, и выводит результат на экран.

## 3. ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕСКИХ СТРУКТУР ПРИ РАБОТЕ С ГРАФАМИ

### 3.1. Способы представления графов

Для представления графа можно использовать матрицу смежности, матрицу инцидентности либо представить его списком дуг.

Основные способы представления (задания) графов:

1) перечисление множеств V (вершин) и E (ребер), задающих граф. G = (V, E);

2) матричные способы описания;

Пусть G = (V, E), |V| = p, а |E| = q, тогда:

a) матрица смежности – квадратная матрица , где



b) матрица инцидентности – прямоугольная матрица .



3) список дуг:

Пример: задан граф G = (V, E), где V = {v1, v2, v3, v4}, E = {v1v2, v2,v3, v1v3, v1v4, v3,v4} = {e1, e2, e3, e4, e5}. Рисунок 3.1.

v2

v1

v3

v4

Рисунок 3.1. – Пример построения графа

В данной программе граф представлен списком дуг, в котором каждая вершина содержит информацию о смежном ей ребре, а каждое ребро содержит информацию о тех вершинах, которые ей смежные. Этот способ более экономичен в плане использования памяти.

v1, v2

v2 v3

v3v4

head: NULL

Рисунок 3.2. – Пример связности ребер графа

### 3.2. Операции над графами

При связанном хранении каждая вершина графа представляет собой структуру graf, состоящую из двух элементов: v1,v2 - предназначены для хранения вершин графа, next - для указателя на структуру, содержащую следующую вершину графа. На первые элементы графа указывает указатель head. Для всех операций над графом используется описание:

typedef struct graf

{ int v1,v2;

struct graf\* next;

} Graf;

Graf \*g, \*head, \*temp;

Для реализации операций могут использоваться следующие фрагменты программ:

1) определить первый элемент в линейном списке (рисунок 3.1):

printf("v1=");

scanf("%d",&head->v1);

v1

v2

head->next=0;

NULL

head NULL

Рисунок 3.3– Создание первого элемента в графе

2) вставить дополнительный элемент после указанного узла (рисунок 3.2):

printf("v=");

scanf("%d",&g->v1);

g->next=head;

v1

new1

v2

head=g;

…. .

head NULL

Рисунок 3.4– Вставка дополнительного элемента

3) печать всех элементов списка:

printf("Ребра графа: \n");

g=head;

i=0;

while(g! =NULL) {

i++;

printf("РЕБРО%d: v1=%d v2=%d\n", i,g->v1,g->v2);

g=g->next; }

4) удаление ребра:



Удалить ребро означает разрушить связь между вершинами, которые являются для данного графа концевыми.

5) удаление вершины:

if((head->v1==i) ||(head->v2==i))

head=head->next;

else{

temp=head;

g=head->next;

while(g) {

if((g->v1==i) ||(g->v2==i)) {

temp->next=g->next;

free(g);

break; }

temp=g;

g=g->next; }}

Удалить вершину означает разрушить связь со смежными ей вершинами и создать новую связь уже без этой вершины. Схема удаления вершины графа, следующей за узлом, на который указывает р находится на рисунке 3.3.



Рисунок 3.5– Схема удаления вершины графа

### 3.3. Описание программной реализации

Данная программа реализована при помощи всего двух функций, которые в полной мере выполняют поставленную перед программой задачу. Задачей является поиск висячих вершин в графе и удаление одной из них, которую укажет пользователь.

Висячей вершиной графа называется такая вершина, степень которой равна единице, то есть вершина, которая имеет всего лишь одно смежное ей ребро. Граф, содержащий висячую вершину изображен на рисунке 3.4.

3 2

1

4 5

Рисунок 3.6– Изображение графа, содержащего висячую вершину

Степени вершин в данном графе таковы: 11, 23, 32, 42, 52, из этого следует, что изолированной вершиной является вершина под номером 1.

Список ребер до удаления висячей вершины в этом графе будет иметь вид: ребо1: 1-2, ребро2: 2-3, ребро3: 3-4, ребро4: 4-5 и ребро5: 5-2.

После удаления единственной висячей список ребер примет вид: ребро1: 2-3, ребро2: 3-4, ребро3: 4-5 и ребро4: 5-2.

Каждая функция данной программы полностью выполняет поставленную перед ней задачу. Первая функция klavа создает первую вершину графа и предлагает пользователю дополнить граф другими вершинами. Функция реализована таким способом, что пользователь вводит вершины попарно, то есть вводит вершины одного ребра. Вторая функция raschet производит поиск висячих вершин графа и, если они имеются, предлагает пользователю выбрать одну из них для ее удаления и удаления смежного ей ребра. В завершение программы на экран пользователю выводится обновленный список ребер введенного графа.

3.3.1. Описание процедур и функций языка

void \*malloc(size\_t size) – данная функция используется для выделения памяти из кучи в байтах. Для таких информационных структур, таких как деревья и списки выделение памяти происходит таким же способом

void free(void \*block) – данная функция очищает память, которая была выделена такими функциями как calloc, malloc или realloc.

3.3.2. Описание созданных функций для работы с динамической памятью, графами

Создание и поддержание динамических структур данных требует динамического распределения памяти: возможности в процессе выполнения программы увеличивать область памяти для хранения новых узлов и освобождать ресурсы памяти, в которых уже нет необходимости. Пределы динамического выделения памяти ограничены только объемом доступной физической памяти или доступной виртуальной памяти в системах с виртуальной памятью. Впрочем, часто эти пределы намного меньше из-за того, что свободная память делится при совместном доступе к ней многих пользователей.

void klava() – данная функция является универсальной, так как изначально она создает первый элемент (первую вершину графа), после чего пользователю предлагается дополнить граф новыми элементами (вершинами). Функция использует стандартную функцию для выделения памяти языка С: malloc.

void raschet() – функция, которая производит ряд проверок в полученном графе на наличие висячих вершин, после чего пользователю предлагается сделать выбор какую из висячих вершин нужно удалить, если таковые имеются. В конечном итоге на экран выводится список ребер графа после удаления висячей вершины.

# ВЫВОДЫ

Желание каждого программиста быть востребованным ставит перед ним определенную цель: идти в одну ногу со временем. С каждым годом это делать становиться все сложнее и сложнее, так как за определенные промежутки времени темпы развития компьютерных технологий постоянно увеличиваются. Задача, которую ставит перед собой программист при разработке нового продукта остается неизменной уже на протяжении долгого времени: получение максимального результата при минимуме затрат времени и средств. Выполнение данной задачи становится возможной только при постоянном самосовершенствовании и самообразовании программиста.

Задание, выданное на летнюю практику, поставило определенные задачи:

1) Научится создавать связные структуры данных, используя указатели;

2) научится создавать и манипулировать динамическими структурами данных, такими как связные списки, очереди и стеки;

3) понять работу различных приложений со связными структурами данных.

Решение выданного задания было реализовано с помощью языка программирования С.

С обладает множеством преимуществ. Он является современным языком программирования, включающим в себя управляющие структуры, наличие которых в языке считается желательным с точки зрения теории и практики программирования. Этот язык построен так, что позволяет естественным образом применять планирование сверху – вниз, структурный подход к программированию, модульное проектирование программ. В результате на С получаются более надежные и “прозрачные программы”.

Вот почему именно язык С был выбран автором для реализации данной задачи.

Структуры – это составные типы данных, построенные с использованием других типов. Классы в С++ являются естественным продолжением структуры struct в С. Вот почему детальное изучение этого раздела является таким необходимым для дальнейшего изучения других языков программирования. Так как прежде чем рассматривать специфику разработки классов на С++ нужно более глубоко разобраться со структурами в С.

# Приложение А

ЛИСТИНГ ПРОГРАММЫ

Задание №1:

#include<stdio. h>

#include<conio. h>

#include<alloc. h>

#include<string. h>

#include<stdlib. h>

// ------------<Структура>---------

typedef struct sp

{

char inf [100] ;

struct sp \*next;

} sp;

sp \*g,\*head,\*teil;

// -----------</Структура>---------

void titl()

{

clrscr();

gotoxy(20,1);

printf("МИHИСТЕРСТВО ОБРАЗОВАHИЯ И HАУКИ УКРАИHЫ");

gotoxy(12,2);

printf("Донецкий государственный институт искусственного интеллекта");

gotoxy(27,8);

printf("Здание на летнюю практику #1");

gotoxy(35,9);

printf("по дисциплине: ");

gotoxy(17,10);

printf("\"Основы программирования и алгоритмические языки. \"");

gotoxy(50,15);

printf("Выполнил: ");

gotoxy(50,16);

printf("студент группы ПО-05г");

gotoxy(50,17);

printf("Пивоваров Артем Генадиевич");

gotoxy(50, 19);

printf("Проверил: ");

gotoxy(50, 20);

printf("асс. Мамбетова Лилия Сергеевна");

gotoxy(2,25);

printf("Для продолжения нажмите любую клавишу... ");

getch();

clrscr();

gotoxy(15,10);

printf("ЗАДАHИЕ: ");

gotoxy(10,12);

printf(" Описать функцию, которая в списке меняет местами");

gotoxy(10,13);

printf("первый и предпоследний элементы. ");

gotoxy(2,25);

printf("Для продолжения нажмите любую клавишу... ");

getch();

}

void program()

{

int kol=1;

char key;

clrscr();

printf("Введите элементы стрруктуры: \n");

head=(sp\*) malloc(sizeof(sp));

g=head;

printf("Введите%i-элемент: ",kol);

scanf("%s",&g->inf);

g->next=0;

teil->next=0;

teil=head;

// --------Ввод остальных эл-тов структуры--------------

kol++;

printf("Введите%i-элемент: ",kol);

g=(sp\*) malloc(sizeof(sp));

scanf("%s",&g->inf);

g->next=0;

teil->next=g;

teil=teil->next;

do

{

if (key! =27)

{

kol++;

printf("Введите%i-элемент: ",kol);

g=(sp\*) malloc(sizeof(sp));

scanf("%s",&g->inf);

g->next=0;

teil->next=g;

teil=teil->next;

}

printf("Для прекращения ввода нажмите ESC; Для продолжения любую клавишу\n");

key=getch();

}

while (key! =27);

printf("Вы ввели такие элементы: \n");

g=head;

kol=0;

while (g! =0)

{

kol++;

printf("Эллемент%i=%s\n",kol,g->inf);

g=g->next;

}

getch();

}

void zamena()

{

char \*tmp;

g=head;

while (g->next! =NULL)

{

if (g->next->next==NULL)

{

strcpy(tmp,g->inf);

strcpy(g->inf,head->inf);

strcpy(head->inf,tmp);

}

g=g->next;

}

// --------------Вывод результата----------------

g=head;

int kol=1;

printf("\nРезультат: ");

while (g! =NULL)

{

printf("\n%i-й элемент равен:%s",kol,g->inf);

kol++;

g=g->next;

}

getch();

}

void cleen()

{

g=head;

while (g! =NULL)

{

free(g);

g=g->next;

}

}

void main()

{

titl();

program();

zamena();

cleen();

}

Задание №2:

// --------------------------------Библиотеки--------------------------------

#include <stdio. h>

#include <conio. h>

#include <alloc. h>

#include <string. h>

#include <stdlib. h>

// --------------------------------------------------------------------------

// ===============

char \*fname [12] ;

FILE \*in,\*out;

int n, i,j;

int v [10] ;

char s [6],s1 [6] ;

// ===============

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef struct graf

{

int v1,v2;

struct graf \*next;

} Graf;

Graf \*g,\*head;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// --------------------------------Реквизиты---------------------------------

// --------------------------------------------------------------------------

void tit()

{

clrscr();

gotoxy(20,1);

printf("МИHИСТЕРСТВО ОБРАЗОВАHИЯ И HАУКИ УКРАИHЫ");

gotoxy(12,2);

printf("Донецкий государственный институт искусственного интеллекта");

gotoxy(27,8);

printf("Здание на летнюю практику #2");

gotoxy(35,9);

printf("по дисциплине: ");

gotoxy(17,10);

printf("\"Основы программирования и алгоритмические языки. \"");

gotoxy(50,15);

printf("Выполнил: ");

gotoxy(50,16);

printf("студент группы ПО-05г");

gotoxy(50,17);

printf("Пивоваров Артем Генадиевич");

gotoxy(50, 19);

printf("Проверил: ");

gotoxy(50, 20);

printf("асс. Мамбетова Лилия Сергеевна");

gotoxy(2,25);

printf("Для продолжения нажмите любую клавишу... ");

getch();

return;

}

// --------------------------------------------------------------------------

// --------------------------------Задание-----------------------------------

// --------------------------------------------------------------------------

void work\_window()

{

clrscr();

gotoxy(35,1);

printf("ЗАДАHИЕ: ");

gotoxy(12,2);

printf(" Определить количество");

gotoxy(12,3);

printf(" изолированных вершин нориентированного графа,");

gotoxy(12,4);

printf("вывести их список. Сделать выбраную вершину неизолированной! \n");

return;

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// ----------------------Ввод данных с клавиатуры---------------------------

void klava()

{

do

{

printf("\nВведите количество вершин: ");

scanf("%s",&s);

n=atoi(s);

itoa(n,s1,10);

if((n<2) ||(n>5))

printf("Ошибка! Выход из диапазона: '2<n<=5'\n");

}

while((n<2) ||(n>5) ||strcmp(s,s1));

// --------------------------------------------------------------------------

for (i=0; i<n; i++)

v [i] =0;

// --------------------------------------------------------------------------

// --------------------Ввод первого элемента списка--------------------------

// --------------------------------------------------------------------------

printf("Введите ребра графа: \n");

head=(Graf \*) malloc(sizeof(Graf));

// ======================================================

do

{

printf("v1=");

scanf("%s",&s);

head->v1=atoi(s);

itoa(head->v1,s1,10);

if (((head->v1) <1) ||((head->v1) >n))

printf("Ошибка! Выход из диапазона: '1<v1<=n'\n");

}

while(((head->v1) <1) ||((head->v1) >n) ||strcmp(s,s1));

// ======================================================

do

{

printf("v2=");

scanf("%s",&s);

head->v2=atoi(s);

itoa(head->v2,s1,10);

if (((head->v2) <1) ||((head->v2) >n))

printf("Ошибка! Выход из диапазона: '1<v2<=n'\n");

}

while(((head->v2) <1) ||((head->v2) >n) ||strcmp(s,s1));

// ======================================================

head->next=NULL;

// --------------------------------------------------------------------------

// -------------------Ввод остальных элементов списка------------------------

// --------------------------------------------------------------------------

char ch=1;

while(ch! =27)

{

printf("Хотите продолжить ввод вершин? \n");

ch=getch();

if (ch! =27)

{

g=(Graf \*) malloc(sizeof(Graf));

// ======================================================

do

{

printf("v1=");

scanf("%s",&s);

g->v1=atoi(s);

itoa(g->v1,s1,10);

if (((g->v1) <1) ||((g->v1) >n))

printf("Ошибка! Выход из диапазона: '1<v1<=n'\n");

}

while(((g->v1) <1) ||((g->v1) >n) ||strcmp(s,s1));

// ======================================================

do

{

printf("v2=");

scanf("%s",&s);

g->v2=atoi(s);

itoa(g->v2,s1,10);

if (((g->v2) <1) ||((g->v2) >n))

printf("Ошибка! Выход из диапазона: '1<v2<=n'\n");

}

while(((g->v2) <1) ||((g->v2) >n) ||strcmp(s,s1));

// ======================================================

g->next=head;

head=g;

}

}

}

// -------------------------------------------------------------------------

// -------------------------------------------------------------------------

// -------------------------------------------------------------------------

int raschet()

{

printf("Ребра графа: \n");

g=head;

i=0;

while(g! =NULL)

{

i++;

printf("РЕБРО%d: v1=%d v2=%d\n", i,g->v1,g->v2);

g=g->next;

}

// --------------------------------------------------------------------------

// --------------Просмотр графа и поиск изолированых вершин------------------

// --------------------------------------------------------------------------

g=head;

while(g! =NULL)

{

v [g->v1-1] ++;

v [g->v2-1] ++;

g=g->next;

}

int f=0;

for(i=1; i<=n; i++)

printf("v [%d] =%d\n", i,v [i-1]);

printf("Изолированные вершины: ");

for(i=1; i<=n; i++)

if (v [i-1] ==0)

printf("%d ", i);

else

if (v [i-1] ! =0)

f=f+1;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (f==n)

{

printf("\nИзолированных вершин HЕТ! ");

// ----------------------------Сохранение в файл------------------------------

i=0;

if ((out=fopen("1. txt", "w")) == NULL)

{

printf("Ошибка открытия входного файла! \n");

return 1;

}

g=head;

while(g! =NULL)

{

i++;

fprintf(out,"Изолированных вершин HЕТ! ");

fprintf(out,"РЕБРО%d: v1=%d v2=%d\n", i,g->v1,g->v2);

g=g->next;

}

fclose(out);

// --------------------------------------------------------------------------

getch();

exit(1);

}

else

// --------------------------------------------------------------------------

// -----------------Добавление вершины в голову списка-----------------------

// --------------------------------------------------------------------------

printf("\nКакую вершину вы хотите сделать не изолированной? \n");

// ======================================================

do

{

printf("Укажите вершину: ");

scanf("%s",&s);

i=atoi(s);

itoa(i,s1,10);

if ((i>n) ||(i<1) ||(v [i-1] ! =0))

printf("Ошибка! \n");

}

while((v [i-1] ! =0) ||strcmp(s,s1) ||(i>n) ||(i<1));

// ======================================================

g=(Graf \*) malloc(sizeof(Graf));

g->v1=i;

g->v2=head->v1;

g->next=head;

head=g;

printf("Ребра графа после добавления одного ребра: \n");

g=head;

i=0;

while(g! =NULL)

{

i++;

printf("РЕБРО%d: v1=%d v2=%d\n", i,g->v1,g->v2);

g=g->next;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ----------------------------Сохранение в файл------------------------------

i=0;

if ((out=fopen("1. txt", "w")) == NULL)

{

printf("Ошибка открытия входного файла! \n");

return 1;

}

g=head;

while(g! =NULL)

{

i++;

fprintf(out," РЕБРО%d: v1=%d v2=%d\n", i,g->v1,g->v2);

g=g->next;

}

fclose(out);

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

void main()

{

clrscr();

// ==============

tit();

work\_window();

klava();

raschet();

// ==============

getch();

}