Министерство науки и высшего образования

Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №3

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "динамические списки"

Выполнили:

студенты группы 23ВВВ4

Королёв Д.В.

Алешин К.А.

Приняли:

Юрова О.В.

Леев М.В.

Пенза, 2024

**Цель работы** - Освоение динамических списков и получение опыта реализации и работы с ними.

**Общие сведения** - Список представляет собой последовательность элементов определенного типа. Простейший тип списка – линейный, когда для каждого из элементов, кроме последнего, имеется следующий, и для каждого, кроме первого имеется предыдущий элемент.

Возможна реализация списков посредством массивов или динамическая реализация.

Динамические списки относятся к динамическим структурам и используются, когда размер данных заранее неизвестен. Созданием динамических данных должна заниматься сама программа во время своего исполнения, этим достигается эффективное распределение памяти, но снижается эффективность доступа к элементам.

Динамические структуры данных отличаются от статических двумя основными свойствами:

1) в них нельзя обеспечить хранение в заголовке всей информации о структуре, поэтому каждый элемент должен содержать информацию, логически связывающую его с другими элементами структуры;

2) для них зачастую не удобно использовать единый массив смежных элементов памяти, поэтому необходимо предусматривать ту или иную схему динамического управления памятью.

Для обращения к динамическим данным применяют указатели.

Набор операций над списком будет включать добавление и удаление элементов, поиск элементов списка.

Различают односвязные, двусвязные и циклические списки.

В простейшем случае каждый элемент содержит всего одну ссылку на следующий элемент, такой список называется односвязным.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка:

**Задание 1. - приоритетная очередь.**

**Листинг**

#include <iostream>

#include <cstdint>

#include <string>

#include <limits>

struct node;

void delete\_node(node\* n);

void info\_for\_node(node\* temp);

struct node {

std::string data;

uint32\_t priority;

node\* next;

node\* prev;

explicit node(const std::string& str, uint32\_t prior)

: data(str), priority(prior), next(nullptr), prev(nullptr)

{}

};

struct priority\_queue {

priority\_queue() : head(nullptr), tail(nullptr), max\_prior(0)

{}

~priority\_queue()

{

while (node\* n = pop())

{

delete n;

}

}

void append(node\* new\_node)

{

if (!head) {

head = tail = new\_node;

max\_prior = new\_node->priority;

return;

}

node\* temp = head;

while (temp != nullptr && new\_node->priority >= temp->priority)

{

temp = temp->next;

}

if (temp == head)

{

new\_node->next = head;

head->prev = new\_node;

head = new\_node;

}

else if (!temp)

{

tail->next = new\_node;

new\_node->prev = tail;

tail = new\_node;

}

else

{

new\_node->next = temp;

new\_node->prev = temp->prev;

temp->prev->next = new\_node;

temp->prev = new\_node;

}

if (new\_node->priority > max\_prior)

{

max\_prior = new\_node->priority;

}

}

void printAll() const

{

if (!head)

{

std::cout << "Queue is empty\n\n";

return;

}

node\* temp = head;

while (temp != nullptr)

{

info\_for\_node(temp);

temp = temp->next;

}

}

node\* pop()

{

if (!tail) { return nullptr; }

node\* temp = tail;

if (head == tail)

{

head = nullptr;

tail = nullptr;

}

else

{

tail = tail->prev;

tail->next = nullptr;

}

temp->prev = nullptr;

return temp;

}

uint32\_t get\_max\_prior() const { return max\_prior; }

private:

node\* head;

node\* tail;

uint32\_t max\_prior;

};

node\* wrapper()

{

std::cout << "Enter info (string): ";

std::string buffer;

std::getline(std::cin, buffer);

uint32\_t prior;

std::cout << "Enter prior (integer): ";

if (!(std::cin >> prior))

{

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cerr << "Invalid input, please enter an integer for priority.\n";

return nullptr;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return new node(buffer, prior);

}

void delete\_node(node\* n)

{

delete n;

}

void info\_for\_node(node\* temp)

{

std::cout << "Info for node:\n";

std::cout << "Data: " << temp->data << "\n";

std::cout << "Priority: " << temp->priority << "\n";

}

void start()

{

priority\_queue pr\_q;

while (true)

{

std::cout << "Options:\n";

std::cout << "1. Append\n";

std::cout << "2. Pop\n";

std::cout << "3. Print Queue\n";

std::cout << "4. Exit\n";

std::cout << "Enter your choice: ";

uint16\_t choice;

if (!(std::cin >> choice))

{

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cerr << "Invalid choice, please enter a number from 1 to 4.\n";

continue;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

switch (choice)

{

case 1:

while (true)

{

std::cout << "Enter '~' to stop appending.\n";

node\* new\_node = wrapper();

if (!new\_node) break;

pr\_q.append(new\_node);

}

break;

case 2:

if (node\* popped = pr\_q.pop())

{

info\_for\_node(popped);

delete\_node(popped);

}

else

{

std::cout << "Queue is empty.\n\n";

}

break;

case 3:

pr\_q.printAll();

break;

case 4:

std::cout << "Exiting...\n";

return;

default:

std::cerr << "Invalid choice, please enter a number from 1 to 4.\n";

break;

}

}

}

int main()

{

start();

return 0;

}

Результат работы программы

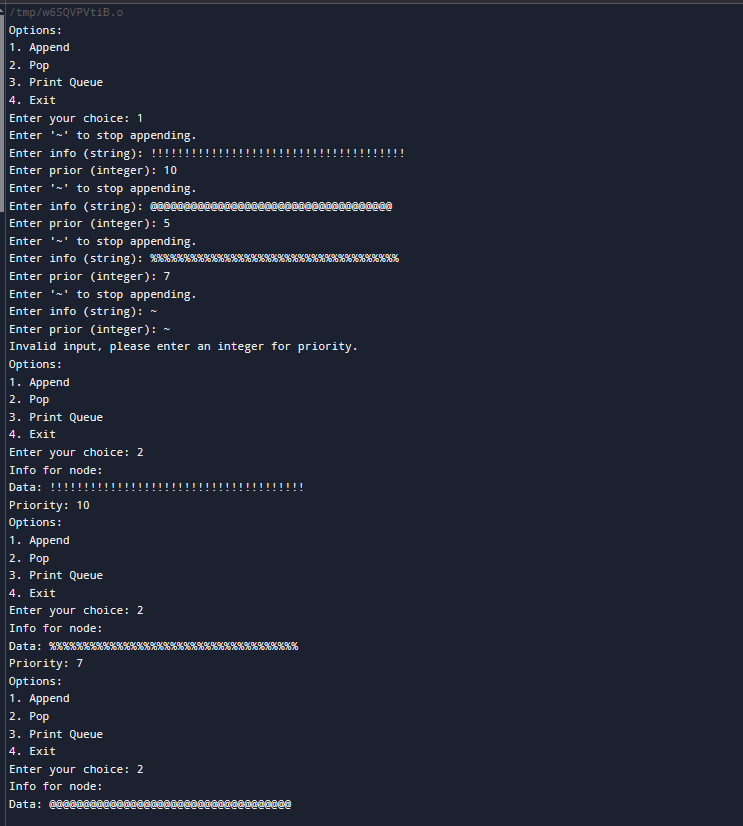


Рисунок № 1

**Задание 2. - очередь**

**Листинг**

#include <iostream>

#include <cstring>

#include <limits>

struct node

{

char inf[256];

struct node\* next;

};

struct node\* head = NULL, \* last = NULL, \* f = NULL; // указатели на первый и последний элементы списка

int dlinna = 0;

void spstore(void), review(void), del(char\* name);

char find\_el[256];

struct node\* find(char\* name); // функция нахождения элемента

struct node\* get\_struct(void); // функция создания элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (node\*)malloc(sizeof(struct node))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

printf("Введите название объекта: \n"); // вводим данные

scanf("%s", s);

if (\*s == 0)

{

printf("Запись не была произведена\n");

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

void spstore(void)

{

struct node\* p = NULL;

p = get\_struct();

if (head == NULL && p != NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

return struc;

}

struc = struc->next;

}

printf("Элемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

void del(char\* name)

{

struct node\* struc = head; // указатель, проходящий по списку установлен на начало списка

struct node\* prev;// указатель на предшествующий удаляемому элемент

int flag = 0; // индикатор отсутствия удаляемого элемента в списке

if (head == NULL) // если голова списка равна NULL, то список пуст

{

printf("Список пуст\n");

return;

}

if (strcmp(name, struc->inf) == 0) // если удаляемый элемент - первый

{

flag = 1;

head = struc->next; // установливаем голову на следующий элемент

free(struc); // удаляем первый элемент

struc = head; // устанавливаем указатель для продолжения поиска

}

else

{

prev = struc;

struc = struc->next;

}

while (struc) // проход по списку и поиск удаляемого элемента

{

if (strcmp(name, struc->inf) == 0) // если нашли, то

{

flag = 1; // выставляем индикатор

if (struc->next) // если найденный элемент не последний в списке

{

prev->next = struc->next; // меняем указатели

free(struc); // удаляем элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else // если найденный элемент последний в списке

{

prev->next = NULL; // обнуляем указатель предшествующего элемента

free(struc); // удаляем элемент

return;

}

}

else // если не нашли, то

{

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

if (flag == 0) // если флаг = 0, значит нужный элемент не найден

{

printf("Элемент не найден\n");

return;

}

}

struct node\* pop() {

if (!head) {

return nullptr;

}

if (!head->next) {

node\* temp = head;

head = nullptr;

return temp;

}

node\* temp = head;

while (temp->next->next != nullptr) {

temp = temp->next;

}

node\* last = temp->next;

temp->next = nullptr;

return last;

}

void queue()

{

while (true)

{

std::cout << "options: \n";

std::cout << "push: 1\n";

std::cout << "pop: 2\n";

std::cout << "exit: 3\n";

std::cout << "your choice: ";

int choice;

std::cin >> choice;

std::cout << std::endl;

if (choice == 1)

{

spstore();

}

else if (choice == 2)

{

struct node\* popped = pop();

if (popped == nullptr)

{

std::cout << "Deque is Empty\n\n";

}

else

{

std::cout << "Last block deleted. Name block - " << popped->inf << std::endl;

}

}

else if (choice == 3)

{

return;

}

else

{

std::cout << "Wrong operation\n\n";

}

}

}

int main()

{

queue();

exit(0);

}

**Результат работы программы**

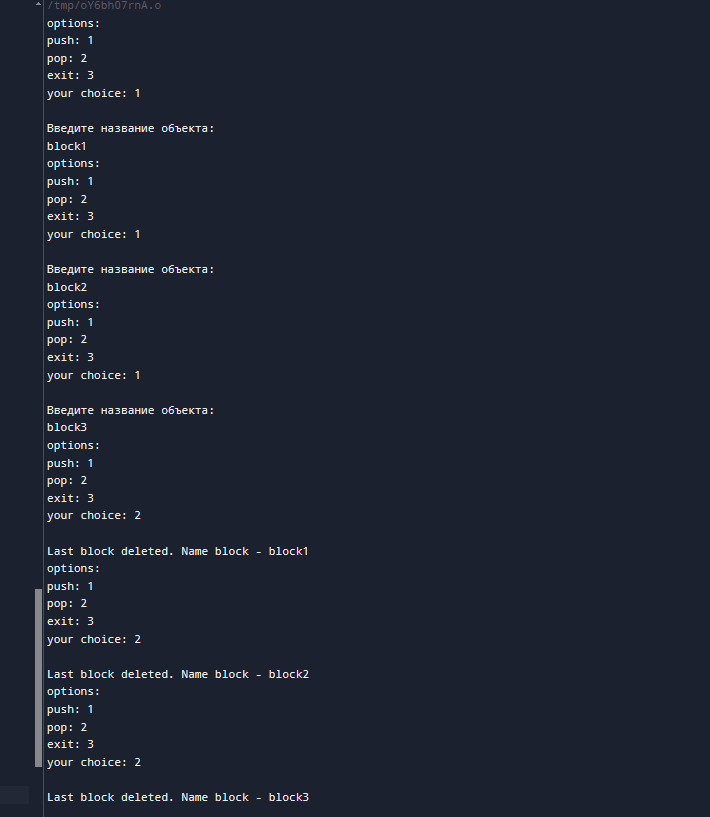


Рисунок 2)

**Задание 3. - stack**

**Листинг**

#include <iostream>

#include <cstring>

#include <limits>

struct node

{

char inf[256];

struct node\* next;

};

struct node\* head = NULL, \* last = NULL, \* f = NULL; // указатели на первый и последний элементы списка

int dlinna = 0;

void spstore(void), review(void), del(char\* name);

char find\_el[256];

struct node\* find(char\* name); // функция нахождения элемента

struct node\* get\_struct(void); // функция создания элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (node\*)malloc(sizeof(struct node))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

printf("Введите название объекта: \n"); // вводим данные

scanf("%s", s);

if (\*s == 0)

{

printf("Запись не была произведена\n");

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

void spstore(void)

{

struct node\* p = NULL;

p = get\_struct();

if (head == NULL && p != NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

last = p;

}

else if (head != NULL && p != NULL) // список уже есть, то вставляем в конец

{

last->next = p;

last = p;

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

return struc;

}

struc = struc->next;

}

printf("Элемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

void del(char\* name)

{

struct node\* struc = head; // указатель, проходящий по списку установлен на начало списка

struct node\* prev;// указатель на предшествующий удаляемому элемент

int flag = 0; // индикатор отсутствия удаляемого элемента в списке

if (head == NULL) // если голова списка равна NULL, то список пуст

{

printf("Список пуст\n");

return;

}

if (strcmp(name, struc->inf) == 0) // если удаляемый элемент - первый

{

flag = 1;

head = struc->next; // установливаем голову на следующий элемент

free(struc); // удаляем первый элемент

struc = head; // устанавливаем указатель для продолжения поиска

}

else

{

prev = struc;

struc = struc->next;

}

while (struc) // проход по списку и поиск удаляемого элемента

{

if (strcmp(name, struc->inf) == 0) // если нашли, то

{

flag = 1; // выставляем индикатор

if (struc->next) // если найденный элемент не последний в списке

{

prev->next = struc->next; // меняем указатели

free(struc); // удаляем элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else // если найденный элемент последний в списке

{

prev->next = NULL; // обнуляем указатель предшествующего элемента

free(struc); // удаляем элемент

return;

}

}

else // если не нашли, то

{

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

if (flag == 0) // если флаг = 0, значит нужный элемент не найден

{

printf("Элемент не найден\n");

return;

}

}

struct node\* pop() {

if (!head) {

return nullptr;

}

if (!head->next) {

node\* temp = head;

head = nullptr;

return temp;

}

node\* temp = head;

while (temp->next->next != nullptr) {

temp = temp->next;

}

node\* last = temp->next;

temp->next = nullptr;

return last;

}

void stack()

{

while (true)

{

std::cout << "options: \n";

std::cout << "push: 1\n";

std::cout << "pop: 2\n";

std::cout << "exit: 3\n";

std::cout << "your choice: ";

int choice;

std::cin >> choice;

std::cout << std::endl;

if (choice == 1)

{

spstore();

}

else if (choice == 2)

{

struct node\* popped = pop();

if (popped == nullptr)

{

std::cout << "Stack is Empty\n\n";

}

else

{

std::cout << "Last block deleted. Name block - " << popped->inf << std::endl;

}

}

else if (choice == 3)

{

return;

}

else

{

std::cout << "Wrong operation\n\n";

}

}

}

int main()

{

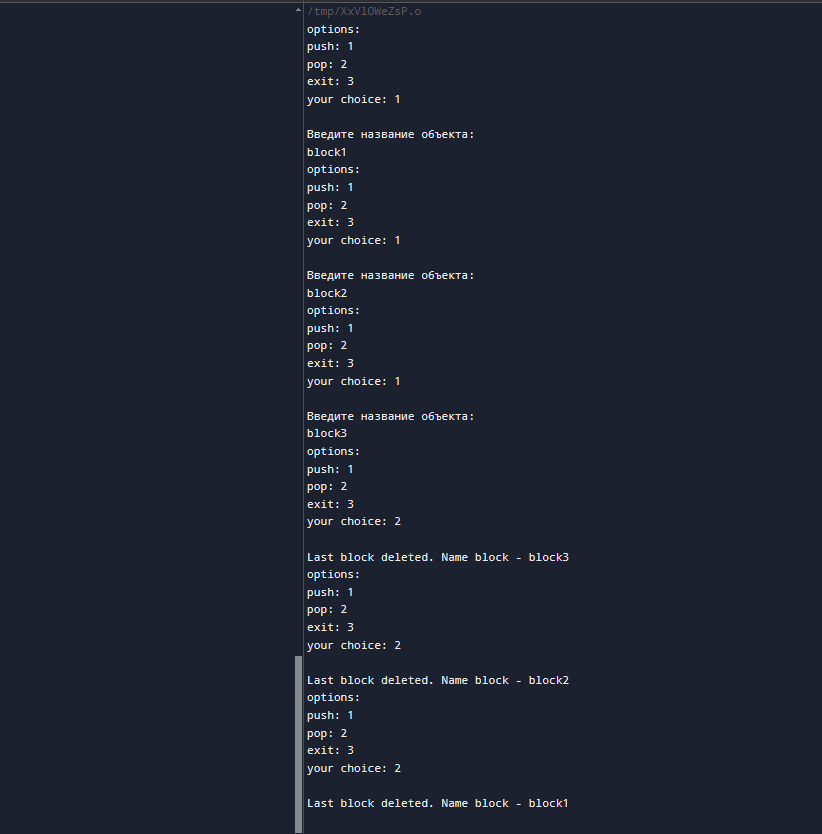
stack();

exit(0);

}

Результат работы программы

Рисунок 3)



**Вывод -** были получены навыки использования и реализации структур данных в основу которых входит связный список. - Приоритетная очередь, очередь, стек. А также методов поиска, добавления, удаления элементов.