

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных**  **технологий** | **Кафедра**  **информационных технологий и вычислительных систем** |

**КУРСОВАЯ РАБОТА**  
ПО ДИСЦИПЛИНЕ

«СТРУКТУРА И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТА | *2* | КУРСА | | *бакалавриата* | ГРУППЫ | *ИДБ-19-03* |
|  | | | *(уровень профессионального образования)* | |  | |

|  |
| --- |
| **Горожанкина Руслана Владиславовича** |
| *(ФИО)* |

|  |
| --- |
| Кольцевой двусвязный список слов |

|  |  |
| --- | --- |
| Направление: | 09.03.01 Информатика и вычислительная техника |
| Профиль подготовки: | Программное обеспечение средств вычислительной техники и автоматизированных систем |

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Лакунина О.Н., ст. преподаватель |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

МОСКВА 2020

Оглавление

Задание на курсовую работу……………………………………………………………2

Описание Структуры данных…………………………………......................................3

Конечная схема реализуемой структуры данных……………………………………..4

Список реализуемых функций для внешней структуры……………………………...5

Описание Структур на языке С…………………………………………………………6

Схема вызовов функций………………………………………………………………...7

Список функций и их назначение……………………………………………………....8

Исходный код программы с комментариями………………….....................................9

1

Задание на курсовую работу

Написать программу, реализующую логическую структуру данных – кольцевой двусвязный список слов. Программа должна работать в диалоговом режиме. Каждая операция должна быть реализована в отдельной функции.

Последовательность должна быть реализована на базе структуры хранения двусвязный список.

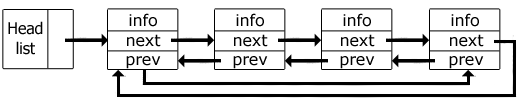
Написать отчет по курсовой работе.

2

Описание структуры данных

Двусвязный список - **это** структура данных, которая состоит из узлов, которые хранят полезные данные, указатели на предыдущий узел и следующий узел.

Кольцевой двусвязный список – это структура данных в которой, последний элемент ссылается на первый и первый ссылается на последний.

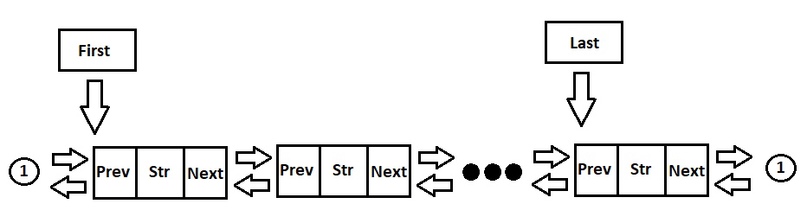


Указатель на предыдущий узел корня списка содержит адрес последнего узла. Указатель на следующий узел последнего узла содержит адрес корня списка.

|  |  |
| --- | --- |
| Head list | начало |
| info | данные |
| next | Переменная, хранящая указатель на следующий элемент списка |
| prev | Переменная, хранящая указатель на предыдущий элемент списка |

3

Конечная схема реализуемой структуры данных



|  |  |
| --- | --- |
| prev | Переменная, хранящая указатель на предыдущий элемент списка |
| next | Переменная, хранящая указатель на следующий элемент списка |
| str | Строка |
| First и last | Указатели на начало и конец |

4

Список реализуемых функций для внешней структуры

1. Сделать пустой список

2. Показать весь массив

3. Проверка на пустой список

4. Установить рабочий указатель в начало списка

5. Установить рабочий указатель в конец списка

6. Проверка указателя на местонахождение: в начале списка

7. Проверка указателя на местонахождение: в конце списка

8. Передвинуть указатель вперёд

9. Передвинуть указатель назад

12. Показать значение до указателя

13. Показать значение после указателя

14. Удалить элемент списка до указателя

15. Удалить элемент после указателя

16. Удалить элемент

17. Взять элемент структуры до указателя

18. Взять элемент структуры после указателя

19. Изменить элемент до указателя

20. Изменить элемент после указателя

21. Добавить элемент до указателя

22. Добавить элемент после указателя

23. Завершить работу с структурой

5

Описание структур на языке С

class List // Класс для списка

{

class Node // Структура для элемента множества

{

public: //спецификатор доступа

char str[80]; //массив элементов типа char

Node\* next; // указатель на следующий элемент

Node\* prev; //указатель на предыдущий элемент

};

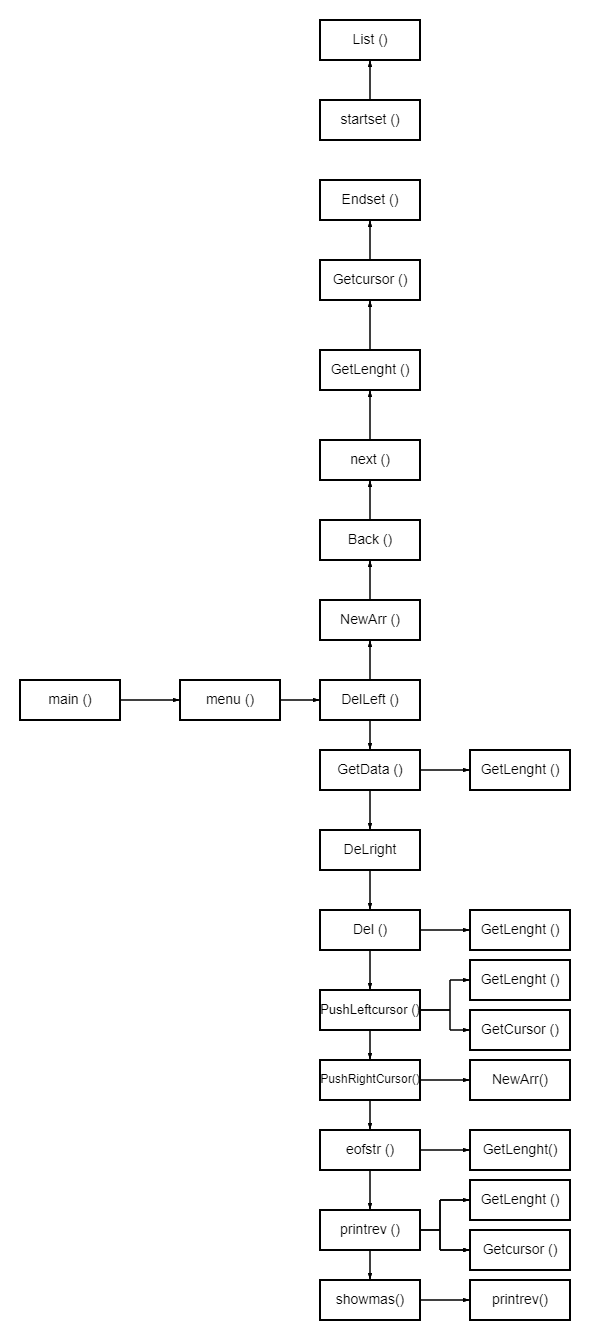
Node\* first; // Указатель на первый элемент

Node\* cursor; //Указатель на указатель в списке

Node\* last; //указатель на последний элемент

6

Схема вызовов функций



7

Список функций и их назначение

~List() и List() – констуктор и деструктор для обнуления списка

StartSet() – поставить указатель в начале

EndSet() – поставить указатель в конце

GetCursor() – на каком месте стоит указатель

GetLenght() – посчитать количества элементов

Next() – передвинуть указатель вперед

Back() – передвинуть указатель назад

NewArr() – создания первого элемента с списке

GetData() – вернуть необходимый элемент

DelRight() – удаление элемент после указателя справа

Del() - удаление элемента

DelLeft() – удаление элемента перед указателем слева

PushLeftCursor() – добавить элемент перед указателем

PushRightCursor() – добавить элемент после указателя

eofstr() – проверка на пустой список

printRev() – вывести элементы в обратном порядке

showMas() - вывести массив

menu() – вывод меню на экран

8

Исходный код программы с комментариями

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <ctime>

class List //Класс для списка

{

class Node// структура для элемента множества

{

public:

char str[80];

Node\* next; //указатель на следующий элемент

Node\* prev; //указатель на предыдщий элемент

};

Node\* first;//Указатель на первый элемент

Node\* cursor;//Указатель на указатель в списке

Node\* last;//указатель на последний элемент

public:

List() : last(NULL), first(NULL), cursor(NULL)// конструктор принимающие элементы

{}

~List()//деструктор для удаления обнуления списка

{

Node\* ptr = NULL, \* pdel;

ptr = first; // указатель на начало

while (ptr != last) // если указатель не в конце

{

pdel = ptr;

ptr = ptr->next;//сохраняем связь

free(pdel);

}

free(last);//очищаем память

}

void StartSet()//ставим указатель в начало списка

{

cursor = first;

}

void EndSet()//указатель в конце списка

{

cursor = last;

}

long GetCursor() //узнать местоположение указателя

{

Node\* ptr = first;

long pos = 0;

while (ptr != last && ptr != cursor)//пока указатель не в конце и не равень указателю

{

ptr = ptr->next; // двигаем

pos++; //считаем

}

return pos;

}

long GetLenght() //считаем количеста элементов в списке

{

Node\* ptr = first;

long lenght = 0;

if (!ptr) //проверка на пустой

return 0;

while (ptr != last) //пока указатель не в конце

{

ptr = ptr->next;//двигаем указатель

lenght++;//считаем

}

return lenght + 1;

}

void Next()//передвигаем указатель вперед

{

if (!cursor->next) //проверка стоит ли указатель в конце

printf("None\n");

else

{

cursor = cursor->next;//двигаем вперед

}

}

void Back()//передвигаем указатель назад

{

if (!cursor->prev) //проверка стоит ли указатель в начале

printf("None\n");

else

{

cursor = cursor->prev;//двигаем назад

}

}

void NewArr(char\* c) //содаем первый элемент в листе

{

Node\* ptr;

ptr = (Node\*)malloc(sizeof(Node)); //выделяем память под элемент

if (!ptr) //проверка на память

{

printf("Out of memory\n");

exit(1);

}

strcpy(ptr->str, c);

ptr->next = NULL;

ptr->prev = NULL;

first = last = cursor = ptr;

return;

}

char\* GetData(long count) //вернуть элемент

{

long i;

Node\* ptr;

if (count >= GetLenght())//проверка длинны

count -= GetLenght();

if (count < 0)//проверка длинны

count += GetLenght();

if (count > GetLenght()) //проверка на местонахождение (за пределами списка или нет)

{

printf("Out of bounds of an list\n");

exit(1);

}

if (GetLenght() / 2 > count)

{

i = 0;

ptr = first;//указатель ставим в начале

while (i != count && ptr)//пока i не равно переменной и не дошли до конца списка

{

ptr = ptr->next;//двигаем указатель вперед

i++;//считаем

}

}

else

{

i = GetLenght() - 1;//длинна минус 1

ptr = last;//указатель ставим в конце

while (i != count && ptr)

{

ptr = ptr->prev;//двигаем указатель назад

i--;//считаем

}

}

if (count == count && ptr)//проверка на местонахождение в списке

return ptr->str;

else

{

printf("Out of bounds of an array\n");

exit(1);

}

}

void DelRight()//удаляем элемент после указателя

{

long i;

Node\* ptr = cursor->next, \* prevptr = 0;

if (ptr)//если указатель равен нулю

prevptr = ptr->prev;

if (ptr == last)//если указатель в конце

last = prevptr;

if (ptr == first)//если указатель в начале

first = first->next;

if (prevptr)prevptr->next = ptr->next;//проверка на местонахождение и движение указателя вперед

if (ptr->next) // проверка на местонахождение

ptr->next->prev = prevptr;

if (prevptr && ptr->next == prevptr)//проверка на соответвие

{

prevptr->next = 0;//обнуляем сохраняя связь

prevptr->prev = 0;

}

free(ptr);//очищаем память

}

void Del()//удаление элемента

{

Node\* ptr = cursor, \* prevptr = 0;

if (ptr)//если указатель равен 0

prevptr = ptr->prev;

cursor = ptr->next;

if (ptr == last)//если указатель стоит на последнем месте

last = prevptr;

if (ptr == first)//если указатель в начале

first = first->next;

if (prevptr)//проверка на местонахождение

prevptr->next = ptr->next;

else

first = ptr->next;

if (ptr->next)//проверка на местонаходение указателя

ptr->next->prev = prevptr;

else

{

last = prevptr;

cursor = prevptr;

}

if (prevptr && ptr->next == prevptr)//проверка

{

prevptr->next = 0;//удаляем сохраняя связь

prevptr->prev = 0;

}

free(ptr);//освободлаем память

if (!GetLenght())//если длинна отлична от нуля

{

first = 0;//обнуление

last = 0;

cursor = 0;

}

}

void DelLeft()//удаляем элемент перед указателем

{

long i;

Node\* ptr = cursor->prev, \* prevptr = 0;

if (ptr)//проверка на местонахождение

prevptr = ptr->prev;

if (ptr == last)//находится ли указатель в конце

last = prevptr;

if (ptr == first)//находится ли указатель в начале

first = first->next;

if (prevptr)//проверка на местонахождение

prevptr->next = ptr->next;

else

first = ptr->next;

ptr->next->prev = prevptr;

if (prevptr && ptr->next == prevptr)//проверка

{

prevptr->next = 0;//удаление

prevptr->prev = 0;

}

free(ptr);//освобождение памяти

}

void PushLeftCursor(char\* ch)//добавить элемент перед указателем

{

long i = 0;

Node\* ptr = cursor;

if (!ptr)//проверка есть ли вообще элементы

{

NewArr(ch); // если нет создаем

return;

}

Node\* p = ptr->prev;

ptr->prev = (Node\*)malloc(sizeof(Node));//выделяем память

if (!ptr->prev)//проверка памяти

{

printf("Out of memory\n");

exit(1);

}

ptr->prev->next = ptr;//двигаем указатель

ptr = ptr->prev;

strcpy(ptr->str, ch);//копируем

ptr->prev = p;

if (p)//проверка на местонахождения в списке

{

p->next = ptr;//двигаем указатель

if (cursor == first)//проверяем указатель в начале или нет

{

first = ptr;//приравниваем

}

}

else

{

first->next = ptr;//меняем указатель

first->prev = ptr;

last = ptr;

last->prev = first;

}

if (GetCursor() == GetLenght())//если указатель в конце

last = ptr;//приравнивмем

}

void PushRightCursor(char\* ch)//добавить элемент после указателя

{

Node\* ptr = cursor;

if (!ptr)//проверяем пустой ли список

{

NewArr(ch);//если да создаем

return;

}

Node\* p = ptr->next;

ptr->next = 0;

ptr->next = (Node\*)malloc(sizeof(Node));//выделяем память

if (!ptr->next)//проверка памяти

{

printf("Out of memory\n");

exit(1);

}

ptr->next->prev = ptr;

ptr = ptr->next;

strcpy(ptr->str, ch);//копируем

ptr->next = p;

if (p)//проверка на местонахождение

{

p->prev = ptr;

if (cursor == last)//если указатель в конце

{

last = ptr;//приравниваем

}

}

else

{

first->next = ptr;//двигаем указатель

first->prev = ptr;

last = ptr;

last->next = first;

}

}

bool eofstr(long count = 0)//проверка на пустой список

{

if (GetLenght() > count)//если количества элементов больше чем 0

return 1;//есть элементы

return 0;//нет

}

void printRev()//выводим список в обратном порядке

{

Node\* ptr = last;

int i = GetLenght();//равно длине списка

do

{

i--;

printf("%s", ptr->str);

if (i == GetCursor())

printf(" <-");//выводим указатель

putchar('\n');

ptr = ptr->prev;

} while (ptr && ptr != last);//если мы не в конце

printf("\n");

}

void showMas()//выводим в прямом порядке список

{

Node\* ptr = first;

int i = 0;

if (!ptr)

return;

printf("\n Лист в прямом порядке:\n");

do

{

printf("%s", ptr->str);

if (i == GetCursor())

printf(" <-");

putchar('\n');

ptr = ptr->next;

i++;

} while (ptr && ptr != first);//если мы не в начале

printf("\n Лист в обратном порядке:\n");

printRev();

}

};

void menu(bool f)//выводим меню

{

printf("\n\n\*\*\*\*\n");

if (f)

{

printf("\

\r1. Сделать пустой список\n \

\r2. Показать весь массив\n \

\r3. Проверка на пустой список\n \

\r4. Установить рабочий указатель в начало списка\n \

\r5. Установить рабочий указатель в конец списка\n \

\r6. Проверка указателя на местонахождение : в начале списка\n \

\r7. Проверка указателя на местонахождение : в конце списка\n \

\r8. Передвинуть указатель вперёд\n \

\r9. Передвинуть указатель назад\n \

\r12. Показать значение до указателя\n \

\r13. Показать значение после указателя\n \

\r14. Удалить элемент списка до указателя\n \

\r15. Удалить элемент после указателя\n \

\r16. Удалить элемент \n \

\r17. Взять элемент структуры до указателя\n \

\r18. Взять элемент структуры после указателя\n \

\r19. Изменить элемент до указателя\n \

\r20. Изменить элемент после указателя\n \

\r21. Добавить элемент до указателя\n \

\r22. Добавить элемент после указателя\n \

\r23. Завершить работу с структурой\n");

}

else

printf("1. Начало работы со списком\n \

\r2. Завершить работу\n");

printf("\*\*\*\*\n\n");

}

void main()

{

setlocale(0, "RU");

bool f = 0;

char ch[80];

int choice;

List arr;

while (1)

{

menu(f);//вызываем меню

scanf("%s", ch);

choice = 0;

for (int i = 0; ch[i] != '\0' && isdigit(ch[i]); i++)//вводим значение для меню

choice = choice \* 10 + ch[i] - '0';

if (f)

{

switch (choice)

{

case 1:

arr.~List();//передаем список для обнуления

arr = List();

printf("\*Список обнулен\n\*");

break;

case 2:

arr.showMas();//выводим массив

break;

case 3:

if (arr.eofstr())//проверяем сколько элемнтов с списке

printf("В списке есть %d элементы\n", arr.GetLenght());

else

printf("cписок пуст\n");

arr.showMas();//выводим

break;

case 4:

arr.StartSet();//ставим указатель в начало

printf("Указатель в начале списка\n");

break;

case 5:

arr.EndSet();//ставим указатель в конец

printf("Указатель в конце списка\n");

break;

case 6:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

if (arr.GetCursor() == 0)//проверка на местонахождение в начале

printf("Указатель в начале\n");

else

printf("Указатель не в начале\n");

break;

case 7:

if (!arr.eofstr())

{

printf("список пуст\n");

break;

}

if (arr.GetCursor() == arr.GetLenght() - 1)//проверка на местонахождение в конце

printf("Указатель в конце\n");

else

printf("Указатель не в конце\n");

break;

case 8:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

arr.Next();//двигаем

arr.showMas();//выводим

break;

case 9:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

arr.Back();//назад

arr.showMas();//вывод

break;

case 12:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

printf("Значение перед указателем: %s", arr.GetData(arr.GetCursor() - 1));//выводим значение перед указателем

arr.showMas();//выводим список

break;

case 13:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

printf("Значение после указателем: %s", arr.GetData(arr.GetCursor() + 1));//выводим значение после указателя

arr.showMas();//вывод

break;

case 14:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

arr.DelLeft();//удаляем элемент перед указателем

printf("Элемент удален\n");

arr.showMas();//вывод

break;

case 15:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

arr.DelRight();//удаляем элемент после указателя

printf("Элемент удален\n");

arr.showMas();//вывод

break;

case 16:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

arr.Del();//удаляем элемент

printf("Элемент удален\n");

arr.showMas();//вывод

break;

case 17:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

strcpy(ch, arr.GetData(arr.GetCursor() - 1));//копируем значение и записываем его в переменную

arr.DelLeft();//удаляем

arr.showMas();//выводим

printf("Значение перед указателем: %s", ch);//выводим элемент который взяли

break;

case 18:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

strcpy(ch, arr.GetData(arr.GetCursor() + 1));//копируем значение и записываем его в переменную

arr.DelRight();//удаляем

arr.showMas();//выводим

printf("Значение после = указателем: %s", ch);

break;

case 19:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

printf("Введите значени\n");

scanf("%s", ch);

strcpy(arr.GetData(arr.GetCursor() - 1), ch);//копируем значение и записываем его в элемент списка

arr.showMas();//выводим

break;

case 20:

if (!arr.eofstr())//проверка на пустой список

{

printf("список пуст\n");

break;

}

printf("Введите значени\n");

scanf("%s", ch);

strcpy(arr.GetData(arr.GetCursor() + 1), ch);//копируем значение и записываем его в элемент списка

arr.showMas();//выводим

break;

case 21:

printf("Введите значени\n");

scanf("%s", ch);

arr.PushLeftCursor(ch);//добавляем элемент

arr.showMas();//вывод

break;

case 22:

printf("Введите значени\n");

scanf("%s", ch);

arr.PushRightCursor(ch);//добавление

arr.showMas();//вывод

break;

case 23:

arr.~List();//вызываем деструктор

f = !f;//хаканчиваем работу со структурой

break;

}

}

else

{

switch (choice)//если ввели другое значение

{

case 1:

f = !f;

arr = List();//начинаем работу

break;

case 2:

return;

}

}

}

}