Пензенский государственный университет

Кафедра «Вычислительная техника»

**Отчёт**

По лабораторной работе №3

По дисциплине: «[Логика и основы алгоритмизации в инженерных задачах](http://moodle.pnzgu.ru/course/view.php?id=48560)»

на тему: «Динамические списки»

*Выполнил студент группы 21ВВ2:*

*Зайкин С.А.*

*Приняли:*

*Митрохин М.А.*

*Юрова О.А.*

Пенза 2022

**Цель работы:**

Изучение организации списковых структур и построение реальных структур данных на базе списков.

**Лабораторные работы:**

**Задание:**

1. Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект  с большим приоритетом становится перед объектом с меньшим приоритетом).
2. \* На основе приведенного кода реализуйте структуру данных *Очередь*.
3. \* На основе приведенного кода реализуйте структуру данных *Стек*.

**Пояснительный текст к программе:**

Для выполнения лабораторной работы был выбран двухсвязный динамический список.

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

1) Для реализации списка нам понадобится структура узел

|  |
| --- |
| typedefstruct\_Node {      void\*value;      struct\_Node \*next;      struct\_Node \*prev;  } Node; |

Указатель prev хранит адрес предыдущего узла, если его нет (значит, это первый узел) то переменная равна NULL. Аналогично, указатель next хранит адрес следующего узла. Структура "Двусвязный Список" будет хранить свой размер (чтобы не пересчитывать количество элементов каждый раз), а также указатель head, ссылающийся на первый элемент, и указатель tail, ссылающийся на последний элемент:

|  |
| --- |
| typedefstruct\_DblLinkedList {      size\_tsize;      Node \*head;      Node \*tail;  } DblLinkedList; |

В случае, когда в списке нет элементов, оба они равны нулю. Если в списке один элемент, то оба указателя ссылаются на один и тот же элемент (соответственное, они равны). Об этом постоянно следует помнить: каждый раз, удаляя или вставляя элемент, придётся проверять, чтобы указатели head и tail правильно хранили адреса.

2) Первая функция, как обычно, создаёт экземпляр структуры:

DblLinkedList:

|  |
| --- |
| DblLinkedList\* createDblLinkedList() {      DblLinkedList \*tmp = (DblLinkedList\*) malloc(sizeof(DblLinkedList));      tmp->size = 0;      tmp->head = tmp->tail = NULL;      returntmp;  } |

Опишем функцию, которая удаляет список:

|  |
| --- |
| voiddeleteDblLinkedList(DblLinkedList \*\*list) {      Node \*tmp = (\*list)->head;      Node \*next = NULL;      while(tmp) {          next = tmp->next;          free(tmp);          tmp = next;      }      free(\*list);      (\*list) = NULL;  } |

3) Вставка спереди (используется в 1 задании). Сначала создаётся новый элемент:

|  |
| --- |
| Node \*tmp = (Node\*) malloc(sizeof(Node));  if(tmp == NULL) {      exit(1);  } |

Потом задаём ему значения:

|  |
| --- |
| tmp->value = data;  tmp->next = list->head;  tmp->prev = NULL; |

Так как он стал первым, то указатель next ссылается на старую голову списка, а предыдущего элемента нет. Теперь, если в списке уже был головной элемент, то его указатель prev должен ссылаться на вновь созданный элемент:

|  |
| --- |
| if(list->head) {      list->head->prev = tmp;  } |

Теперь проверим указатель tail. Если он пустой, то после добавления нового элемента он должен ссылаться на него:

|  |
| --- |
| if(list->tail == NULL) {      list->tail = tmp;  } |

Теперь перекинем указатель head на вновь созданный элемент и увеличим значение счётчика size:

list->head = tmp;

list->size++;

3) Удаление из начала списка. Прибавляются только перекидывания дополнительных указателей и проверка, чтобы указатель на последний элемент, в случае, если элементов больше не осталось, стал равным нулю.

Сначала создадим указатель на первый элемент списка. Он понадобится, чтобы после изменения всех указателей prev и next мы смогли удалить узел:

|  |
| --- |
| Node \*prev;  void\*tmp;  if(list->head == NULL) {      exit(2);  }  prev = list->head; |

После этого перекинем указатель head на следующий за ним элемент:

|  |
| --- |
| list->head = list->head->next; |

Далее проверяем, что удаляемый элемент не является одновременно последним (когда в списке всего один элемент), после чего освобождаем память.

4) Вставка в конец и удаление с конца очень похожи - просто мы переворачиваем список. Соответственное, все prev меняются на next, а head на tail.

void pushBack(DblLinkedList \*list, void \*value) {

    Node \*tmp = (Node\*) malloc(sizeof(Node));

    if (tmp == NULL) {

        exit(3);

    }

    tmp->value = value;

    tmp->next = NULL;

    tmp->prev = list->tail;

    if (list->tail) {

        list->tail->next = tmp;

    }

    list->tail = tmp;

    if (list->head == NULL) {

        list->head = tmp;

    }

    list->size++;

}

void\* popBack(DblLinkedList \*list) {

    Node \*next;

    void \*tmp;

    if (list->tail == NULL) {

        exit(4);

    }

    next = list->tail;

    list->tail = list->tail->prev;

    if (list->tail) {

        list->tail->next = NULL;

    }

    if (next == list->head) {

        list->head = NULL;

    }

    tmp = next->value;

    free(next);

    list->size--;

    return tmp;

}

5) Добавление с приоритетом в 1 задании работает по принципу: элементы добавляются сразу в нужном порядке. Для этого новый добавляемый элемент проходит проверку на размерность и получает свой номер в списке, в котором он должен располагаться:

Node\* tmp = list->head;

Node\* tmp\_prev = tmp;

int index = 0;

while (tmp!=NULL) {

if (value <= \*((int\*)(tmp->value))) {

tmp\_prev = tmp;

tmp = tmp->next;

index++;

}

else

break;

}

Если элемент заходит первым, то выполняется функция добавления в начало, если же последним, то добавление в конец, иначе мы получаем индекс, затем создаём новый узел и изменяем все указатели:

ins = (Node\*)malloc(sizeof(Node));

ins->value = Value;

ins->prev = tmp\_prev;

ins->next = tmp\_prev->next;

if (tmp\_prev->next) {

tmp\_prev->next->prev = ins;

}

tmp\_prev->next = ins;

if (!tmp\_prev->prev) {

list->head = tmp\_prev;

}

if (!tmp\_prev->next) {

list->tail = tmp\_prev;

}

6) Функция вывода списка на экран. На вход функции получаем наш список и проходимся по всему списку, выводя поочередно элементы:

void printDblLinkedList(DblLinkedList\* list) {

Node\* tmp = list->head;

while (tmp) {

printf("%d ", \*((int\*)tmp->value));

tmp = tmp->next;

}

printf("\n");

}

**Листинг:**

**1 задание:**

//очередь c приоритетом

#include <iostream>

using namespace std;

//реализация списка (двухсвязный)

typedef struct \_Node {

void\* value;

struct \_Node\* next; //хранит адрес следующего узла

struct \_Node\* prev; //хранит адрес предыдущего узла

} Node;

// структура элемента

typedef struct \_DblLinkedList {

size\_t size;

Node\* head;

Node\* tail;

//В случае, когда в списке нет элементов, оба они равны нулю

} DblLinkedList;

//cоздаёт экземпляр структуры DblLinkedList

DblLinkedList\* createDblLinkedList() {

DblLinkedList\* tmp = (DblLinkedList\*)malloc(sizeof(DblLinkedList));

tmp->size = 0;

tmp->head = tmp->tail = NULL;

return tmp;

}

//удаление списка

void deleteDblLinkedList(DblLinkedList\*\* list) {

Node\* tmp = (\*list)->head;

Node\* next = NULL;

while (tmp) {

next = tmp->next;

free(tmp);

tmp = next;

}

free(\*list);

(\*list) = NULL;

}

//добавление в конец

void pushBack(DblLinkedList\* list, void\* Value, int value) {

Node\* tmp = list->head;

Node\* tmp\_prev = tmp;

int index = 0;

while (tmp!=NULL) {

if (value <= \*((int\*)(tmp->value))) {

tmp\_prev = tmp;

tmp = tmp->next;

index++;

}

else

break;

}

if (index==0) {

Node\* ins = (Node\*)malloc(sizeof(Node));

if (ins == NULL) {

exit(1);

}

ins->value = Value;

ins->next = list->head;

ins->prev = NULL;

if (list->head) {

list->head->prev = ins;

}

list->head = ins;

if (list->tail == NULL) {

list->tail = ins;

}

list->size++;

}

else if (tmp==NULL) {

Node\* ins = (Node\*)malloc(sizeof(Node));

if (ins == NULL) {

exit(3);

}

ins->value = Value;

ins->next = NULL;

ins->prev = list->tail;

if (list->tail) {

list->tail->next = ins;

}

list->tail = ins;

if (list->head == NULL) {

list->head = ins;

}

list->size++;

}

else{

Node\* ins = NULL;

if (tmp\_prev == NULL) {

exit(5);

}

ins = (Node\*)malloc(sizeof(Node));

ins->value = Value;

ins->prev = tmp\_prev;

ins->next = tmp\_prev->next;

if (tmp\_prev->next) {

tmp\_prev->next->prev = ins;

}

tmp\_prev->next = ins;

if (!tmp\_prev->prev) {

list->head = tmp\_prev;

}

if (!tmp\_prev->next) {

list->tail = tmp\_prev;

}

list->size++;

}

}

//удаление первого элемента

void\* popFront(DblLinkedList\* list) {

Node\* prev;

void\* tmp;

if (list->head == NULL) {

exit(2);

}

prev = list->head;

list->head = list->head->next;

if (list->head) {

list->head->prev = NULL;

}

if (prev == list->tail) {

list->tail = NULL;

}

tmp = prev->value;

free(prev);

list->size--;

return tmp;

}

//вывод первого элемента

Node\* getN(DblLinkedList\* list) {

Node\* tmp = list->head;

return tmp;

}

//вывод списка

void printDblLinkedList(DblLinkedList\* list) {

Node\* tmp = list->head;

while (tmp) {

printf("%d ", \*((int\*)tmp->value));

tmp = tmp->next;

}

printf("\n");

}

int main()

{

setlocale(0, "");

DblLinkedList\* list = createDblLinkedList();

int a = 10, b = 30, c = 20, d = 50, e = 40;

pushBack(list, &a,a);

pushBack(list, &b,b);

pushBack(list, &c,c);

pushBack(list, &d,d);

pushBack(list, &e,e);

//вывод очереди

printDblLinkedList(list);

printf("Первый элемент %d\n", \*((int\*)(popFront(list))));

printf("Первый элемент %d\n", \*((int\*)(getN(list))->value));

popFront(list);

printf("Первый элемент %d\n", \*((int\*)(list->head->value)));

popFront(list);

deleteDblLinkedList(&list);

system("pause");

return 0;

}

**2 задание:**

//очередь

#include <iostream>

using namespace std;

//реализация списка (двухсвязный)

typedefstruct \_Node {

void\* value;

struct \_Node\* next; //хранит адрес следующего узла

struct \_Node\* prev; //хранит адрес предыдущего узла

} Node;

//структура элемента

typedefstruct \_DblLinkedList {

size\_t size;

Node\* head;

Node\* tail;

//В случае, когда в списке нет элементов, оба они равны нулю

} DblLinkedList;

//cоздаёт экземпляр структуры DblLinkedList

DblLinkedList\* createDblLinkedList() {

DblLinkedList\* tmp = (DblLinkedList\*)malloc(sizeof(DblLinkedList));

tmp->size = 0;

tmp->head = tmp->tail = NULL;

return tmp;

}

//удаление списка

void deleteDblLinkedList(DblLinkedList\*\* list) {

Node\* tmp = (\*list)->head;

Node\* next = NULL;

while (tmp) {

next = tmp->next;

free(tmp);

tmp = next;

}

free(\*list);

(\*list) = NULL;

}

//добавление в конец

void pushBack(DblLinkedList\* list, void\* value) {

Node\* tmp = (Node\*)malloc(sizeof(Node));

if (tmp == NULL) {

exit(3);

}

tmp->value = value;

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list->head = tmp;

}

list->size++;

}

//удаление первого элемента

void\* popFront(DblLinkedList\* list) {

Node\* prev;

void\* tmp;

if (list->head == NULL) {

exit(2);

}

prev = list->head;

list->head = list->head->next;

if (list->head) {

list->head->prev = NULL;

}

if (prev == list->tail) {

list->tail = NULL;

}

tmp = prev->value;

free(prev);

list->size--;

return tmp;

}

//вывод первого элемента

Node\* getN(DblLinkedList\* list) {

Node\* tmp = list->head;

return tmp;

}

//вывод списка

void printDblLinkedList(DblLinkedList\* list) {

Node\* tmp = list->head;

while (tmp) {

printf("%d ", \*((int\*)tmp->value));

tmp = tmp->next;

}

printf("\n");

}

int main()

{

setlocale(0, "");

DblLinkedList\* list = createDblLinkedList();

int a = 20, b = 10, c = 30, d = 50, e = 40;

pushBack(list, &a);

pushBack(list, &b);

pushBack(list, &c);

pushBack(list, &d);

pushBack(list, &e);

//вывод очереди

printDblLinkedList(list);

printf("Первый элемент %d\n", \*((int\*)(popFront(list))));

printf("Первый элемент %d\n", \*((int\*)(getN(list))->value));

popFront(list);

printf("Первый элемент %d\n", \*((int\*)(list->head->value)));

popFront(list);

deleteDblLinkedList(&list);

system("pause");

return 0;

}

**3 задание:**

//стек

#include <iostream>

using namespace std;

//реализация списка (двухсвязный)

typedefstruct \_Node {

void\* value;

struct \_Node\* next; //хранит адрес следующего узла

struct \_Node\* prev; //хранит адрес предыдущего узла

} Node;

// структура элемента

typedefstruct \_DblLinkedList {

size\_t size;

Node\* head;

Node\* tail;

//В случае, когда в списке нет элементов, оба они равны нулю

} DblLinkedList;

//cоздаёт экземпляр структуры DblLinkedList

DblLinkedList\* createDblLinkedList() {

DblLinkedList\* tmp = (DblLinkedList\*)malloc(sizeof(DblLinkedList));

tmp->size = 0;

tmp->head = tmp->tail = NULL;

return tmp;

}

//удаление списка

void deleteDblLinkedList(DblLinkedList\*\* list) {

Node\* tmp = (\*list)->head;

Node\* next = NULL;

while (tmp) {

next = tmp->next;

free(tmp);

tmp = next;

}

free(\*list);

(\*list) = NULL;

}

//добавление в конец

void pushBack(DblLinkedList\* list, void\* value) {

Node\* tmp = (Node\*)malloc(sizeof(Node));

if (tmp == NULL) {

exit(3);

}

tmp->value = value;

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list->head = tmp;

}

list->size++;

}

//удаление последнего элемента

void\* popBack(DblLinkedList\* list) {

Node\* next;

void\* tmp;

if (list->tail == NULL) {

exit(4);

}

next = list->tail;

list->tail = list->tail->prev;

if (list->tail) {

list->tail->next = NULL;

}

if (next == list->head) {

list->head = NULL;

}

tmp = next->value;

free(next);

list->size--;

return tmp;

}

//вывод последнего элемента

Node\* getN(DblLinkedList\* list) {

Node\* tmp = list->tail;

return tmp;

}

//вывод списка

void printDblLinkedList(DblLinkedList\* list) {

Node\* tmp = list->head;

while (tmp) {

printf("%d ", \*((int\*)tmp->value));

tmp = tmp->next;

}

printf("\n");

}

int main()

{

setlocale(0,"");

DblLinkedList\* list = createDblLinkedList();

int a = 20, b = 10, c = 30, d = 50, e = 40;

pushBack(list, &a);

pushBack(list, &b);

pushBack(list, &c);

pushBack(list, &d);

pushBack(list, &e);

//вывод очереди

printDblLinkedList(list);

printf("Первый элемент %d\n", \*((int\*)(popBack(list))));

printf("Последний элемент %d\n", \*((int\*)(getN(list))->value));

popBack(list);

printf("Последний элемент %d\n", \*((int\*)(list->tail->value))); //можно и без вызова функции

popBack(list);

deleteDblLinkedList(&list);

system("pause");

return 0;

}

**Вывод:** изучил организацию списковых структур и построение реальных структур данных на базе списков.