|  |  |
| --- | --- |
| + | **Отчёт по лабораторной работе** №25-26  по курсу 1 фундаментальная информатика  студента группы М8О-105Б-21 Бондаревой Елены Евгеньевны, № по списку 1  Контакты www, e-mail, icq, skype : lena\_bondareva\_03@mail.ru  Работа выполнена: «28» мая 2022г.  Преподаватель: каф.806 В.К.Титов  Входной контроль знаний с оценкой  Отчёт сдан, итоговая оценка  Подпись преподавателя |

1. **Тема**: «Автоматизация сборки программ модульной структуры на языке Си с использованием утилиты make. Абстрактные типы данных. Рекурсия. Модульное программирование на языке Си».
2. **Цель работы**: Применение быстрой сортировки Хоара к деку; знакомство с утилитой make.
3. **Задание (№3):** конкатенация двух очередей.
4. **Оборудование** (*лабораторное*):

ЭВМ -, процессор -, имя узла сети-с ОП -МБ

НМД -ГБ. Терминал - адрес -. Принтер -.

Другие устройства -.

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz , ОП 6 ГБ, НМД 240 ГБ. Монитор IPS 1920x1080

Другие устройства -.

1. **Программное обеспечение** (*лабораторное*):

Операционная система семейства -, наименование - версия -

Интерпретатор команд - версия -

Система программирования -версия -

Редактор текстов - версия -

Утилиты операционной системы -

Прикладные системы и программы -

Местонахождения и имена файлов программ и данных-

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства UNIX/GNU , наименование Ubuntu версия x86\_64

Интерпретатор команд bash

Редактор текстов emax

Утилиты операционной системы: head, du, grep, sum, tee, file, find, diff, tail, od, wc, cut, tar, touch, paste, uniq, gzip, sort, cmp, bzip2.

Прикладные системы и программы VTM-diagram

Местонахождения и имена файлов программ и данных -

1. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальное описание с пред- и постусловиями)

При конкатенации двух деков мы берём первый элемент второго дека и добавляем его в конец первого до тех пор, пока второй дек не станет пустым.

Реализацию дека выполним двумя методами: отображением на массив и отображением на динамические структуры.

Метод быстрой сортировки Хоара основывается на разбиении массива на 2 части относительно опорного элемента и рекурсивной сортировки каждой части массива.

Шаги алгоритма таковы:

1) Выбираем в массиве некоторый элемент, который будем называть опорным элементом.   
2) Операция разделения массива: реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него.

PushBack(D1,V) добавляет элемент V в конец D1, а т.к. все элементы D1 меньше V, то дек всё ещё остаётся упорядоченным. Cat(D1, D2) конкатенирует Деки D1 и D2, так как элементы D1 меньше элементов D2, то остаётся упорядоченность. Cat(D, D1) собирает наш изначальный дек заново в отсортированный дек.

1. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты, либо соображения по тестированию].

**udt\_deck.h:**

#ifndef UDT\_DECK

#define UDT\_DECK

#include<stdio.h>

#include<stdlib.h>

#define N 100

#define Tvalue int

struct Deck

{

int first, last, size;

Tvalue body[N];

};

void Init(Deck& D)

{

D.first = D.size = 0; D.last = N - 1;

}

int Empty(Deck D)

{

return D.size == 0;

}

void PushFront(Deck& D, Tvalue V)

{

if (D.size == N) printf("DECK IS OVERFLOW");

else { D.body[(D.first) ? --D.first : (D.first = N - 1)] = V; D.size++; }

}

void PushBack(Deck& D, Tvalue V)

{

if (D.size == N) printf("DECK IS OVERFLOW");

else { D.body[++D.last %= N] = V; D.size++; }

}

Tvalue PopFront(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

D.size--; int i = D.first++; D.first %= N;

return D.body[i];

} return -1;

}

Tvalue PopBack(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

D.size--; int i = D.last;

(D.last) ? D.last-- : (D.last = N - 1);

return D.body[i];

} return -1;

}

Tvalue Top(Deck D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else return D.body[D.first]; return -1;

}

int Size(Deck D) { return D.size; }

void Display(Deck D)

{

printf("[");

for (int i = D.first; i < D.first + D.size; i++)

printf("%d%s", D.body[i % N], (i < D.first + D.size - 1) ? "," : "");

printf("]\n");

}

void Cat(Deck& D1, Deck& D2)

{

while (!Empty(D2))

PushBack(D1, PopFront(D2));

}

void Append(Deck& D1, Deck& D2)

{

int k = Size(D1);

Cat(D1, D2);

for (int i = 0; i < k; i++)

PushBack(D1, PopFront(D1));

}

void QuickSort(Deck& D)

{

Deck D1, D2; Tvalue V, V1;

if (!Empty(D))

{

Init(D1); Init(D2);

V = PopFront(D);

while (!Empty(D))

{

if (Top(D) < V) PushBack(D1, PopFront(D));

else PushBack(D2, PopFront(D));

}

QuickSort(D1);

QuickSort(D2);

PushBack(D1, V);

Cat(D1, D2);

Cat(D, D1);

}

}

#endif

**DeckTask.cpp:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

#include "udt\_deck.h"

int main()

{

srand(time(0));

struct Deck D1, D2;

int k = 1, i, n;

Deck D;

Init(D);

Tvalue v;

while (k)

{

printf("\nChoose action:"

"\n1. Create Random Deck\n2. Print Deck\n3. Size of Deck"

"\n4. Insert in front of Deck"

"\n5. Insert in back of Deck"

"\n6. Delete from front of Deck"

"\n7. Delete from back of Deck"

"\n8. Sorting Deck\n9. Clear Deck"

"\n10. Special Action (Reverse of Deck)"

"\n11. Concatenation"

"\n0. Exit from program"

"\nInput number =>");

scanf("%d", &k);

if (k == 1)

{

printf("Input number of elements =>"); scanf("%d", &n);

for (i = 0; i < n; i++) { v = rand() % 10; PushBack(D, v); }

Display(D);

}

if (k == 2) { Display(D); }

if (k == 3) { n = Size(D); printf("\nSize of Deck =%d\n", n); }

if (k == 4) {

printf("\nInput value of element =>"); scanf("%d", &v);

PushFront(D, v);

}

if (k == 5) {

printf("\nInput value of element =>");

scanf("%d", &v);

PushBack(D, v);

}

if (k == 6) { v = PopFront(D); printf("\nElement =%d is deleted\n", v); }

if (k == 7) { v = PopBack(D); printf("\nElement =%d is deleted\n", v); }

if (k == 8) { QuickSort(D); Display(D); }

if (k == 9) Init(D);

if (k == 10) {

Init(D1);

n = Size(D);

for (i = 0; i < n; i++) PushBack(D1, PopBack(D));

// for(i=0;i<n;i++) PushFront(D1,PopFront(D));

Cat(D, D1); Display(D);

}

if (k == 11) {

Init(D1);

printf("Input number of elements =>"); scanf("%d", &n);

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

v = rand() % 10;

PushBack(D1, v);

}

Display(D1);

Cat(D, D1);

Display(D);

}

}

return 0;

}

**dyn\_deck.h:**

#ifndef DYN\_DECK

#define DYN\_DECK

#include<stdio.h>

#include<stdlib.h>

#define Tvalue int

struct Node {

Tvalue data;

Node\* next;

Node\* pred;

};

struct Deck

{

Node\* head;

Node\* tail;

};

void Init(Deck& D)

{

D.head = D.tail = NULL;

}

int Empty(Deck D)

{

return D.head == NULL;

}

void PushFront(Deck& D, Tvalue V)

{

Node\* p = new Node();

p->data = V;

p->next = NULL;

p->pred = NULL;

if (Empty(D)) {

D.head = p;

D.tail = p;

}

else {

p->next = D.head;

D.head->pred = p;

D.head = p;

}

}

void PushBack(Deck& D, Tvalue V)

{

Node\* p = new Node();

p->data = V;

p->next = NULL;

p->pred = NULL;

if (Empty(D)) {

D.head = p;

D.tail = p;

}

else {

D.tail->next = p;

p->pred = D.tail;

D.tail = p;

}

}

Tvalue PopFront(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

Node\* p = D.head;

Tvalue data = p->data;

D.head = D.head->next;

if (D.head != NULL)

D.head->pred = NULL;

else

D.tail = NULL;

delete p;

return data;

}

return -1;

}

Tvalue PopBack(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

Node\* p = D.tail;

Tvalue data = p->data;

D.tail = D.tail->pred;

if (D.tail != NULL)

D.tail->next = NULL;

else

D.head = NULL;

delete p;

return data;

}

return -1;

}

Tvalue Top(Deck D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else return D.head->data; return -1;

}

int Size(Deck D) {

int count = 0;

Node\* p = D.head;

while (p != NULL)

{

count++;

p = p->next;

}

return count;

}

void Display(Deck D)

{

printf("[");

Node\* p = D.head;

while (p != NULL)

{

printf("%d", p->data);

if (p->next != NULL) {

printf(",");

}

p = p->next;

}

printf("]\n");

}

void Cat(Deck& D1, Deck& D2)

{

while (!Empty(D2))

PushBack(D1, PopFront(D2));

}

void Append(Deck& D1, Deck& D2)

{

int k = Size(D1);

Cat(D1, D2);

for (int i = 0; i < k; i++)

PushBack(D1, PopFront(D1));

}

void QuickSort(Deck& D)

{

Deck D1, D2; Tvalue V, V1;

if (!Empty(D))

{

Init(D1); Init(D2);

V = PopFront(D);

while (!Empty(D))

{

if (Top(D) < V) PushBack(D1, PopFront(D));

else PushBack(D2, PopFront(D));

}

QuickSort(D1);

QuickSort(D2);

PushBack(D1, V);

Cat(D1, D2);

Cat(D, D1);

}

}

void Destroy(Deck& D)

{

Node\* p = D.head;

while (p != NULL)

{

Node\* t = p->next;

delete p;

p = t;

}

D.head = NULL;

D.tail = NULL;

}

#endif

**DeckTaskDynamic.cpp:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

#include "dyn\_deck.h"

int main()

{

srand(time(0));

struct Deck D1, D;

int k = 1, i, n;

Init(D);

Init(D1);

Tvalue v;

while (k)

{

printf("\nChoose action:"

"\n1. Create Random Deck\n2. Print Deck\n3. Size of Deck"

"\n4. Insert in front of Deck"

"\n5. Insert in back of Deck"

"\n6. Delete from front of Deck"

"\n7. Delete from back of Deck"

"\n8. Sorting Deck\n9. Clear Deck"

"\n10. Special Action (Reverse of Deck)"

"\n11. Concatenation"

"\n0. Exit from program"

"\nInput number =>");

scanf("%d", &k);

if (k == 1)

{

printf("Input number of elements =>");

scanf("%d", &n);

for (i = 0; i < n; i++) { v = rand() % 10; PushBack(D, v); }

Display(D);

}

if (k == 2) { Display(D); }

if (k == 3) { n = Size(D); printf("\nSize of Deck =%d\n", n); }

if (k == 4) {

printf("\nInput value of element =>");

scanf("%d", &v);

PushFront(D, v);

}

if (k == 5) {

printf("\nInput value of element =>");

scanf("%d", &v);

PushBack(D, v);

}

if (k == 6) { v = PopFront(D); printf("\nElement =%d is deleted\n", v); }

if (k == 7) { v = PopBack(D); printf("\nElement =%d is deleted\n", v); }

if (k == 8) { QuickSort(D); Display(D); }

if (k == 9) Destroy(D);

if (k == 10) {

Destroy(D1);

n = Size(D);

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

PushBack(D1, PopBack(D));

Cat(D, D1); Display(D);

}

if (k == 11) {

Destroy(D1);

printf("Input number of elements =>");

scanf("%d", &n);

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

v = rand() % 10;

PushBack(D1, v);

}

Display(D1);

Cat(D, D1);

Display(D);

}

}

Destroy(D);

Destroy(D1);

return 0;

}

*Пункты 1-7 отчёта составляются* ***строго до*** *начала лабораторной работы.*

Допущен к выполнению работы. Подпись преподавателя

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с текстовыми примерами, подписанный преподавателем)

elena@elena-Aspire-A315-53G:~$ cat tit.txt

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~ ~

~ Лабораторная работа № 25-26 ~

~ Модульное программирование на языке Си ~

~ ~

~ Бондарева Елена Евгеньевна ~

~ М8О-105Б-21 ~

~ ~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

elena@elena-Aspire-A315-53G:~$ cat udt\_deck.h

#ifndef UDT\_DECK

#define UDT\_DECK

#include<stdio.h>

#include<stdlib.h>

#define N 100

#define Tvalue int

struct Deck

{

int first, last, size;

Tvalue body[N];

};

void Init(Deck& D)

{

D.first = D.size = 0; D.last = N - 1;

}

int Empty(Deck D)

{

return D.size == 0;

}

void PushFront(Deck& D, Tvalue V)

{

if (D.size == N) printf("DECK IS OVERFLOW");

else { D.body[(D.first) ? --D.first : (D.first = N - 1)] = V; D.size++; }

}

void PushBack(Deck& D, Tvalue V)

{

if (D.size == N) printf("DECK IS OVERFLOW");

else { D.body[++D.last %= N] = V; D.size++; }

}

Tvalue PopFront(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

D.size--; int i = D.first++; D.first %= N;

return D.body[i];

} return -1;

}

Tvalue PopBack(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

D.size--; int i = D.last;

(D.last) ? D.last-- : (D.last = N - 1);

return D.body[i];

} return -1;

}

Tvalue Top(Deck D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else return D.body[D.first]; return -1;

}

int Size(Deck D) { return D.size; }

void Display(Deck D)

{

printf("[");

for (int i = D.first; i < D.first + D.size; i++)

printf("%d%s", D.body[i % N], (i < D.first + D.size - 1) ? "," : "");

printf("]\n");

}

void Cat(Deck& D1, Deck& D2)

{

while (!Empty(D2))

PushBack(D1, PopFront(D2));

}

void Append(Deck& D1, Deck& D2)

{

int k = Size(D1);

Cat(D1, D2);

for (int i = 0; i < k; i++)

PushBack(D1, PopFront(D1));

}

void QuickSort(Deck& D)

{

Deck D1, D2; Tvalue V, V1;

if (!Empty(D))

{

Init(D1); Init(D2);

V = PopFront(D);

while (!Empty(D))

{

if (Top(D) < V) PushBack(D1, PopFront(D));

else PushBack(D2, PopFront(D));

}

QuickSort(D1);

QuickSort(D2);

PushBack(D1, V);

Cat(D1, D2);

Cat(D, D1);

}

}

#endif

elena@elena-Aspire-A315-53G:~$ cat DeckTask.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

#include "udt\_deck.h"

int main()

{

srand(time(0));

struct Deck D1, D2;

int k = 1, i, n;

Deck D;

Init(D);

Tvalue v;

while (k)

{

printf("\nChoose action:"

"\n1. Create Random Deck\n2. Print Deck\n3. Size of Deck"

"\n4. Insert in front of Deck"

"\n5. Insert in back of Deck"

"\n6. Delete from front of Deck"

"\n7. Delete from back of Deck"

"\n8. Sorting Deck\n9. Clear Deck"

"\n10. Special Action (Reverse of Deck)"

"\n11. Concatenation"

"\n0. Exit from program"

"\nInput number =>");

scanf("%d", &k);

if (k == 1)

{

printf("Input number of elements =>"); scanf("%d", &n);

for (i = 0; i < n; i++) { v = rand() % 10; PushBack(D, v); }

Display(D);

}

if (k == 2) { Display(D); }

if (k == 3) { n = Size(D); printf("\nSize of Deck =%d\n", n); }

if (k == 4) {

printf("\nInput value of element =>"); scanf("%d", &v);

PushFront(D, v);

}

if (k == 5) {

printf("\nInput value of element =>");

scanf("%d", &v);

PushBack(D, v);

}

if (k == 6) { v = PopFront(D); printf("\nElement =%d is deleted\n", v); }

if (k == 7) { v = PopBack(D); printf("\nElement =%d is deleted\n", v); }

if (k == 8) { QuickSort(D); Display(D); }

if (k == 9) Init(D);

if (k == 10) {

Init(D1);

n = Size(D);

for (i = 0; i < n; i++) PushBack(D1, PopBack(D));

// for(i=0;i<n;i++) PushFront(D1,PopFront(D));

Cat(D, D1); Display(D);

}

if (k == 11) {

Init(D1);

printf("Input number of elements =>"); scanf("%d", &n);

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

v = rand() % 10;

PushBack(D1, v);

}

Display(D1);

Cat(D, D1);

Display(D);

}

}

return 0;

}

elena@elena-Aspire-A315-53G:~$ cat makefile1

DeckOnMassive.exe: DeckOnMassive.o

g++ -c -o Deck.o DeckTask.cpp

g++ -o udt\_deck.exe Deck.o

DeckOnMassive.o: udt\_deck.h

elena@elena-Aspire-A315-53G:~$ make -f makefile1

g++ -c -o Deck.o DeckTask.cpp

g++ -o udt\_deck.exe Deck.o

elena@elena-Aspire-A315-53G:~$ ./udt\_deck.exe

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>1

Input number of elements =>11

[8,3,3,6,0,2,1,5,6,2,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>4

Input value of element =>5

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>2

[5,8,3,3,6,0,2,1,5,6,2,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>7

Element =9 is deleted

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>6

Element =5 is deleted

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>5

Input value of element =>7

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>2

[8,3,3,6,0,2,1,5,6,2,7]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>8

[0,1,2,2,3,3,5,6,6,7,8]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>10

[8,7,6,6,5,3,3,2,2,1,0]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>11

Input number of elements =>5

[9,7,5,7,5]

[8,7,6,6,5,3,3,2,2,1,0,9,7,5,7,5]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>3

Size of Deck =16

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>8

[0,1,2,2,3,3,5,5,5,6,6,7,7,7,8,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>9

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>3

Size of Deck =0

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>0

elena@elena-Aspire-A315-53G:~$ cat dyn\_deck.h

#ifndef DYN\_DECK

#define DYN\_DECK

#include<stdio.h>

#include<stdlib.h>

#define Tvalue int

struct Node {

Tvalue data;

Node\* next;

Node\* pred;

};

struct Deck

{

Node\* head;

Node\* tail;

};

void Init(Deck& D)

{

D.head = D.tail = NULL;

}

int Empty(Deck D)

{

return D.head == NULL;

}

void PushFront(Deck& D, Tvalue V)

{

Node\* p = new Node();

p->data = V;

p->next = NULL;

p->pred = NULL;

if (Empty(D)) {

D.head = p;

D.tail = p;

}

else {

p->next = D.head;

D.head->pred = p;

D.head = p;

}

}

void PushBack(Deck& D, Tvalue V)

{

Node\* p = new Node();

p->data = V;

p->next = NULL;

p->pred = NULL;

if (Empty(D)) {

D.head = p;

D.tail = p;

}

else {

D.tail->next = p;

p->pred = D.tail;

D.tail = p;

}

}

Tvalue PopFront(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

Node\* p = D.head;

Tvalue data = p->data;

D.head = D.head->next;

if (D.head != NULL)

D.head->pred = NULL;

else

D.tail = NULL;

delete p;

return data;

}

return -1;

}

Tvalue PopBack(Deck& D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else {

Node\* p = D.tail;

Tvalue data = p->data;

D.tail = D.tail->pred;

if (D.tail != NULL)

D.tail->next = NULL;

else

D.head = NULL;

delete p;

return data;

}

return -1;

}

Tvalue Top(Deck D)

{

if (Empty(D)) printf("DECK IS EMPTY");

else return D.head->data; return -1;

}

int Size(Deck D) {

int count = 0;

Node\* p = D.head;

while (p != NULL)

{

count++;

p = p->next;

}

return count;

}

void Display(Deck D)

{

printf("[");

Node\* p = D.head;

while (p != NULL)

{

printf("%d", p->data);

if (p->next != NULL) {

printf(",");

}

p = p->next;

}

printf("]\n");

}

void Cat(Deck& D1, Deck& D2)

{

while (!Empty(D2))

PushBack(D1, PopFront(D2));

}

void Append(Deck& D1, Deck& D2)

{

int k = Size(D1);

Cat(D1, D2);

for (int i = 0; i < k; i++)

PushBack(D1, PopFront(D1));

}

void QuickSort(Deck& D)

{

Deck D1, D2; Tvalue V, V1;

if (!Empty(D))

{

Init(D1); Init(D2);

V = PopFront(D);

while (!Empty(D))

{

if (Top(D) < V) PushBack(D1, PopFront(D));

else PushBack(D2, PopFront(D));

}

QuickSort(D1);

QuickSort(D2);

PushBack(D1, V);

Cat(D1, D2);

Cat(D, D1);

}

}

void Destroy(Deck& D)

{

Node\* p = D.head;

while (p != NULL)

{

Node\* t = p->next;

delete p;

p = t;

}

D.head = NULL;

D.tail = NULL;

}

#endif

elena@elena-Aspire-A315-53G:~$ cat DeckTaskDynamic.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

#include "dyn\_deck.h"

int main()

{

srand(time(0));

struct Deck D1, D;

int k = 1, i, n;

Init(D);

Init(D1);

Tvalue v;

while (k)

{

printf("\nChoose action:"

"\n1. Create Random Deck\n2. Print Deck\n3. Size of Deck"

"\n4. Insert in front of Deck"

"\n5. Insert in back of Deck"

"\n6. Delete from front of Deck"

"\n7. Delete from back of Deck"

"\n8. Sorting Deck\n9. Clear Deck"

"\n10. Special Action (Reverse of Deck)"

"\n11. Concatenation"

"\n0. Exit from program"

"\nInput number =>");

scanf("%d", &k);

if (k == 1)

{

printf("Input number of elements =>");

scanf("%d", &n);

for (i = 0; i < n; i++) { v = rand() % 10; PushBack(D, v); }

Display(D);

}

if (k == 2) { Display(D); }

if (k == 3) { n = Size(D); printf("\nSize of Deck =%d\n", n); }

if (k == 4) {

printf("\nInput value of element =>");

scanf("%d", &v);

PushFront(D, v);

}

if (k == 5) {

printf("\nInput value of element =>");

scanf("%d", &v);

PushBack(D, v);

}

if (k == 6) { v = PopFront(D); printf("\nElement =%d is deleted\n", v); }

if (k == 7) { v = PopBack(D); printf("\nElement =%d is deleted\n", v); }

if (k == 8) { QuickSort(D); Display(D); }

if (k == 9) Destroy(D);

if (k == 10) {

Destroy(D1);

n = Size(D);

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

PushBack(D1, PopBack(D));

Cat(D, D1); Display(D);

}

if (k == 11) {

Destroy(D1);

printf("Input number of elements =>");

scanf("%d", &n);

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

v = rand() % 10;

PushBack(D1, v);

}

Display(D1);

Cat(D, D1);

Display(D);

}

}

Destroy(D);

Destroy(D1);

return 0;

}

elena@elena-Aspire-A315-53G:~$ cat makefile2

DeckOnDynamic.exe: DeckOnDynamic.o

g++ -c -o Deck.o DeckTaskDynamic.cpp

g++ -o dyn\_deck.exe Deck.o

DeckOnDynamic.o: dyn\_deck.h

elena@elena-Aspire-A315-53G:~$ make -f makefile2

g++ -c -o Deck.o DeckTaskDynamic.cpp

g++ -o dyn\_deck.exe Deck.o

elena@elena-Aspire-A315-53G:~$ ./dyn\_deck.exe

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>1

Input number of elements =>13

[2,0,6,1,7,2,6,6,9,3,4,9,2]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>10

[2,9,4,3,9,6,6,2,7,1,6,0,2]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>6

Element =2 is deleted

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>5

Input value of element =>7

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>2

[9,4,3,9,6,6,2,7,1,6,0,2,7]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>8

[0,1,2,2,3,4,6,6,6,7,7,9,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>11

Input number of elements =>5

[0,2,8,3,8]

[0,1,2,2,3,4,6,6,6,7,7,9,9,0,2,8,3,8]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>3

Size of Deck =18

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>8

[0,0,1,2,2,2,3,3,4,6,6,6,7,7,8,8,9,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>9

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>2

[]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>1

Input number of elements =>8

[9,3,3,6,8,3,2,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>11

Input number of elements =>3

[7,1,5]

[9,3,3,6,8,3,2,9,7,1,5]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>8

[1,2,3,3,3,5,6,7,8,9,9]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>10

[9,9,8,7,6,5,3,3,3,2,1]

Choose action:

1. Create Random Deck

2. Print Deck

3. Size of Deck

4. Insert in front of Deck

5. Insert in back of Deck

6. Delete from front of Deck

7. Delete from back of Deck

8. Sorting Deck

9. Clear Deck

10. Special Action (Reverse of Deck)

11. Concatenation

0. Exit from program

Input number =>0

**9. Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб.  или  дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
| 1 | дом | 27.05.  2022 | 20:00 | Поставила лишнюю скобку в одном из циклов | Внимательно писать код. |  |

**10**. Замечание автора по существу работы

**11.** Выводы:

В результате выполнения работы я изучила принцип работы утилиты make, составила и отладила определенный модуль реализации по заданной схеме модуля для абстрактного типа данных.

Недочеты, допущенные при выполнении задания, могут быть устранены следующим образом

Подпись студента