Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №3  
по дисциплине: «Динамические списки»

Выполнили студенты группы 22ВВВ2:  
Беляев Д.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Динамические списки

**Цель работы**

Научиться создавать и работать с динамическими списками

**Лабораторное 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*.

**Листинг**

* **list.h**

#ifndef LIST

#define LIST

typedef struct Node {

int value;

struct Node\* next;

} DataType;

int IsDataTypeNull(DataType\* head);

DataType\* CreateDataType(int data);

void Add(int data, DataType\* head);

DataType\* TryAddElementAt(int data, int pos, DataType\* head);

DataType\* TryRemoveElementAt(int pos, DataType\* head);

int\* TryCheckElementAt(int pos, DataType\* head);

void PrintDataType(DataType\* head);

int\* GetLength(DataType\* head);

#endif

* **prior\_queue.h**

#ifndef PRIOR\_QUEUE

#define PRIOR\_QUEUE

#include "list.h"

DataType\* CreatePrQueue(int data);

void EnqueuePr(int num, DataType\* prQueue);

DataType\* DequeuePr(DataType\* prQueue, int\*val);

#endif

* **queue.h**

#ifndef QUEUE

#define QUEUE

#include "list.h"

DataType\* CreateQueue(int data);

void Enqueue(int num, DataType\* prQueue);

DataType\* Dequeue(DataType\* prQueue, int\* val);

#endif

* **stack.h**

#ifndef STACK

#define STACK

#include "list.h"

DataType\* CreateStack(int data);

void Push(int num, DataType\* stack);

DataType\* Pop(DataType\* stack, int\* val);

#endif

* **list.c**

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

DataType\* CreateElement(int data)

{

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

tmp->value = data;

tmp->next = NULL;

return tmp;

}

int IsDataTypeNull(DataType\* head)

{

if (head == NULL)

return 1;

else

return 0;

}

DataType\* CreateDataType(int data)

{

return CreateElement(data);

}

void Add(int data, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return;

}

DataType\* tmp = CreateElement(data);

DataType\* l = head;

while (l -> next != NULL)

{

l = l->next;

}

l->next = tmp;

}

DataType\* TryAddElementAt(int data, int pos, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return head;

}

DataType\* tmp = CreateElement(data);

if (pos == 0)

{

DataType\* headEl = head;

tmp->next = headEl;

head = tmp;

return head;

}

DataType\* l = head;

for (int i = 0; i < pos - 1; i++)

{

if (l == NULL)

{

printf("ERROR: Index out of list\n");

return head;

}

l = l->next;

}

tmp->next = l->next;

l->next = tmp;

return head;

}

DataType\* TryRemoveElementAt(int pos, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return head;

}

if (pos == 0)

{

DataType\* headEl = head;

head = headEl->next;

free(headEl);

return head;

}

DataType\* l = head;

DataType\* tmp = NULL;

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

{

if (l == NULL || l->next == NULL)

{

printf("ERROR: Index out of list\n");

return head;

}

tmp = l;

l = l->next;

}

tmp->next = l->next;

free(l);

return head;

}

int\* TryCheckElementAt(int pos, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

DataType\* l = head;

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

{

if (l->next == NULL)

return NULL;

l = l->next;

}

return &(l->value);

}

void PrintDataType(DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return;

}

DataType\* l = head;

while (l->next != NULL)

{

printf("%d ", l->value);

l = l->next;

}

printf("%d\n", l->value);

}

int\* GetLength(DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

int count = 0;

DataType\* l = head;

while (l != NULL)

{

l = l->next;

count++;

}

return &count;

}

* **prqueue.c**

#include <stdio.h>

#include <stdlib.h>

#include "prior\_queue.h"

DataType\* CreatePrQueue(int data)

{

DataType\* prQueue = CreateDataType(data);

return prQueue;

}

void EnqueuePr(int num, DataType\* prQueue)

{

if (IsDataTypeNull(prQueue) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

Add(num, prQueue);

}

DataType\* DequeuePr(DataType\* prQueue, int \*val)

{

if (IsDataTypeNull(prQueue) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

int size = \*GetLength(prQueue);

int comparer = prQueue->value;

int maxPriorityElementPos = 0;

int maxPriorityElementValue = comparer;

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

{

int\* tmp = TryCheckElementAt(i, prQueue);

if (\*tmp > comparer)

{

maxPriorityElementPos = i;

maxPriorityElementValue = \*tmp;

comparer = maxPriorityElementValue;

}

}

prQueue = TryRemoveElementAt(maxPriorityElementPos, prQueue);

\*val = maxPriorityElementValue;

return prQueue;

}

* **queue.c**

#include <stdio.h>

#include <stdlib.h>

#include "prior\_queue.h"

DataType\* CreateQueue(int data)

{

DataType\* queue = CreateDataType(data);

return queue;

}

void Enqueue(int num, DataType\* queue)

{

if (IsDataTypeNull(queue) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

Add(num, queue);

}

DataType\* Dequeue(DataType\* queue, int\* val)

{

if (IsDataTypeNull(queue) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

\*val = \*TryCheckElementAt(0, queue);

queue = TryRemoveElementAt(0, queue);

return queue;

}

* **stack.c**

#include <stdio.h>

#include <stdlib.h>

#include "stack.h"

DataType\* CreateStack(int data)

{

DataType\* stack = CreateDataType(data);

return stack;

}

void Push(int num, DataType\* stack)

{

if (IsDataTypeNull(stack) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

Add(num, stack);

}

DataType\* Pop(DataType\* stack, int\* val)

{

if (IsDataTypeNull(stack) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

int lastElementPosition = \*GetLength(stack) - 1;

\*val = \*TryCheckElementAt(lastElementPosition, stack);

stack = TryRemoveElementAt(lastElementPosition, stack);

return stack;

}

* **main.c**

#include <stdio.h>

#include "list.h"

#include "prior\_queue.h"

#include "queue.h"

#include "stack.h"

int main()

{

#pragma region list

printf("---- List part ----\n");

DataType\* l = CreateDataType(1);

Add(2, l);

Add(3, l);

PrintDataType(l);

l = TryAddElementAt(0, 0, l);

l = TryAddElementAt(1, 10, l);

PrintDataType(l);

printf("%d\n", \*TryCheckElementAt(1, l));

l = TryRemoveElementAt(0, l);

PrintDataType(l);

printf("%d\n", \*GetLength(l));

l = TryRemoveElementAt(0, l);

l = TryRemoveElementAt(0, l);

l = TryRemoveElementAt(0, l);

l = TryRemoveElementAt(0, l);

#pragma endregion

#pragma region prior queue

printf("\n---- Prior queue part ----\n");

DataType\* pQ = CreatePrQueue(1);

EnqueuePr(20, pQ);

EnqueuePr(10, pQ);

EnqueuePr(100, pQ);

int valPQ = 0;

pQ = DequeuePr(pQ, &valPQ);

printf("%d\n", valPQ);

pQ = DequeuePr(pQ, &valPQ);

printf("%d\n", valPQ);

pQ = DequeuePr(pQ, &valPQ);

printf("%d\n", valPQ);

pQ = DequeuePr(pQ, &valPQ);

printf("%d\n", valPQ);

pQ = DequeuePr(pQ, &valPQ);

pQ = DequeuePr(pQ, &valPQ);

#pragma endregion

#pragma region queue

printf("\n---- Queue part ----\n");

DataType\* q = CreateQueue(1);

Enqueue(20, q);

Enqueue(10, q);

Enqueue(100, q);

int valQ = 0;

q = Dequeue(q, &valQ);

printf("%d\n", valQ);

q = Dequeue(q, &valQ);

printf("%d\n", valQ);

q = Dequeue(q, &valQ);

printf("%d\n", valQ);

q = Dequeue(q, &valQ);

printf("%d\n", valQ);

q = Dequeue(q, &valQ);

q = Dequeue(q, &valQ);

#pragma endregion

#pragma region stack

printf("\n---- Stack part ----\n");

DataType\* s = CreateStack(1);

Push(20, s);

Push(10, s);

Push(100, s);

int valS = 0;

s = Pop(s, &valS);

printf("%d\n", valS);

s = Pop(s, &valS);

printf("%d\n", valS);

s = Pop(s, &valS);

printf("%d\n", valS);

s = Pop(s, &valS);

printf("%d\n", valS);

s = Pop(s, &valS);

s = Pop(s, &valS);

#pragma endregion

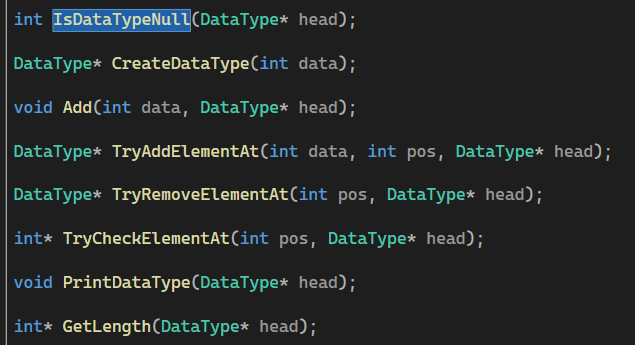
return 0;

}

**Задания**

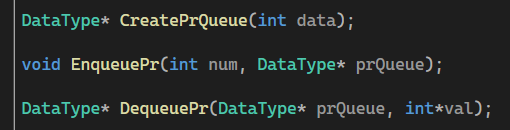
Задание 0

Реализовал динамический список. Методы: проверка на Null, создание списка, добавление в список элемента, добавление и удаление элемента на опр. позиции, просмотр значения элемента на опр. позиции, вывод массива и просмотр длины массива.



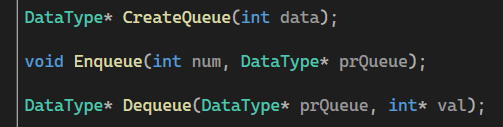
Задание 1

Реализовал приоритетную очередь. Методы: создание приоритетной очереди, добавление и удаление элемента из приоритетной очереди. Особенность данной очереди – забирается не первый элемент из очереди, а элемент с наибольшим приоритетом.



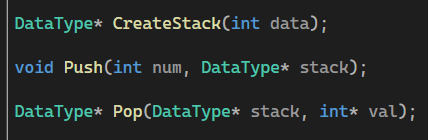
Задание 2

Реализовал очередь. Методы: создание очереди, добавление и удаление элемента из очереди. Особенность – первый вошедший в очередь элемент, выйдет первым

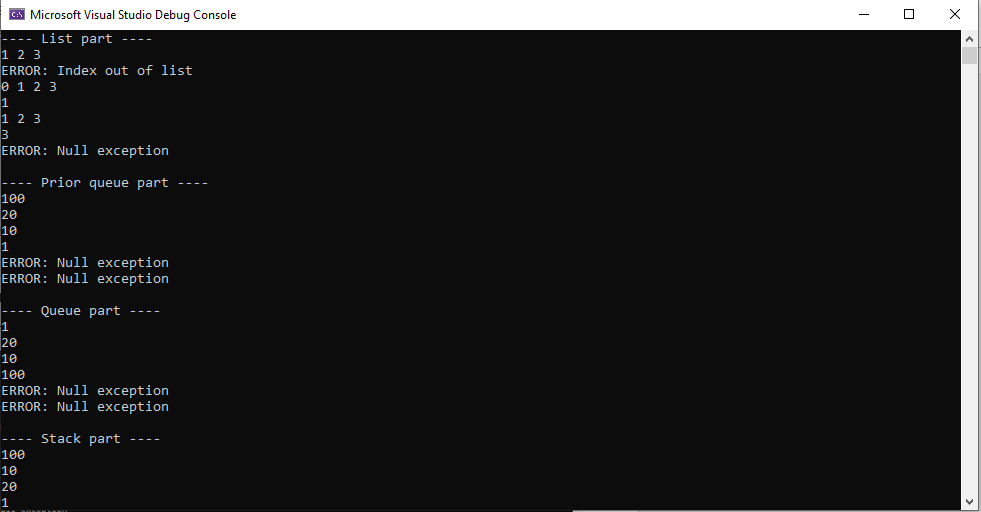


Задание 3

Реализовал стек. Методы: создание стека, добавление и удаление элемент из стека. Особенность – последний вошедший в стек элемент, выйдет первым



**Результат работы программы**

****

**Вывод**

Я научился создавать и работать с динамическими списками