МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 6**

по дисциплине: Алгоритмы и структуры данных

тема: ««Структуры данных «стек» и «очередь» C»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

асс. Солонченко Роман Евгеньевич

Белгород 2023г.

**Лабораторная работа №6**

**««Структуры данных «стек» и «очередь» C»**

**Цель работы:** изучить СД типа «стек» и «очередь», научиться их программно реализовать и использовать.**Содержание отчета:**

1. Тема лабораторной работы.

2. Цель работы.

3. Характеристика СД типа «стек» и «очередь» (п.1 задания).

4. Индивидуальное задание.

5. Текст модуля для реализации СД типа «линейный список», текст программы для отладки модуля, тестовые данные результат работы программы.

6. Текст программы для решения задачи с использованием модуля, тестовые данные, результат работы программы.**Задание к лабораторной работе :**

1. Для СД типа «стек» и «очередь» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

1.1.2. Набор допустимых операций.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД. Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «стек» и «очередь» в соответствии с вариантом индивидуального задания в виде модуля.

3. Разработать программу, моделирующую вычислительную систему с постоянным шагом по времени (дискретное время) в соответствии с вариантом индивидуального задания (табл.16) с использованием модуля, полученного в результате выполнения пункта 2. Результат работы программы представить в виде таблицы 15. В первом столбце указывается время моделирования 0, 1, 2, …, N. Во втором — для каждого момента времени указываются имена объектов (очереди — F1, F2, …, FN; стеки — S1, S2, …, SM; процессоры — P1, P2, …, PK), а в третьем — задачи (имя, время), находящиеся в объектах.

Выполнение заданий:  
1. Для СД типа «стек» и «очередь» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

Для стека: линейная структура - последовательность.

Для очереди: линейная структура – последовательность.

1.1.2. Набор допустимых операций.

Для стека: инициализация, включение, исключение, чтение, проверка пустоты, уничтожение.

Для очереди: инициализация, включение, исключение, проверка пустоты, уничтожение.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

Для стека: последовательная или связная.

Для очереди: последовательная или связная.

1.2.2. Объем памяти, занимаемый экземпляром СД.

Для стека: зависит от максимального числа элементов в стеке

Для очереди: зависит от максимального числа элементов в очереди

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

Для стека: может использоваться как статический и динамический массив, или реализованный на них список.

Для очереди: аналогично стеку.

1.2.4. Характеристику допустимых значений.

Для стека: CAR(стек) = CAR(BaseType)0 + CAR(BaseType)1 + … + CAR(BaseType)max,

Для очереди: CAR(FIFO) = CAR(BaseType)0 + CAR(BaseType)1 + … + CAR(BaseType)max

1.2.5. Тип доступа к элементам.

Для стека: в зависимости от СД, на котором реализован.

Для очереди: в зависимости от СД, на котором реализована.

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

**Стек**

const short StackSize = 100;

typedef int BaseType;

typdef struct {

BaseType Buf [StackSize];

unsigned uk;

} Stack;

**Очередь**

const short FifoSize = 100;

typedef int BaseType;

typdef struct {

BaseType Buf [FifoSize];

unsigned uk1;

unsigned uk2;

unsigned n;

} Fifo;

Вариант 4

2. Реализовать СД типа «стек» и «очередь» в соответствии с вариантом индивидуального задания в виде модуля.

Файл list.h

#ifndef ALGORITHMS\_AND\_DATA\_STRUCTURES\_LIST\_H  
#define ALGORITHMS\_AND\_DATA\_STRUCTURES\_LIST\_H  
  
extern const short ListOk;  
extern const short ListEmpty;  
extern const short ListNotMem;  
extern const short ListEnd;  
  
extern short ListError;  
  
typedef struct{  
 int data;  
 unsigned time;  
 int p;  
} TInquiry;  
  
  
typedef TInquiry BaseType;  
typedef struct element\* elptr;  
  
typedef struct element  
{  
 BaseType data;  
 elptr next;  
};  
  
typedef struct  
{  
 elptr start;  
 elptr ptr;  
} List;  
  
void InitList(List\* l);  
void PutList(List\* l, BaseType e);  
void GetList(List\* l, BaseType\* e);  
short EmptyList(List\* l);  
void BeginListPtr(List\* l);  
void EndListPtr(List\* l);  
void MovePtr(List\* l);  
void DoneList(List\* l);  
  
#endif //ALGORITHMS\_AND\_DATA\_STRUCTURES\_LIST\_H

Файл list.c

#include <stdlib.h>  
#include "list.h"  
  
const short ListOk = 0;  
const short ListEmpty = 1;  
const short ListNotMem = 2;  
const short ListEnd = 4;  
  
short ListError;  
  
void InitList(List\* l)  
{  
 elptr pntr = (elptr) malloc(sizeof(struct element));  
  
 if(pntr != NULL)  
 {  
 pntr->next = NULL;  
 }  
 else  
 {  
 ListError = ListNotMem;  
 return;  
 }  
  
 l->start = pntr;  
 l->ptr = pntr;  
 ListError = ListOk;  
}  
  
void PutList(List\* l, BaseType e)  
{  
 elptr pntr = (elptr)malloc(sizeof(struct element));  
  
 if (pntr != NULL)  
 {  
 pntr->data = e;  
 pntr->next = NULL;  
 }  
 else  
 {  
 ListError = ListNotMem;  
 return;  
 }  
  
 pntr->next = l->ptr->next;  
 l->ptr->next = pntr;  
 ListError = ListOk;  
}  
  
void GetList(List\* l, BaseType\* e)  
{  
 if (l->start != l->ptr && l->ptr->next == NULL)  
 {  
  
 ListError = ListEnd;  
 }  
 else if (EmptyList(l))  
 {  
 ListError = ListEmpty;  
 }  
 else  
 {  
 elptr pntr = l->ptr->next;  
 \*e = pntr->data;  
 l->ptr->next = pntr->next;  
 free(pntr);  
 ListError = ListOk;  
 }  
}  
  
short EmptyList(List\* l)  
{  
 return l->start->next == NULL;  
}  
  
void BeginListPtr(List\* l)  
{  
 if (EmptyList(l))  
 ListError = ListEmpty;  
 else  
 {  
 l->ptr = l->start;  
 ListError = ListOk;  
 }  
}  
  
void EndListPtr(List\* l)  
{  
 if (EmptyList(l))  
 ListError = ListEmpty;  
 else  
 {  
 BeginListPtr(l);  
  
 while (l->ptr->next != NULL)  
 MovePtr(l);  
 }  
}  
  
void MovePtr(List\* l)  
{  
 if (EmptyList(l))  
 ListError = ListEmpty;  
 else if (l->ptr->next == NULL)  
 ListError = ListEnd;  
 else  
 {  
 l->ptr = l->ptr->next;  
 ListError = ListOk;  
 }  
}  
  
void DoneList(List\* l)  
{  
 BeginListPtr(l);  
  
 while (!EmptyList(l))  
 {  
 BaseType null;  
  
 GetList(l, &null);  
 }  
  
 free(l->start);  
}

Файл stack.h

#ifndef ALGORITHMS\_AND\_DATA\_STRUCTURES\_STACK\_H  
#define ALGORITHMS\_AND\_DATA\_STRUCTURES\_STACK\_H  
  
#include "list.h"  
  
extern const short StackOk; // ListOk  
extern const short StackEmpty; // ListEmpty  
extern const short StackNotMem; // ListNotMem  
  
extern short StackError;  
  
typedef List Stack;  
  
void InitStack(Stack\* s);  
void PutStack(Stack\* s, BaseType e);  
void GetStack(Stack\* s, BaseType\* e);  
short EmptyStack(Stack\* s);  
void DoneStack(Stack\* s);  
  
#endif //ALGORITHMS\_AND\_DATA\_STRUCTURES\_STACK\_H

Файл stack.c

#include <stdio.h>  
#include <stdlib.h>  
#include "stack.h"  
  
const short StackOk = 0; // ListOk  
const short StackEmpty = 1; // ListEmpty  
const short StackNotMem = 2; // ListNotMem  
  
short StackError;  
  
void InitStack(Stack\* s)  
{  
 InitList(s);  
 StackError = ListError;  
}  
  
void PutStack(Stack\* s, BaseType e)  
{  
 PutList(s, e);  
 StackError = ListError;  
}  
  
void GetStack(Stack\* s, BaseType\* e)  
{  
 GetList(s, e);  
 StackError = ListError;  
}  
  
short EmptyStack(Stack\* s)  
{  
 return EmptyList(s);  
}  
  
void DoneStack(Stack\* s)  
{  
 DoneList(s);  
 StackError = ListError;  
}

Файл fifo.h

#ifndef ALGORITHMS\_AND\_DATA\_STRUCTURES\_FIFO\_H  
#define ALGORITHMS\_AND\_DATA\_STRUCTURES\_FIFO\_H  
  
#include "list.h"  
  
extern const short FifoOk; // ListOk  
extern const short FifoEmpty; // ListEmpty  
extern const short FifoNotMem; // ListNotMem  
  
extern short FifoError;  
  
typedef List Fifo;  
  
void InitFifo(Fifo\* f);  
void PutFifo(Fifo\* f, BaseType e);  
void GetFifo(Fifo\* f, BaseType\* e);  
short EmptyFifo(Fifo\* f);  
void DoneFifo(Fifo\* f);  
  
#endif //ALGORITHMS\_AND\_DATA\_STRUCTURES\_FIFO\_H

Файл fifo.c

#include "fifo.h"  
  
const short FifoOk = 0; // ListOk  
const short FifoEmpty = 1; // ListEmpty  
const short FifoNotMem = 2; // ListNotMem  
  
short FifoError;  
  
void InitFifo(Fifo\* f)  
{  
 InitList(f);  
 FifoError = ListError;  
}  
  
void PutFifo(Fifo\* f, BaseType e)  
{  
 EndListPtr(f);  
 PutList(f, e);  
 FifoError = ListError;  
}  
  
void GetFifo(Fifo\* f, BaseType\* e)  
{  
 BeginListPtr(f);  
 GetList(f, e);  
 FifoError = ListError;  
}  
  
short EmptyFifo(Fifo\* f)  
{  
 return EmptyList(f);  
}  
  
void DoneFifo(Fifo\* f)  
{  
 DoneList(f);  
 FifoError = ListError;  
}

3. Разработать программу, моделирующую вычислительную систему с постоянным шагом по времени (дискретное время) в соответствии с вари-антом индивидуального задания (табл.16) с использованием модуля, полученного в результате выполнения пункта 2. Результат работы программы представить в виде таблицы 15. В первом столбце указывается время моделирования 0, 1, 2, …, N. Во втором — для каждого момента времени указываются имена объектов (очереди — F1, F2, …, FN; стеки — S1, S2, …, SM; процессоры — P1, P2, …, PK), а в третьем — задачи (имя, время), находящиеся в объектах.

Файл main.c

#include <stdio.h>  
#include <malloc.h>  
#include <windows.h>  
#include "../../libs/alg/labs/lab6/fifo.h"  
#include "../../libs/alg/labs/lab6/stack.h"  
  
  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 Fifo\* f1 = (Fifo\*) malloc(sizeof(Fifo));  
 Fifo\* f2 = (Fifo\*) malloc(sizeof(Fifo));  
 Fifo\* f3 = (Fifo\*) malloc(sizeof(Fifo));  
  
 Stack\* s1 = (Stack\*) malloc(sizeof(Stack));  
 Stack\* s2 = (Stack\*) malloc(sizeof(Stack));  
  
 InitFifo(f1);  
 InitFifo(f2);  
 InitFifo(f3);  
 InitStack(s1);  
 InitStack(s2);  
 short p1 = 0, p2 = 0;  
  
 int k1 = 0, k2 = 0;  
  
 BaseType q, q1, q2;  
  
 int k = 0, n = 1;  
  
 int f = 0;  
  
 printf("Введите запрос: ");  
 scanf("%d %d %d", &(q.data), &(q.time), &(q.p));  
  
 while (k < n)  
 {  
 if(!f)  
 {  
 if(q.p == 0)  
 {  
 PutFifo(f1, q);  
 }  
 else if(q.p == 1){  
 PutFifo(f2, q);  
 }  
 else{  
 PutFifo(f3, q);  
 }  
  
 }  
  
 if ((!EmptyFifo(f1) && !p1) || (!EmptyFifo(f1) && !p2) )  
 {  
 if(!p1)  
 {  
 printf("f1 в q1\n");  
 GetFifo(f1, &q1);  
 k1 = 0;  
 p1 = 1;  
 }  
 else{  
 printf("f1 в q2\n");  
 GetFifo(f1, &q2);  
 k2 = 0;  
 p2 = 1;  
 }  
 }  
  
 if (EmptyFifo(f1) && ((!EmptyFifo(f2) && !p1) || (!EmptyFifo(f2) && !p2)))  
 {  
 if(!p1)  
 {  
 printf("f2 в q1\n");  
 GetFifo(f2, &q1);  
 k1 = 0;  
 p1 = 1;  
 }  
 else{  
 printf("f2 в q2\n");  
 GetFifo(f2, &q2);  
 k2 = 0;  
 p2 = 1;  
 }  
 }  
  
 if (EmptyFifo(f1) && EmptyFifo(f2) && ((!EmptyFifo(f3) && !p1) || (!EmptyFifo(f3) && !p2)))  
 {  
 if(!p2)  
 {  
 printf("f3 в q2\n");  
 GetFifo(f3, &q2);  
 k2 = 0;  
 p2 = 1;  
 }  
 else{  
 printf("f3 в q1\n");  
 GetFifo(f3, &q1);  
 k1 = 0;  
 p1 = 1;  
 }  
 }  
  
 if (!EmptyFifo(f1) && p1 && p2 && (q1.p != 0 || q2.p != 0))  
 {  
 printf("Работа со стеком для f1\n");  
 if(q1.p != 0)  
 {  
 PutStack(s1, q1);  
 GetFifo(f1, &q1);  
 k2 = 0;  
 }  
 else{  
 PutStack(s1, q2);  
 GetFifo(f1, &q2);  
 k2 = 0;  
 }  
 }  
 else if(!EmptyFifo(f2) && p1 && p2 && (q1.p == 2 || q2.p == 2))  
 {  
 printf("Работа со стеком для f2\n");  
 if(q1.p == 2)  
 {  
 PutStack(s2, q1);  
 GetFifo(f2, &q1);  
 k2 = 0;  
 }  
 else{  
 PutStack(s2, q2);  
 GetFifo(f2, &q2);  
 k2 = 0;  
 }  
 }  
 else if(!(EmptyStack(s1)|| !EmptyStack(s2))&& (!p1 || !p2))  
 {  
 printf("Достаем значение из стека\n");  
 if(!p1)  
 {  
 GetStack(s1, &q1);  
 k1 = 0;  
 p1 = 1;  
 }  
 else{  
 GetStack(s2, &q2);  
 k2 = 0;  
 p2 = 1;  
 }  
  
 }  
  
 if (p1)  
 {  
 k1++;  
 if (k1 >= q1.time)  
 {  
 printf("Обр. процессора p1: %d \n", q1.data);  
 k1 = 0;  
 p1 = 0;  
 k++;  
 }  
 }  
  
 if (p2)  
 {  
 k2++;  
 if (k2 >= q2.time)  
 {  
 printf("Обр. процессора p2: %d \n", q2.data);  
 k2 = 0;  
 p2 = 0;  
 k++;  
 }  
 }  
  
 if(!f)  
 {  
 printf("Введите запрос: ");  
 scanf("%d %d %d", &(q.data), &(q.time), &(q.p));  
 n++;  
  
 }  
 if (!f && q.data == 0)  
 {  
 f = 1;  
 n--;  
 }  
 }  
  
 return 0;  
}



Вывод: в ходе выполнения лабораторной работы были изучены СД типа «стек» и «очередь», научиться их программно реализовать и использовать.