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» (Pascal/C)

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили: асс. Солонченко Роман
Евгеньевич

Белгород 2023 г.

Лабораторная работа №6
Структуры данных «стек» и «очередь» (Pascal/C)
Вариант 8

Цель работы: изучить СД типа «стек» и «очередь», научиться их программно реализовывать и использовать.

1. Для СД типа «стек» определить:

   1.1. Абстрактный уровень представления СД:

      1.1.1. Характер организованности и изменчивости.
      Характер организованности - **линейный**. Характер изменчивости - **динамический**.

      1.1.2. Набор допустимых операций.
      Инициализация, включение элемента, исключение элемента, чтение текущего элемента, проверка пустоты стека, уничтожение стрека.

   1.2. Физический уровень представления СД:

      1.2.1. Схему хранения.
      Схема хранения - **связный.**

      1.2.2. Объем памяти, занимаемый экземпляром СД.
      Размер СД состоит из размера дескриптора, размера фиктивного элемента и размера всех элементов. Размер дескриптора: указатель на начало (машинное слово - 4 байт), рабочий указатель (машинное слово - 4 байт) и количество элементов (int - 4 байт). Размер элементов: $1+N\cdot(sizeof(BaseType)$ $4)$ байт. $V = 13 + N \cdot (sizeof(BaseType) + 4)$

      1.2.3. Формат внутреннего представления СД и способ его интерпретации.
      Дескриптор находится в статической памяти, элементы ОЛС находятся в динамической памяти.

      1.2.4. Характеристику допустимых значений.
      $Car(C) = 1+Car(BaseType)+Car(BaseType)^2+...+Car(BaseType)^{max}$.

      1.2.5. Тип доступа к элементам.
      Тип доступа к элементам - **последовательный**.

   1.3. Логический уровень представления СД.

      1.3.1. Способ описания СД и экземпляра СД на языке программирования.

```
Stack l;
InitStack(&l);
```

Для СД типа «очередь» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.
Характер организованности - **линейный**. Характер изменчивости - **динамический**.

1.1.2. Набор допустимых операций.
Инициализация, включение элемента, исключение элемента, чтение текущего элемента, проверка пустоты очереди, уничтожение очереди.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.
Схема хранения - **последовательный.**

1.2.2. Объем памяти, занимаемый экземпляром СД.
Дескриптор содержит 3 поля типа int - индекс головы, хвоста и количество элементов. $V = 3 \cdot 4 + N \cdot (sizeof(BaseType))$, где N - максимальное количество элементов.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.
СД находится в статической памяти.

1.2.4. Характеристику допустимых значений.
$Car(C) = 1 + Car(BaseType) + Car(BaseType)^2 + ... + Car(BaseType)^N$.

1.2.5. Тип доступа к элементам.
Тип доступа к элементам - **последовательный**.

1.3. Логический уровень представления СД.

1.3.1. Способ описания СД и экземпляра СД на языке программирования.

```
Queue q;
InitQueue(&q);
```

2. Реализовать СД типа «стек» и «очередь» в соответствии с вариантом индивидуального задания в виде модуля.
main.c (тесты)

```
#include <algc.h>

#include <stdio.h>
#include <assert.h>
#include <string.h>

void testPutStack() {
    Stack s1;
    InitStack(&s1);

    PutStack(&s1, 1);
    assert(StackError == StackOk);
```

```
    PutStack(&s1, 2);
    assert(StackError == StackOk);
    PutStack(&s1, 3);
    assert(StackError == StackOk);

    int e;
    GetStack(&s1, &e);
    assert(e == 3);
    GetStack(&s1, &e);
    assert(e == 2);
    GetStack(&s1, &e);
    assert(e == 1);

    DoneStack(&s1);
}

void testGetStack() {
    Stack s1;
    InitStack(&s1);
    int e;
    GetStack(&s1, &e);
    assert(StackError == StackUnder);


    PutStack(&s1, 1);
    PutStack(&s1, 2);
    PutStack(&s1, 3);

    GetStack(&s1, &e);
    assert(e == 3 && StackError == StackOk);
    GetStack(&s1, &e);
    assert(e == 2 && StackError == StackOk);
    GetStack(&s1, &e);
    assert(e == 1 && StackError == StackOk);
    GetStack(&s1, &e);
    assert(StackError == StackUnder);

    DoneStack(&s1);
}

void testEmptyStack() {
    Stack s1;
    InitStack(&s1);
```

```
    assert(EmptyStack(s1) && StackError == StackOk);

    PutStack(&s1, 1);
    assert(!EmptyStack(s1) && StackError == StackOk);

    PutStack(&s1, 2);
    assert(!EmptyStack(s1) && StackError == StackOk);

    PutStack(&s1, 3);
    assert(!EmptyStack(s1) && StackError == StackOk);

    DoneStack(&s1);
}

void testReadStack() {
    Stack s1;
    InitStack(&s1);
    int e;
    ReadStack(&s1, &e);
    assert(StackError == StackUnder);


    PutStack(&s1, 1);
    ReadStack(&s1, &e);
    assert(e == 1 && StackError == StackOk);

    PutStack(&s1, 2);
    ReadStack(&s1, &e);
    assert(e == 2 && StackError == StackOk);

    PutStack(&s1, 3);
    ReadStack(&s1, &e);
    assert(e == 3 && StackError == StackOk);

    DoneStack(&s1);
}

void testStack() {
    testPutStack();
    testGetStack();
    testEmptyStack();
    testReadStack();
```

```
}

void testPutQueue() {
    Queue q1;
    InitQueue(&q1);

    PutQueue(&q1, 1);
    assert(QueueError == QueueOk);
    PutQueue(&q1, 2);
    assert(QueueError == QueueOk);
    PutQueue(&q1, 3);
    assert(QueueError == QueueOk);
    int e;
    GetQueue(&q1, &e);
    GetQueue(&q1, &e);
    GetQueue(&q1, &e);

    for (int i = 0; i < QueueSize; i++) {
        PutQueue(&q1, 1);
        assert(QueueError == QueueOk);
    }

    PutQueue(&q1, 1);
    assert(QueueError == QueueOver);
}

void testGetQueue() {
    Queue q1;
    InitQueue(&q1);
    int e;
    GetQueue(&q1, &e);
    assert(QueueError == QueueUnder);

    PutQueue(&q1, 1);
    PutQueue(&q1, 2);
    PutQueue(&q1, 3);
    GetQueue(&q1, &e);
    assert(QueueError == QueueOk && e == 1);
    GetQueue(&q1, &e);
    assert(QueueError == QueueOk && e == 2);
    GetQueue(&q1, &e);
    assert(QueueError == QueueOk && e == 3);
}
```

```
void testEmptyQueue() {
    Queue q1;
    InitQueue(&q1);
    assert(EmptyQueue(&q1) && QueueError == QueueOk);

    for (int i = 0; i < 1000; i++) {
        PutQueue(&q1, 1);
        assert(!EmptyQueue(&q1) && QueueError == QueueOk);
    }
}

void testReadQueue() {
    Queue q1;
    InitQueue(&q1);
    int e;
    ReadQueue(&q1, &e);
    assert(QueueError == QueueUnder);

    PutQueue(&q1, 1);
    ReadQueue(&q1, &e);
    assert(QueueError == QueueOk && e == 1);
    PutQueue(&q1, 2);
    ReadQueue(&q1, &e);
    assert(QueueError == QueueOk && e == 1);
}

void testQueue() {
    testPutQueue();
    testGetQueue();
    testEmptyQueue();
    testReadQueue();
}

void test() {
    testStack();
    testQueue();
}

int main() {
    test();

    return 0;
```

```
}
```

stack.h (заголовки)

```
#ifndef STACK
#define STACK

#define SINGLY_CONNECTED_LIST_CUSTOM_BASE_TYPE

#ifndef CUSTOM_BASE_TYPE
typedef int BaseType;
#endif

#include <lab5/singlyconnectedlist.h>

#define StackOk ListOk
#define StackUnder ListUnder
#define StackOver ListNotMem

extern int StackError;

typedef List Stack;

void InitStack(Stack *s);
void PutStack(Stack *s, BaseType E);
void GetStack(Stack *s, BaseType *E);
int EmptyStack(Stack s);
void ReadStack(Stack *s, BaseType *E);
void DoneStack(Stack *s);

#endif
```

queue.h (заголовки)

```
#ifndef QUEUE
#define QUEUE

#define QueueSize 1000

#define QueueOk 0
#define QueueUnder 1
#define QueueOver 2
```

```
extern int QueueError;

#ifndef CUSTOM_BASE_TYPE
typedef int BaseType;
#endif

typedef struct {
    BaseType Buf[QueueSize];
    unsigned Uk1; // Голова
    unsigned Uk2; // Хвост
    unsigned N;
} Queue;

void InitQueue(Queue* f);
void PutQueue(Queue *f, BaseType E);
void GetQueue(Queue *f, BaseType *E);
void ReadQueue(Queue *f, BaseType *E);
int EmptyQueue(Queue *f);

#endif
```

stack.c (реализации функций)

```
#include <lab5/singlyconnectedlist.h>
#include <lab6/stack.h>

int StackError = StackOk;

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;
}
int EmptyStack(Stack s) {
    StackError = StackOk;

    return s.N == 0;
}

void ReadStack(Stack *s, BaseType *E) {
    ReadList(s, E);

    StackError = ListError;
}

void DoneStack(Stack *s) {
    DoneList(s);

    StackError = ListError;
}
```

queue.c (реализации функций)

```
#include <lab6/queue.h>

int QueueError = QueueOk;

void InitQueue(Queue* f) {
    f->Uk1 = 0;
    f->Uk2 = 0;
    f->N = 0;

    QueueError = QueueOk;
}

void PutQueue(Queue *f, BaseType E) {
    if (f->N >= QueueSize) {
        QueueError = QueueOver;
        return;
    }

    QueueError = QueueOk;
    f->Buf[f->Uk2] = E;
```

```
    f->Uk2 = (f->Uk2 + 1) % QueueSize;
    f->N++;
}

void GetQueue(Queue *f, BaseType *E) {
    if (f->N <= 0) {
        QueueError = QueueUnder;
        return;
    }

    QueueError = QueueOk;
    *E = f->Buf[f->Uk1];
    f->Uk1 = (f->Uk1 + 1) % QueueSize;
    f->N--;
}

void ReadQueue(Queue *f, BaseType *E) {
    if (f->N <= 0) {
        QueueError = QueueUnder;
        return;
    }

    QueueError = QueueOk;
    *E = f->Buf[f->Uk1];
}

int EmptyQueue(Queue *f) {
    QueueError = QueueOk;

    return f->N == 0;
}
```

3. Текст программы моделирования системы.
main.c (основная программа)

```
typedef struct {
    char Name[10]; // имя запроса
    unsigned Time; // время обслуживания
    char T; // тип задачи 1 – T1, 2 – T2
} TInquiry;

#define CUSTOM_BASE_TYPE
```

```c
typedef TInquiry BaseType;

#define TInquiryType1 1
#define TInquiryType2 2

#include <stddef.h>
#include <stdio.h>
#include <time.h>
#include <string.h>

#include <algc.h>
#include "../../libs/alg/lab5/task2.c"
#include "../../libs/alg/lab6/queue.c"
#include "../../libs/alg/lab6/stack.c"

void gen(Queue *F1, Queue *F2) {
    char tmp;
    printf("Input first task type amount (leave -1 for random from 1 to 5): ");
    int f1Amount;
    scanf("%d", &f1Amount);
    if (f1Amount < 0) {
        f1Amount = 1 + rand() % 5;
        printf("%d tasks\n", f1Amount);
    }
    gets(&tmp);

    for (int i = 0; i < f1Amount; i++) {
        printf("Input task name (leave blank for random): ");
        TInquiry f1Task;
        f1Task.T = TInquiryType1;
        gets(f1Task.Name);
        if (!f1Task.Name[0]) {
            int j = 0;
            while (j < sizeof(f1Task.Name) - 1 && rand() % 10 < 7)
                f1Task.Name[j++] = 'a' + rand() % ('z' - 'a');

            f1Task.Name[j] = '\0';

            printf("Task name: %s\n", f1Task.Name);
        }

        printf("Input task time (leave 0 for random from 1 to 7): ");
        scanf("%d", &f1Task.Time);
```

```
        gets(&tmp);
        if (f1Task.Time == 0) {
            f1Task.Time = 1 + rand() % 7;

            printf("Task time: %d\n", f1Task.Time);
        }

        PutQueue(F1, f1Task);
    }

    printf("Input second task type amount (leave -1 for random from 1 to 5): ");
    int f2Amount;
    scanf("%d", &f2Amount);
    if (f2Amount < 0) {
        f2Amount = 1 + rand() % 5;
        printf("%d tasks\n", f2Amount);
    }
    gets(&tmp);

    for (int i = 0; i < f2Amount; i++) {
        printf("Input task name (leave blank for random): ");
        TInquiry f2Task;
        f2Task.T = TInquiryType2;
        gets(f2Task.Name);
        if (!f2Task.Name[0]) {
            int j = 0;
            while (j < sizeof(f2Task.Name) - 1 && rand() % 10 < 7)
                f2Task.Name[j++] = 'a' + rand() % ('z' - 'a');

            f2Task.Name[j] = '\0';

            printf("Task name: %s\n", f2Task.Name);
        }

        printf("Input task time (leave 0 for random from 1 to 7): ");
        scanf("%d", &f2Task.Time);
        gets(&tmp);
        if (f2Task.Time == 0) {
            f2Task.Time = 1 + rand() % 7;

            printf("Task time: %d\n", f2Task.Time);
        }
```

```
        PutQueue(F2, f2Task);
    }
}

void outputTInquiry(TInquiry t) {
    printf("=========================\n\n");
    printf("Name: %s\n", t.Name);
    printf("Time: %u\n", t.Time);
    printf("Type: %s\n", t.T == TInquiryType1 ? "1" : t.T == TInquiryType2 ? "2" :
    ↪  "unknown");
    printf("\n");
}

int main() {
    srand(time(0));

    Stack S;
    InitStack(&S);

    Queue F1, F2;
    InitQueue(&F1);
    InitQueue(&F2);
    gen(&F1, &F2);

    TInquiry *P1 = NULL, *P2 = NULL;

    while (1) {
        if (P1 != NULL) {
            P1->Time--;

            if (!P1->Time) {
                free(P1);
                P1 = NULL;
            }
        }

        if (P2 != NULL) {
            P2->Time--;

            if (!P2->Time) {
                free(P2);
                P2 = NULL;
            }
```

```
        }

        if (!EmptyQueue(&F1)) {
            if (P1 == NULL) {
                P1 = malloc(sizeof(TInquiry));
                GetQueue(&F1, P1);
            } else if (P1->T == 2 && P2 == NULL) {
                P2 = malloc(sizeof(TInquiry));
                P2 = P1;
                GetQueue(&F1, P1);
            } else {
                TInquiry t;
                GetQueue(&F1, &t);
                PutStack(&S, t);
            }
        } else if (P1 == NULL && P2 != NULL && !EmptyStack(S)){
            P1 = malloc(sizeof(TInquiry));
            GetStack(&S, P1);
        }
        if (!EmptyQueue(&F2)) {
            if (P2 == NULL) {
                P2 = malloc(sizeof(TInquiry));
                GetQueue(&F2, P2);
            } else if (P1 == NULL && EmptyQueue(&F1)) {
                P1 = malloc(sizeof(TInquiry));
                GetQueue(&F2, P1);
            }
        } else if (P2 == NULL && !EmptyStack(S)) {
            P2 = malloc(sizeof(TInquiry));
            GetStack(&S, P2);
        }

        printf("Tasks in first queue:\n");
        for (int i = F1.Uk1; i != F1.Uk2; i = (i + 1) % QueueSize) {
            TInquiry t = F1.Buf[i];
            outputTInquiry(t);
        }

        printf("Tasks in second queue:\n");
        for (int i = F2.Uk1; i != F2.Uk2; i = (i + 1) % QueueSize) {
            TInquiry t = F1.Buf[i];
            outputTInquiry(t);
        }
```

```
    if (P1 == NULL) {
        printf("P1 is not busy.\n");
    } else {
        printf("P1 current task:\n");
        outputTInquiry(*P1);
    }

    if (P2 == NULL) {
        printf("P2 is not busy.\n");
    } else {
        printf("P2 current task:\n");
        outputTInquiry(*P2);
    }



    printf("Tasks in stack:\n");
    ptrel stackTask = S.Start->next;
    while (stackTask != NULL) {
        outputTInquiry(stackTask->data);

        stackTask = stackTask->next;
    }



    while (1) {
        printf("Type g to add more tasks, n for next iteration, e to exit program\n");
        int input = getchar();

        if (input == 'g') {
            gen(&F1, &F2);
        } else if (input == 'n') {
            break;
        } else if (input == 'e') {
            return 0;
        }

        getchar();
    }
}

return 0;
```

```
}
```

Результаты работы программы:

Время	Объекты	Задачи
0	F_1	(F1T0, 5), (F1T1, 3), (F1T2, 2)
	F_2	(F2T0, 7), (F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	
	P_1	
	P_2	
1	F_1	(F1T1, 3), (F1T2, 2)
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	
	P_1	(F1T0, 5)
	P_2	(F2T0, 7)
2	F_1	(F1T2, 2)
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T1, 3)
	P_1	(F1T0, 4)
	P_2	(F2T0, 6)
3	F_1	
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T2, 2), (F1T1, 3)
	P_1	(F1T0, 3)
	P_2	(F2T0, 5)
4	F_1	
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T2, 2), (F1T1, 3)
	P_1	(F1T0, 2)
	P_2	(F2T0, 4)
5	F_1	
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T2, 2), (F1T1, 3)
	P_1	(F1T0, 1)
	P_2	(F2T0, 3)
6	F_1	
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T1, 3)
	P_1	(F1T2, 2)
	P_2	(F2T0, 2)
7	F_1	
	F_2	(F2T1, 7), (F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T1, 3)
	P_1	(F1T2, 1)

	P_2	(F2T0, 1)
8	F_1	
	F_2	(F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	(F1T1, 3)
	P_1	
	P_2	(F2T1, 7)
9	F_1	
	F_2	(F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	
	P_1	(F1T1, 3)
	P_2	(F2T1, 6)
10	F_1	
	F_2	(F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	
	P_1	(F1T1, 2)
	P_2	(F2T1, 5)
11	F_1	
	F_2	(F2T2, 5), (F2T3, 4), (F2T4, 7)
	S	
	P_1	(F1T1, 1)
	P_2	(F2T1, 4)
12	F_1	
	F_2	(F2T3, 4), (F2T4, 7)
	S	
	P_1	(F2T2, 5)
	P_2	(F2T1, 3)
13	F_1	
	F_2	(F2T3, 4), (F2T4, 7)
	S	
	P_1	(F2T2, 4)
	P_2	(F2T1, 2)
14	F_1	
	F_2	(F2T3, 4), (F2T4, 7)
	S	
	P_1	(F2T2, 3)
	P_2	(F2T1, 1)
15	F_1	
	F_2	(F2T4, 7)
	S	
	P_1	(F2T2, 2)
	P_2	(F2T3, 4)
16	F_1	
	F_2	(F2T4, 7)
	S	
	P_1	(F2T2, 1)
	P_2	(F2T3, 3)

17	F_1	
	F_2	
	S	
	P_1	(F2T4, 7)
	P_2	(F2T3, 2)
18	F_1	
	F_2	
	S	
	P_1	(F2T4, 6)
	P_2	(F2T3, 1)
19	F_1	
	F_2	
	S	
	P_1	(F2T4, 5)
	P_2	
20	F_1	
	F_2	
	S	
	P_1	(F2T4, 4)
	P_2	
21	F_1	
	F_2	
	S	
	P_1	(F2T4, 3)
	P_2	
22	F_1	
	F_2	
	S	
	P_1	(F2T4, 2)
	P_2	
23	F_1	
	F_2	
	S	
	P_1	(F2T4, 1)
	P_2	
24	F_1	
	F_2	
	S	
	P_1	
	P_2	

Вывод: в ходе лабораторной работы изучили СД типа «стек» и «очередь», научились их программно реализовывать и использовать.