**МИНОБРНАУКИ РОССИИ**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

Дисциплина: Алгоритмы и структуры данных

по теме «Структуры данных «линейные списки» »

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Синюк В.Г.

Белгород 2018**Лабораторная работа №6**

***Структуры данных «стек» и «очередь» (Pascal/С)***

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

З а д а н и е

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.

Система состоит из двух процессоров *P*1 и *P*2, трех очередей *F*1, *F*2, *F*3 и стека (рис.12).

P*1*

*Стек*

P*2*

*ОЧЕРЕДЬ* F*1*

*ОЧЕРЕДЬ* F*2*

*ОЧЕРЕДЬ* F*3*

Генератор задач

Рис.12. Система задач 7 и 10

В систему могут поступать запросы на выполнение задач трех типов — Т1, Т2, Т3. Задача типа Т1 может выполняться только процессором P1. Задача типа Т2 может выполняться только процессором P2. Задача типа Т3 может выполняться любым процессором. Запрос можно представить записью.

Поступающие запросы ставятся в соответствующие типам задач очереди. Если очередь F1 не пуста и процессор P1 свободен, то задача из очереди F1 поступает на обработку в процессор P1. Если процессор Р1 обрабатывает задачу типа Т3, а процессор Р2 свободен и очередь F2 пуста, то задача из процессора Р1 поступает в процессор Р2, а задача из очереди F1 в процессор Р1, если же процессор Р2 занят или очередь F2 не пуста, то задача из процессора Р1 помещается в стек.

Если очередь F2 не пуста и процессор P2 свободен, то задача из очереди F2 поступает на обработку в процессор P2. Если процессор Р2 обрабатывает задачу типа Т3, а процессор Р1 свободен и очередь F1 пуста, то задача из процессора Р2 поступает в процессор Р1, а задача из очереди F2 — в процессор Р2, если же процессор Р1 занят или очередь F1 не пуста, то задача из процессора Р1 помещается в стек.

Если очередь F3 не пуста и процессор P1 свободен, и очередь F1 пуста или свободен процессор Р2 и очередь F2 пуста, то задача из очереди F3 поступает на обработку в свободный процессор. Задача из стека поступает на обработку в свободный процессор Р1, если очередь F1 пуста, или в свободный процессор Р2, если очередь F2 пуста

#include <stdio.h>

#include <stdlib.h>

#include "queue.h"

#include "stack.h"

typedef struct prc{

char status;

base\_type task;

}processor;

int main(){

processor p1,p2;

p1.status=0;

p2.status=0;

stack \*s=stack\_init(50);

queue \*q1=queue\_init(50),\*q2=queue\_init(50),\*q3=queue\_init(50);

char clock\_stop=0,input\_flag=0;

base\_type task\_in;

FILE \*file\_in;

int tick=0;

char file\_name[128];

printf("File name?: ");

scanf("%s", file\_name);

if ((file\_in = fopen(file\_name, "r"))==NULL){

printf("Unable to open");

return 1;

}

while(!clock\_stop){

printf("TICK %d\n",tick );

//обработчики задач

if((!p1.status==0)&&(p1.task.time!=0)){

p1.task.time--;

if(p1.task.time==0){

p1.status=0;

}

}

if((!p2.status==0)&&(p2.task.time!=0)){

p2.task.time--;

if(p2.task.time==0){

p2.status=0;

}

}

//планировщик

if(!queue\_empty(q1)){

if(p1.status==3){

if((p2.status==0)&&(queue\_empty(q2))){

p2.task=p1.task;

p2.status=3;

p1.task=queue\_get(q1);

p1.status=1;

}

else{

stack\_put(s,p1.task);

p1.task=queue\_get(q1);

p1.status=1;

}

}

else{

if(p1.status==1){

stack\_put(s,queue\_get(q1));

}

else{

p1.task=queue\_get(q1);

p1.status=1;

}

}

}

if(!queue\_empty(q2)){

if(p2.status==3){

if((p1.status==0)&&(queue\_empty(q1))){

p1.task=p2.task;

p1.status=3;

p2.task=queue\_get(q2);

p2.status=2;

}

else{

stack\_put(s,p2.task);

p2.task=queue\_get(q2);

p2.status=2;

}

}

else{

if(p2.status==2){

stack\_put(s,queue\_get(q2));

}

else{

p2.task=queue\_get(q2);

p2.status=2;

}

}

}

if(!queue\_empty(q3)){

if((p1.status==0)&&(queue\_empty(q1))){

p1.task=queue\_get(q3);

p1.status=3;

}

if((p2.status==0)&&(queue\_empty(q2))){

p2.task=queue\_get(q3);

p2.status=3;

}

}

if((queue\_empty(q1))&&(queue\_empty(q3))&&(p1.status==0)){

if (!(stack\_empty(s))){

base\_type temp\_task=stack\_read(s);

if((temp\_task.type==3)&&(temp\_task.type==1)){

p1.task=temp\_task;

p1.status=temp\_task.type;

stack\_get(s);

}}

}

if((queue\_empty(q2))&&(queue\_empty(q3))&&(p2.status==0)){

if(!(stack\_empty(s))){

base\_type temp\_task=stack\_read(s);

if((temp\_task.type==3)&&(temp\_task.type==2)){

p2.task=temp\_task;

p2.status=temp\_task.type;

stack\_get(s);

}}

}

//ввод задач

if(!input\_flag){

switch (task\_input(file\_in,&task\_in)) {

case 0:{

input\_flag=1;

break;}

case 1:{

queue\_put(q1,task\_in);

break;}

case 2:{

queue\_put(q2,task\_in);

break;}

case 3:{

queue\_put(q3,task\_in);

break;}

}}

//вывод

printf("q1: "); queue\_print(q1);

printf("q2: "); queue\_print(q2);

printf("q3: "); queue\_print(q3);

printf("s : "); stack\_print(s);

printf("p1: ");

if (p1.status>0){

task\_out(p1.task);

printf("\n");

} else printf("idle\n");

printf("p2: ");

if (p2.status>0){

task\_out(p2.task);

printf("\n");

} else printf("idle\n");

clock\_stop=(p1.status==0)&&(p2.status==0)&&(queue\_empty(q1))&&(queue\_empty(q2))&&(queue\_empty(q3))&&stack\_empty(s)&&input\_flag;

tick++;

}

return 1;

}

Очередь на ОЛС

Заголовочный файл

#ifndef QUEUE\_H\_

#define QUEUE\_H\_

#include "list.h"

extern const char queue\_ok;

extern const char queue\_under;

extern const char queue\_over;

extern const char queue\_mem\_error;

char queue\_error; // Переменная ошибок

typedef list queue;

queue\* queue\_init(unsigned size); // Инициализация очереди

void queue\_put(queue \*q, base\_type e); // Поместить элемент в очередь

base\_type queue\_get(queue \*q); // Извлечь элемент из очереди

base\_type queue\_read(queue \*q); // Прочитать элемент

void queue\_print(queue \*q);

void queue\_done(queue \*q); //Деструктор

//Предикаты

int queue\_empty(queue \*q);

int queue\_full(queue \*q);

#endif

Исходный код

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

#include "queue.h"

const char queue\_ok = 1;

const char queue\_under = 2;

const char queue\_over = 3;

const char queue\_mem\_error = 4;

queue\* queue\_init(unsigned size){

list\* temp\_pointer;

temp\_pointer=list\_init(size);

if(temp\_pointer==NULL){

queue\_error=queue\_mem\_error;

exit(queue\_error);

}

queue\_error=queue\_ok;

return temp\_pointer;

}

void queue\_put(queue \*q, base\_type e){

if(queue\_full(q)){

queue\_error=queue\_over;

exit(queue\_error);

}

pos\_move\_to\_last(q);

list\_put(q,e);

}

base\_type queue\_get(queue \*q){

if(queue\_empty(q)){

queue\_error=queue\_under;

exit(queue\_error);

}

pos\_move\_to\_first(q);

list\_next(q);

base\_type temp;

temp=list\_get(q);

return temp;

}

base\_type queue\_read(queue \*q){

if(queue\_empty(q)){

queue\_error=queue\_under;

exit(queue\_error);

}

pos\_move\_to\_first(q);

list\_next(q);

base\_type temp;

temp=list\_read(q);

return temp;

}

void queue\_print(queue \*q){

if(q->list\_lenght==0){

printf("empty \n");

}

else{

pos\_move\_to\_first(q);

list\_next(q);

for(int i=0;i<q->list\_lenght;i++){

task\_out(list\_read(q));

printf(", ");

list\_next(q);

printf("\n");

}}

}

int queue\_empty(queue \*q){

return list\_empty(q);

}

int queue\_full(queue \*q){

return list\_full(q);

}

void queue\_done(queue \*q){

list\_done(q);

}

Стек на массиве

Заголовочный файл

#ifndef STACK\_H\_

#define STACK\_H\_

#include "list.h"

extern const char stack\_ok;

extern const char stack\_underflow;

extern const char stack\_overflow;

extern const char stack\_mem\_error;

char stack\_error; // Переменная ошибок

typedef struct{

int stack\_top;

unsigned stack\_lenght;

unsigned stack\_size;

base\_type\* stack\_arr;

} stack;

stack\* stack\_init(unsigned size); //Инициализация стека

void stack\_done(stack \*s); //Деструктор

void stack\_put(stack \*s,base\_type e); //Поместить элемент в стек

base\_type stack\_get(stack \*s); //Извлечь элемент из стека

base\_type stack\_read(stack \*s); //чтение без извлечения

void stack\_print(stack \*s);

// Предикаты

int stack\_empty(stack\* s);

int stack\_full(stack\* s);

#endif

Исходный код

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

#include "stack.h"

const char stack\_ok = 1;

const char stack\_underflow = 2;

const char stack\_overflow = 3;

const char stack\_mem\_error = 4;

stack\* stack\_init(unsigned size){

stack\* temp\_pointer;

temp\_pointer=(stack\*)malloc(sizeof(stack));

if(temp\_pointer==NULL){

stack\_error=stack\_mem\_error;

exit(stack\_error);

}

temp\_pointer->stack\_size=size;

temp\_pointer->stack\_arr=(base\_type\*)malloc(sizeof(base\_type)\*size);

temp\_pointer->stack\_top=-1;

stack\_error=stack\_ok;

temp\_pointer->stack\_lenght=0;

return temp\_pointer;

}

void stack\_put(stack \*s, base\_type e){

if(stack\_full(s)){

stack\_error=stack\_overflow;

exit(stack\_error);

}

s->stack\_top++;

s->stack\_arr[s->stack\_top]=e;

s->stack\_lenght++;

}

base\_type stack\_get(stack \*s){

if(stack\_empty(s)){

stack\_error=stack\_underflow;

exit(stack\_error);

}

base\_type temp;

temp=s->stack\_arr[s->stack\_top];

s->stack\_top--;

s->stack\_lenght--;

return temp;

}

base\_type stack\_read(stack \*s){

if(stack\_empty(s)){

stack\_error=stack\_underflow;

exit(stack\_error);

}

return s->stack\_arr[s->stack\_top];

}

void stack\_print(stack \*s){

if(s->stack\_lenght!=0){

for(int i=0;i<s->stack\_lenght;i++){

task\_out(s->stack\_arr[i]);

printf(", ");

}

}

else{

printf("empty");

}

printf("\n");

}

int stack\_full(stack \*s){

if(s->stack\_top>=s->stack\_size-1) return 1;

else return 0;

}

int stack\_empty(stack \*s){

if(s->stack\_top==-1) return 1;

else return 0;

}

void stack\_done(stack \*s){

free(s->stack\_arr);

free(s);

}

