**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**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 №5

дисциплина: Алгоритмы и Структуры данных

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

Выполнил: ст. группы ПВ-21

Донцов Александр Алексеевич

Проверил: Синюк Василий Григорьевич

Белгород 2018

**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.

Заголовочный файл Turn.h

#pragma once

#if !defined(\_\_FIFO1\_H)

#define \_\_FIFO1\_H

const int ListOk = 0;

const int ListNotMem = 1;

const int ListUnder = 2;

const int ListEnd = 3;

typedef void\* BaseType;

struct el {

BaseType data;

};

typedef struct el element;

typedef struct {

element\* arr;

int first;

int last;

int MaxSize;

int SizeEl;

int size;

}List;

int const FifoOk = ListOk;

int const FifoUnder = ListUnder;

int const FifoOver = ListNotMem;

int FifoError; // Переменная ошибок

typedef List Fifo;

void InitFifo(Fifo \*f, unsigned SizeMem, unsigned SizeEl); // Инициализация очереди

void PutFifo(Fifo \*f, BaseType E); // Поместить элемент в очередь

void GetFifo(Fifo \*f, BaseType \*E); // Извлечь элемент из очереди

void ReadFifo(Fifo \*f, BaseType \*E); // Прочитать элемент

int EmptyFifo(Fifo \*f); // Проверка, пуста ли очередь?

int FullFifo(Fifo \*f);// Проверка, полна ли очередь?

void DoneFifo(Fifo \*f); // Разрушить очередь

#endif

// 1. Очередь на ПЛС. «Хвост» очереди — последний, а «голова» — пер-вый элемент ПЛС.

Исходный файл Turn.cpp

#include "Turn.h"

#include "malloc.h"

#include "windows.h"

void InitFifo(Fifo \*f, unsigned SizeMem, unsigned SizeEl) {// Инициализация очереди

f->arr = (element\*)malloc(sizeof(el)\*SizeMem);

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

f->arr[i].data = (BaseType)malloc(sizeof(f->SizeEl));

f->MaxSize = SizeMem;

f->SizeEl = SizeEl;

f->size = 0;

f->first = 0;

f->last = -1;

FifoError = ListOk;

}

void PutFifo(Fifo \*f, BaseType E) { // Поместить элемент в очередь

f->size++;

if (FullFifo(f))

FifoError = ListEnd;

exit(ListEnd);

f->last++;

if (f->last == f->MaxSize)

f->last = 0;

f->arr[f->last - 1].data = E;

FifoError = ListOk;

}

void GetFifo(Fifo \*f, BaseType \*E) { // Извлечь элемент из очереди

if (EmptyFifo(f)) {

FifoError = ListNotMem;

exit(ListNotMem);

}

\*E = f->arr[f->first].data;

f->first++;

if (f->first == f->MaxSize)

f->first = 0;

FifoError = ListOk;

}

void ReadFifo(Fifo \*f, BaseType \*E) { // Прочитать элемент

\*E = f->arr[f->first].data;

FifoError = ListOk;

}

int EmptyFifo(Fifo \*f) { // Проверка, пуста ли очередь?

if (f->size == 0)

return 1;

else

return 0;

}

void DoneFifo(Fifo \*f) { // Разрушить очередь

for (int i = 0; i < f->MaxSize; i++)

free(f->arr[i].data);

free(f);

}

int FullFifo(Fifo \*f) { // Проверка, заполнена ли очередь?

if (f->size == f->MaxSize)

return 1;

else

return 0;

}

Заголовочный файл Stack.h

#pragma once

#if !defined(\_\_STACK1\_H)

#define \_\_STACK1\_H

#include "Structs.h"

int const StackSize = 1000;

int const StackOk = 0;

int const StackOver = 1;

int const StackUnder = 2;

int StackError; // Переменная ошибок

typedef TInquiry BaseType; // Определить тип элемента стека

typedef struct {

BaseType Buf[StackSize];

unsigned Uk; // Указывает на элемент, являющийся вершиной стека

}Stack;

void InitStack(Stack \*\*s); // Инициализация стека

int EmptyStack(Stack \*s); // Проверка: стек пуст?

void PutStack(Stack \*s, BaseType E); // Поместить элемент в стек

void GetStack(Stack \*s, BaseType \*E); // Извлечь элемент из стека

void ReadStack(Stack \*s, BaseType \*E); // Прочитать элемент из вершины стека

void DoneStack(Stack \*f); // Разрушить очередь

#endif

Исходный файл Stack.cpp

#include "Stack.h"

#include <malloc.h>

#include "windows.h"

void InitStack(Stack \*\*s) { // Инициализация стека

\*s = (Stack\*)malloc(sizeof(Stack));

(\*s)->Uk = -1;

}

int EmptyStack(Stack \*s) { // Проверка: стек пуст?

if (s->Uk != -1)

return 1;

else

return 0;

}

void PutStack(Stack \*s, BaseType E) { // Поместить элемент в стек \*/

if (s->Uk == StackSize) {

StackError = StackOver;

exit(StackError);

}

s->Buf[s->Uk] = E;

s->Uk++;

StackError = StackOk;

}

void GetStack(Stack \*s, BaseType \*E) { // Извлечь элемент из стека \*/

if (s->Uk == StackSize) {

StackError = StackUnder;

exit(StackError);

}

s->Uk--;

\*E = s->Buf[s->Uk];

}

void ReadStack(Stack \*s, BaseType \*E) { // Прочитать элемент

if (EmptyStack(s)) {

StackError = StackUnder;

exit(StackUnder);

}

\*E = s->Buf[s->Uk];

}

void DoneStack(Stack \*s) { // Разрушить очередь

free(s);

}

Заголовочный файл processors.h

#pragma once

#ifndef PROCESSORS\_H\_

#define PROCESSORS\_H\_

#include "Structs.h"

void InitCPU(CPU \*p);//инициализация процессора

int isFreeCPU(CPU\* p);//свободен ли процессло?

void putTaskCPU(CPU\* p, TInquiry \*task);//положить задачу в процессор

int doCPU(CPU \*);//делать вычисления

#endif

Исходный файл processors.cpp

#include "processors.h"

#include "stdlib.h"

#include "stdio.h"

#include "System.h"

//инициализация процессора

void InitCPU(CPU \*p) {

p->flag = 1;

p->task = NULL;

}

//свободен ли процесcор? (1 - свободен, иначе 0 - занят)

int isFreeCPU(CPU \*p) {

return p->flag;

}

//положить задачу в процессор

void putTaskCPU(CPU\* p, TInquiry \*task) {

p->task = task;

p->flag = 0;

}

//делать вычисления

int doCPU(CPU\* p) {

printf("CPU: ");

if (p->task == NULL) { //если процессор без задачи

//простой

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

}

else {

printf("%s (Priority: %c, Time: %d)\n", p->task->Name, p->task->P, p->task->Time);

p->task->Time--;

if (p->task->Time <= 0) { //если задача решена, то освободить процессор

p->flag = 1; // устанавливаем процессор свободен

freeTask(p->task);

p->task = NULL; //удаляем задачу из процессора

//printf("-------\n");

return 1;

}

}

return 0;

}

Заголовочный файл System.h

#pragma once

#ifndef SYSTEM\_H\_

#define SYSTEM\_H\_

#include "Turn.h"

#include "Stack.h"

#include "processors.h"

//количество решаемых задач

#define COUNT\_TASK 5

//-------Глобальные переменные---------

extern Fifo f0;

extern Fifo f2;

extern Fifo f3;

extern Stack \*stack;

extern CPU cpu1;

extern CPU cpu2;

extern CPU cpu3;

extern short CountFinishTask;//количество решённых задачы

void outputLists();//вывод очередей и стеков

TInquiry\* createTask(void);//создание (ввод) задач

void freeTask(TInquiry\* task);//уничтожение задачи

void generatorTasks(void);//генератор задач

void distributorTasks(void);//распределитель задач

void systemInit(void);//инициализация системы

void doSystem(int teak);//работа системы (возвращает количество решённыз задач)

int isFinishSystem(void);//система закончила свою работу? (все задачи решены)

#endif

Исходный файл System.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "System.h"

#include "stdio.h"

#include "malloc.h"

#include "processors.h"

#include "Structs.h"

#include "Turn.h"

//-------Глобальные переменные---------

Fifo fifo0;

Fifo fifo1;

Fifo fifo2;

Stack \*stack;

CPU cpu;

short CountFinishTask = 0; //количество решённых задач

//создание (ввод) задач

TInquiry\* createTask(void) {

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

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

printf("Enter name of task: ");

scanf("%s", tmp->Name);

printf("Enter priority of task (0,1,2): ");

scanf("%d", &tmp->P);

printf("Enter time for solution: ");

scanf("%d", &tmp->Time);

return tmp;

}

//уничтожение задачи

void freeTask(TInquiry\* task) {

free(task);

}

//генератор задач

void generatorTasks(void) {

int count;// счётчик созданных задач

printf("Enter count tasks: ");

scanf("%d", &count);

TInquiry\* tmp;

for (int i = count; i > 0; i--) {

tmp = createTask();//создаём задачу

switch (tmp->P) {

case('0'):

PutFifo(&fifo0, tmp); //кладём её в очередь

break;

case('1'):

PutFifo(&fifo1, tmp); //кладём её в очередь

break;

case('2'):

PutFifo(&fifo2, tmp); //кладём её в очередь

}

count++;

}

}

//распределитель задач------------

void distributorTasks(void) {

BaseType \*E;

TInquiry\* tmp;

while (!EmptyFifo(&fifo0)) {

if (cpu.flag) {

GetFifo(&fifo0, E);

tmp = (TInquiry\*)E;

cpu.task = tmp;

cpu.flag = 0;

}

else {

if (cpu.task->P != '0') {

E = cpu.task;

PutStack(stack, E)

}

}

}

if (cpu.flag) {

while (!EmptyFifo(&fifo1)) {

}

if (cpu.flag) {

while (!EmptyFifo(&fifo2)) {

}

}

}

}

//инициализация системы

void systemInit(void) {

InitStack(&stack);

InitFifo(&fifo0, sizeof(TInquiry), 1000);

InitFifo(&fifo1, sizeof(TInquiry), 1000);

InitFifo(&fifo2, sizeof(TInquiry), 1000);

InitCPU(&cpu);

}

//работа системы (возвращает количество решённых задач)

void doSystem(int teak) {

printf("----------Teak #%d------------\n", teak);

//вывод очереди

printf("Fifo:\n");

outputLists();

distributorTasks(); // распределние задач

//вывод стека

printf("Stack:\n");

outputLists();

CountFinishTask += doCPU(&cpu);

}

//система закончила свою работу? (все задачи решены)

int isFinishSystem(void) {

}

Главный алгоритм

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdlib.h"

#include "stdio.h"

#include "System.h"

int main() {

systemInit();

int teak = 0;

do {

generatorTasks();

doSystem(teak++);

} while (!isFinishSystem());

doSystem(teak++);

return 0;

}