МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Белгородский государственный технологический университет им. В.Г. Шухова»

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ Кафедра программного обеспечения вычислительной техники и автоматизированных систем**

Лабораторная работа №6

по дисциплине: Алгоритмы и Структуры Данных

Работу выполнил:

Студент группы ПВ-21:

Донцов Александр Алексеевич

Проверил:

Синюк Василий Григорьевич

Белгород – 2018

Л а б о р а т о р н а я р а б о т а № 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.

**Результаты работы программы**

|  |  |  |
| --- | --- | --- |
| Время | Объекты | Задачи |
| 0 | F1 | (имя,время),(имя,время),. . .,(имя,время) |
| : | : : : |
| FN | (имя,время),(имя,время),. . .,(имя,время) |
| S1 | (имя,время),(имя,время),. . .,(имя,время) |
| : | : : : |
| SM | (имя,время),(имя,время),. . .,(имя,время) |
| P1 | (имя,время) |
| : | : |
| PK | (имя,время) |
| 1 | F1 | (имя,время),(имя,время),. . .,(имя,время) |
| : | **: : :** |
| FN | (имя,время),(имя,время),. . .,(имя,время) |
| S1 | (имя,время),(имя,время),. . .,(имя,время) |
| : | **: : :** |
| SM | (имя,время),(имя,время),. . ., (имя,время) |
| P1 | (имя,время) |
| : | **:** |
| PK | (имя,время) |
| : | **:** | **: : :** |

**Вариант 1**

Стек на ОЛС. Вершина стека – первый элемент ОЛС.

Очередь на ОЛС. «Хвост» очереди — последний, а «голова» — первый элемент ОЛС.

Система состоит из трех процессоров P1, P2, P3, очереди F, стека S и распределителя R (рис.11).

**Реализация на языке C:**

typedef struct TInquiry

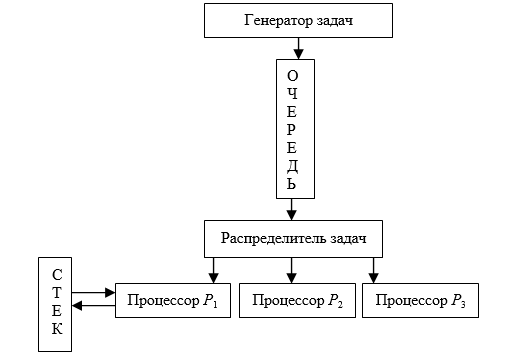
{

char Name[10]; // Имя запроса

unsigned Time; // Время обслуживания

char T; // Тип задачи: 1 — Т1, 2 — Т2, 3 — Т3 };

Поступающие запросы ставятся в очередь. Если в «голове» очереди находится задача Ti и процессор Ti свободен, то распределитель ставит задачу на выполнение в процессор Pi, а если процессор Pi занят, то распределитель отправляет задачу в стек и из очереди извлекается следующая задача. Если в вершине стека находится задача, процессор которой в данный момент свободен, то эта задача извлекается и отправляется на выполнение.



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

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

Характер организованности: линейную структура — последовательность

Характер изменчивости: динамическая

1.1.2. Набор допустимых операций.

1. Инициализация.

2. Включение элемента.

3. Исключение элемента.

4. Чтение текущего элемента.

9. Определение длины списка.

10. Уничтожение.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

Последовательная или связная схема хранения

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

Располагаться ЛС может в статической или динамической памяти.

1.2.4. Характеристику допустимых значений.

CAR(ЛС) = CAR(BaseType)0 + CAR(BaseType)1 +… + CAR(BaseType)max,

где CAR(BaseType) — кардинальное число элемента ЛС типа BaseType, max — максимальное количество элементов в ЛС

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

//базовый тип

typedef void \* BaseType;

// элемент списка

struct element{

BaseType Data; // данные

element\* Next; // указатель на следующий элемент

};

// двухсвязный линейный список

typedef struct {

//указатель на первый элемент списка

element\* first;

//указатель на текущий элемент

element\* current;

} List;

List L;

|  |  |  |
| --- | --- | --- |
| Время | Объекты | Задачи |
| 0 | F | T5 (Type: 3, Time: 3) T4 (Type: 2, Time: 2) T3 (Type: 3, Time: 3) T2 (Type: 2, Time: 2) T1 (Type: 1, Time: 1) |
| S | -------------------------------------------------------------------- |
| P1 | T1 (Type: 1, Time: 1) |
| P2 | T2 (Type: 2, Time: 2) |
| P3 | T3 (Type: 3, Time: 3) |
| 1 | F | T5 (Type: 3, Time: 3) T4 (Type: 2, Time: 2) |
| S | T5 (Type: 3, Time: 3) T4 (Type: 2, Time: 2) |
| P1 | -------------------------------------------------------------------- |
| P2 | T2 (Type: 2, Time: 1) |
| P3 | T3 (Type: 3, Time: 2) |
| 2 | F | -------------------------------------------------------------------- |
| S | T5 (Type: 3, Time: 3) T4 (Type: 2, Time: 2) |
| P1 | -------------------------------------------------------------------- |
| P2 | -------------------------------------------------------------------- |
| P3 | T3 (Type: 3, Time: 1) |
| 3 | F | -------------------------------------------------------------------- |
| S | -------------------------------------------------------------------- |
| P1 | -------------------------------------------------------------------- |
| P2 | T4 (Type: 2, Time: 2) |
| P3 | T5 (Type: 3, Time: 3) |
| 4 | F | -------------------------------------------------------------------- |
| S | -------------------------------------------------------------------- |
| P1 | -------------------------------------------------------------------- |
| P2 | T4 (Type: 2, Time: 1) |
| P3 | T5 (Type: 3, Time: 2) |
| 5 | F | -------------------------------------------------------------------- |
| S | -------------------------------------------------------------------- |
| P1 | -------------------------------------------------------------------- |
| P2 | -------------------------------------------------------------------- |
| P3 | T5 (Type: 3, Time: 1) |
| 6 | F | -------------------------------------------------------------------- |
| S | -------------------------------------------------------------------- |
| P1 | -------------------------------------------------------------------- |
| P2 | -------------------------------------------------------------------- |
| P3 | -------------------------------------------------------------------- |

|  |  |
| --- | --- |
| Входные данные | Работа системы |
|  |  |

**Реализация ОЛС**

ListOLS.h

#pragma once

#ifndef LISTOLS\_H\_

#define LISTOLS\_H\_

// константы ошибок

const short List\_OK = 0;

const short List\_Empty = 1;

const short List\_NotMem = 2;

const short List\_End = 3;

//переменная ошибок

extern short ListError;

//базовый тип

typedef void \* BaseType;

// элемент списка

struct element{

BaseType Data; // данные

element\* Next; // указатель на следующий элемент

};

// двухсвязный линейный список

typedef struct {

//указатель на первый элемент списка

element\* first;

//указатель на текущий элемент

element\* current;

} List;

//конструктор

void ListInit(List \*L);

//деструктор

void ListDone(List \*L);

//добавление элемента после рабочего указателя

void ListPut(List \*L, BaseType E);

//добавление элемента в начало списка

void ListPutFirst(List \*L, BaseType E);

//взятие элемента по рабочему указателю

BaseType ListGet(List \*L);

//взятие элемента по рабочему указателю

BaseType ListRead(List \*L);

// перенос рабочего указателя на следующий элемент

void ListMoveNext(List \*L);

//являеться ли список пустым

int ListEmpty(List \*L);

//находиться ли рабочий указатель в конце списка

int ListEnd(List \*L);

//перенос указателя в конец списка

void ListPtrEnd(List \*L);

//перенос указателя в начало списка

void ListPtrBegin(List \*L);

#endif

ListOLS.cpp

#include "ListOLS.h"

#include "malloc.h"

short ListError;

//очищение памяти элемента

void freeElem(element \*el) { free(el); }

//конструктор

void ListInit(List \*L) {

L->first = NULL;

L->current = NULL;

ListError = List\_OK;

}

//деструктор

void ListDone(List \*L) {//

element \* del;// удаляемый элемент

element \* next; // следующий элемент

//если список не пустой

if (L->first != NULL) {

next = L->first;

do {

del = next;

freeElem(del);

next = next->Next;

} while (next != NULL);

}

L->current = NULL;

L->first = NULL;

}

//создание и инициализация элемента списка

element \*creatElem(BaseType E) {

element \* el = (element \*)malloc(sizeof(element));

//если не удалось выделить память указателю

//возвращаем нулевой указатель

if (el == NULL)

return NULL;

el->Data = E;

el->Next = NULL;

return el;

}

//добавление элемента в начало списка

void ListPutFirst(List \*L, BaseType E) {

element \*el = creatElem(E);

//если не удалось выделить память

if (el == NULL) {

//инициализируем переменную ошибки

ListError = List\_NotMem;

return;

}

//если список пуст

if (ListEmpty(L))

L->first = el;

else {

el->Next = L->first;

L->first = el;

}

ListError = List\_OK;

}

//добавление элемента после рабочего указателя

void ListPut(List \*L, BaseType E) {

//создаем элемент списка

element \*el = creatElem(E);

//если не удалось выделить память

if (el == NULL) {

//инициализируем переменную ошибки

ListError = List\_NotMem;

return;

}

//если список пуст, то вставляем в начало

if (ListEmpty(L)) {

L->first = el;

L->current = el;

}

else if (L->current->Next == NULL) {// если рабочий указатель в конце списка

L->current->Next = el;

L->current = el;

}

else {// если рабочий указатель в середине (общий случай)

el->Next = L->current->Next;

L->current->Next = el;

}

ListError = List\_OK;

}

//взятие элемента по рабочему указателю

BaseType ListRead(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return NULL;

}

return L->current->Data;

}

//взятие(удаление) элемента по рабочему указателю

BaseType ListGet(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return NULL;

}

//запоминаем данные текущего элемента

BaseType data = L->current->Data;

//УДАЛЯЕМ ТЕКУЩИЙ ЭЛЕМЕНТ

//1 ситуация - текущий элемент первый в списке

if (L->current == L->first) {

L->first = L->current->Next;

freeElem(L->current);

L->current = L->first;

} else { //общий случай

element\* previus = L->first; //указатель на предыдущий элемент

while (previus->Next != L->current)

previus = previus->Next;

previus->Next = L->current->Next;

freeElem(L->current);

L->current = previus;

}

ListError = List\_OK;

return data;

}

// перенос рабочего указателя на следующий элемент

void ListMoveNext(List \*L) {

L->current = L->current->Next;

}

//являеться ли список пустым

int ListEmpty(List \*L) {

return (L->first == NULL) ? 1 : 0;

}

//находиться ли рабочий указатель в конце списка

int ListEnd(List \*L) {

return (L->current == NULL) ? 1 : 0;

}

//перенос указателя в конец списка

void ListPtrEnd(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return;

}

L->current = L->first;

while (L->current->Next != NULL)

L->current = L->current->Next;

}

//перенос указателя в начала списка

void ListPtrBegin(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return;

}

L->current = L->first;

}

**Реализация FIFO**

FIFO.h

#pragma once

#ifndef FIFO\_H\_

#define FIFO\_H\_

#include "ListOLS.h"

const short FifoOk = List\_OK;

const short FifoEmpty = List\_Empty;

const short FifoOver = List\_NotMem;

// Переменная ошибок

extern int FifoError;

typedef List Fifo;

//Инициализация очереди

void InitFifo(Fifo \*f);

//Поместить элемент в очередь

void PutFifo(Fifo \*f, void \*E);

//Извлечь элемент из очереди

void \*GetFifo(Fifo \*f);

//Прочитать элемент

void \*ReadFifo(Fifo \*f);

// Проверка, пуста ли очередь?

int EmptyFifo(Fifo \*f);

// Разрушить очередь

void DoneFifo(Fifo \*f);

#endif

FIFO.cpp

#include "FIFO.h"

int FifoError;

//Инициализация очереди

void InitFifo(Fifo \*f) {

ListInit(f);

}

//Поместить элемент в очередь

void PutFifo(Fifo \*f, void \*E) {

//вставка элемента в начало

ListPutFirst(f, E);

}

//Извлечь элемент из очереди

void \*GetFifo(Fifo \*f) {

//перенос рабочего указателя в конец

ListPtrEnd(f);

return ListGet(f);

}

//Прочитать элемент

void \*ReadFifo(Fifo \*f) {

//перенос рабочего указателя в конец

ListPtrEnd(f);

return ListRead(f);

}

// Проверка, пуста ли очередь?

int EmptyFifo(Fifo \*f) {

return ListEmpty(f);

}

// Разрушить очередь

void DoneFifo(Fifo \*f) {

ListDone(f);

}

**Реализация Stack**

Stack.h

#pragma once

#ifndef STACK\_H\_

#define STACK\_H\_

#include "ListOLS.h"

const short StackOk = List\_OK;

const short StackUnder = List\_Empty;

const short StackOver = List\_NotMem;

// Переменная ошибок

extern int StackError;

typedef List Stack;

//Инициализация стека

void InitStack(Stack \*s);

//Поместить элемент в стек

void PutStack(Stack \*s, void \*E);

//Извлечь элемент из стека

void \*GetStack(Stack \*s);

//Прочитать элемент из вершины стека

void \*ReadStack(Stack \*s);

//Проверка: стек пуст?

int EmptyStack(Stack \*s);

//Уничтожить стек

void DoneStack(Stack \*s);

#endif

Stack.cpp

#include "Stack.h"

int StackError;

//Инициализация стека

void InitStack(Stack \*s) { ListInit(s); }

//Поместить элемент в стек

void PutStack(Stack \*s, void \*E) { ListPutFirst(s, E); }

//Извлечь элемент из стека

void \*GetStack(Stack \*s) {

//перенос рабочего указателя в начало списка

ListPtrBegin(s);

return ListGet(s);

}

//Проверка: стек пуст?

int EmptyStack(Stack \*s) { return ListEmpty(s); }

//Прочитать элемент из вершины стека

void \*ReadStack(Stack \*s) {

//перенос рабочего указателя в начало списка

ListPtrBegin(s);

return ListRead(s);

}

//Уничтожить стек

void DoneStack(Stack \*s) { ListDone(s); }

**Реализация Системы**

System.h

#pragma once

#ifndef SYSTEM\_H\_

#define SYSTEM\_H\_

#include "FIFO.h"

#include "Stack.h"

#include "processors.h"

//количество решаемых задач

#define COUNT\_TASK 5

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

extern Fifo fifo;

extern Stack stack;

extern CPU cpu1;

extern CPU cpu2;

extern CPU cpu3;

extern short CountFinishTask;//количество решённых задач

//создание (ввод) задач

TInquiry\* createTask(void);

//уничтожение задачи

void freeTask(TInquiry\* task);

//генератор задач

void generatorTasks(void);

//распределитель задач

void distributorTasks(void);

//инициализация системы

void systemInit(void);

//работа системы (возвращает количество решённыз задач)

void doSystem(int teak);

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

int isFinishSystem(void);

#endif

System.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "System.h"

#include "stdio.h"

#include "malloc.h"

#include "processors.h"

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

Fifo fifo;

Stack stack;

CPU cpu[3];

short CountFinishTask = 0; //количество решённых задач

//вывод списка

void outputList(List \*list) {

TInquiry\* tmp = NULL;

ListPtrBegin(list);

while (!ListEnd(list)) {

tmp = (TInquiry\*)ListRead(list);

printf("%s (Type: %d, Time: %d)\n", tmp->Name, tmp->T, tmp->Time);

ListMoveNext(list);

}

}

//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 type of task (1,2,3): ");

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

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 = 0;// счётчик созданных задач

TInquiry\* tmp;

while (count < COUNT\_TASK) {

tmp = createTask();//создаём задачу

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

count++;}}

//распределитель задач

void distributorTasks(void) {

TInquiry\* tmp = NULL;

//пока очередь не пуста и мы не обработали все процессоры

int count = 0;

while (!EmptyFifo(&fifo) && (count < 3)) {

tmp = (TInquiry\*)GetFifo(&fifo);

//ищим подходящий процессор

for (int i = 0; i < 3; i++) {

if (tmp->T == getTypeTaskCPU(&cpu[i])) // ищем процессор подходящего типа

if (isFreeCPU(&cpu[i])) { //если он свободен

putTaskCPU(&cpu[i], tmp);

break;

} else // закидывам в стек ожидания

PutStack(&stack, tmp);

}

count++;

}

count = 0;

//пока стек не пуст и мы не обработали все процессоры

while (!EmptyStack(&stack) && (count < 3)) {

tmp = (TInquiry\*)GetStack(&stack);

//ищим подходящий процессор

for (int i = 0; i < 3; i++) {

if (tmp->T == getTypeTaskCPU(&cpu[i])) // ищем процессор подходящего типа

if (isFreeCPU(&cpu[i])) { //если он свободен

putTaskCPU(&cpu[i], tmp);

break;

}

else // закидывам в стек ожидания

PutStack(&stack, tmp);

}

count++;

}

}

//инициализация системы

void systemInit(void) {

InitStack(&stack);

InitFifo(&fifo);

for (int i = 0; i < 3; i++)

InitCPU(&cpu[i], i + 1);

generatorTasks(); //генерация задачи

}

//работа системы (возвращает количество решённыз задач)

void doSystem(int teak) {

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

//вывод очереди

printf("Fifo:\n");

outputList(&fifo);

distributorTasks(); // распределние задач

//вывод стека

printf("Stack:\n");

outputList(&stack);

for (int i = 0; i < 3; i++)

CountFinishTask += doCPU(&cpu[i]);

}

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

int isFinishSystem(void) {

return (CountFinishTask == COUNT\_TASK) ? 1 : 0;

}

**Структуры**

#pragma once

#ifndef STRUCTS\_H\_

#define STRUCTS\_H\_

typedef char TypeTask;

//задача

typedef struct TInquiry {

char Name[10]; // Имя запроса

unsigned Time; // Время обслуживания

TypeTask T; // Тип задачи: 1 — Т1, 2 — Т2, 3 — Т3

};

//структура процессора

typedef struct CPU {

int flag; // флаг - CPU занят или свободен

TypeTask T; // тип задачи, которую может решать процессор

TInquiry \*task; // решаемая задача в данный момент

};

#endif

**Процессор**

processors.h

#pragma once

#ifndef PROCESSORS\_H\_

#define PROCESSORS\_H\_

#include "Structs.h"

//инициализация процессора

void InitCPU(CPU \*p, TypeTask T);

//свободен ли процессор?

int isFreeCPU(CPU\* p);

//возвращает тип задачи, которую может решать процессор

TypeTask getTypeTaskCPU(CPU\* p);

//положить задачу в процессор

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

//делать вычисления

int doCPU(CPU \*);

#endif

processors.cpp

#include "processors.h"

#include "stdlib.h"

#include "stdio.h"

#include "System.h"

//инициализация процессора

void InitCPU(CPU \*p, TypeTask T) {

p->flag = 1;

p->T = T;

p->task = NULL;

}

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

int isFreeCPU(CPU \*p) {

return p->flag;

}

//возвращает тип задачи, которую может решать процессор

TypeTask getTypeTaskCPU(CPU\* p) {

return p->T;

}

//положить задачу в процессор

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

p->task = task;

p->flag = 0;

}

//делать вычисления

int doCPU(CPU\* p) {

printf("CPU #%d: ", p->T);

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

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

} else {

printf("%s (Type: %d, Time: %d)\n", p->task->Name, p->task->T, p->task->Time);

p->task->Time--;

//если задача решена, то освободить процессор

if (p->task->Time <= 0) {

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

freeTask(p->task);

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

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

return 1;

}

}

return 0;

}

**Главная программа**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdlib.h"

#include "stdio.h"

#include "System.h"

int main() {

systemInit();

int teak = 0;

do {

doSystem(teak++);

} while (!isFinishSystem());

doSystem(teak++);

return 0;

}