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

Дисциплина: Алгоритмы и структуры данных

Лабораторная работа № 6

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

|  |  |
| --- | --- |
|  | Выполнила: ст. группы ПВ-21  Зановская А.И.  Проверил: Синюк В.Г. |

Белгород

2017

**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:

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 ин-терпретации.

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам

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.

**Задание 1**

1. Абстрактный уровень представления СД:
   1. Характер организованности и изменчивости.

Стек: Линейная структура, последовательность.

Очередь: Линейная структура, последовательность.

* 1. Набор допустимых операций

Стек: Инициализация. Включение элемента. Исключение элемента. Чтение элемента. Проверка пустоты стека. Уничтожение стека.

Очередь: Инициализация. Включение элемента. Исключение элемента. Проверка пустоты очереди. Уничтожение очереди.

1. Физический уровень представления СД:
   1. Схему хранения.

Стек: Последовательная или связная.

Очередь: Последовательная или связная.

* 1. Объем памяти, занимаемый экземпляром СД.

Стек: Зависит от реализации.

Очередь: Зависит от реализации.

* 1. Формат внутреннего представления СД и способ его интерпретации.

Стек: Может быть реализован в динамической/статической памяти. ПЛС, ОЛС.

Очередь: Может быть реализован в динамической/статической памяти. ПЛС, ОЛС.

* 1. Характеристику допустимых значений.

Стек: Зависит от реализации.

Очередь: Зависит от реализации.

* 1. Тип доступа к элементам.

Стек: Последовательный.

Очередь: Последовательный.

1. Логический уровень представления СД.

Стек: Stack S;

Очередь: Fifo F;

Вариант 10

Система состоит из двух процессоров P1 и P2 , трех очередей F2, F3, F3 и стека (см. рис.12). В систему поступают запросы. Запрос можно представить записью

**на языке C:**

typedef struct TInquiry

{

char Name[10]; // имя запроса

unsigned Time; // время обслуживания

char P; /\* приоритет задачи: 0 — высший,

1 — средний, 2 — низший \*/

};

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди F2. Задача из очереди F2 поступает в свободный процессор P1 или P2, если оба свободны, то в P1. Если очередь F2 пуста, то обрабатываются задачи из очереди F3. Задача из очереди F3 поступает в свободный процессор P1 или P2, если оба свободны, то в P1. Если очереди F2 и F3 пусты, то обрабатываются задачи из очереди F3. Задача из очереди F3 поступает в свободный процессор P1 или P2, если оба свободны, то в P2. Если процессоры заняты и поступает задача с более высоким приоритетом, чем обрабатываемая в одном из процессоров, то задача из процессора помещается в стек, а поступающая — в процессор. Задача из стека поступает в один из освободившихся процессоров, если все задачи с более высоким приоритетом уже обработаны.

P1

Стек

P2

О

Ч

Е

Р

Е

Д

Ь

#### F2

О

Ч

Е

Р

Е

Д

Ь

#### F3

О

Ч

Е

Р

Е

Д

Ь

#### F3

Генератор задач

**Стек как отображение на массив**

Заголовочный файл

#ifndef \_STACK\_ARRAY\_H\_INCLUDED

#define \_STACK\_ARRAY\_H\_INCLUDED

const short StackSize = 100;

const short stackOk = 0;

const short stackFull = 1;

const short stackEmpty = 2;

extern short StackError = 0;

typedef int BaseType;

typedef struct

{

BaseType buf[StackSize];

short uk;

} Stack;

void StackInit (Stack \*S);

void StackPut (Stack \*S, BaseType E);

void StackGet (Stack \*S, BaseType \*E);

int StackFull (Stack \*S);

int StackEmpty (Stack \*S);

#endif

**Функции для работы со стеком**

#include <stdio.h>

#include "Stack\_Array.h"

int StackError = 0;

//Инициализация стека

void StackInit(Stack \*S)

{

S->uk = 0;

StackError = stackOk;

}

//Включение в стек

void StackPut(Stack \*S, BaseType E)

{

if (StackFull(S))

return;

S->buf[S->uk] = E;

S->uk++;

}

//Исключение из стека

void StackGet(Stack \*S, BaseType \*E)

{

if (StackEmpty(S))

return;

\*E = S->buf[S->uk - 1];

S->uk--;

}

//Стек полон

int StackFull(Stack \*S)

{

return (S->uk == StackSize);

}

//Стек пуст

int StackEmpty(Stack \*S)

{

return (S->uk == 0);

}

**Очередь как отображение на массив**

Заголовочный файл

#ifndef \_FIFOARRAY\_H\_INCLUDED

#define \_FIFOARRAY\_H\_INCLUDED

const short FifoSize = 100;

const int fifoOk = 0;

const int fifoFull = 1;

const int fifoEmpty = 2;

extern short FifoError;

typedef int BaseType;

typedef struct

{

BaseType buf[FifoSize];

int uk1; // Указатель на хвост

int uk2; // Указатель на голову

int n; // Количество элементов в очереди

} Fifo;

void FifoInit (Fifo \*F);

void FifoPut (Fifo \*F, BaseType E);

void FifoGet (Fifo \*F, BaseType \*E);

int FifoEmpty (Fifo \*F);

int FifoFull (Fifo \*F);

#endif

**Функции для работы с очередью**

#include <stdio.h>

#include "FifoArray.h"

int FifoError;

//Инициализация очереди

void FifoInit (Fifo \*F)

{

F->uk1 = 0;

F->uk2 = 0;

F->n = 0;

FifoError = fifoOk;

}

//Включение в очередь

void FifoPut(Fifo \*F, BaseType E)

{

if (FifoFull(F))

return;

F->buf[F->uk2] = E;

F->uk2 = (F->uk2 + 1) % FifiSize;

F->n++;

}

//Исключение из очереди

void FifoGet (Fifo \*F, BaseType \*E)

{

if (FifoEmpty(F))

return;

\*E = F->buf[F->uk1];

F->uk1 = (F->uk1 + 1) % FifoSize;

F->n--;

}

//Очередь полна

int FifoFull(Fifo \*F) {

return (F->n == FifoSize);

}

//Очередь пуста

int FifoEmpty(Fifo \*F){

return (F->len == 0);

}

Основная программа

void generate (TInquiry \*pa, int n)

{

int i=0;

for (i=0;i<n;i++)

{

printf ("Name: ");

scanf ("%s",pa[i].Name);

printf ("Time: ");

scanf ("%d",&pa[i].Time);

printf ("Priority: ");

scanf ("%s",&pa[i].P);

printf("\n");

}

}

void FifoSort (TInquiry \*pa, Fifo \*F1,Fifo \*F2,Fifo \*F3, int n)

{

int i;

for (i=0; i<n; i++)

{

if (pa[i].P == 0)

FifoPut(F1, pa[i]);

if (pa[i].P == 1)

FifoPut(F2, pa[i]);

if (pa[i].P == 2)

FifoPut(F3, pa[i]);

}

}

int main()

{

srand(time(NULL));

int n=3, i=0; printf("n >> "); //количество запросов

scanf("%d", &n);

TInquiry P1, P2, tmp;

TInquiry \*pa = (TInquiry\*)calloc(n, sizeof(TInquiry)); //выделение памяти на массив

generate(pa, n); //генерация массива запросов

Fifo F1, F2, F3; //Объявление и инициализация очередей

FifoInit(&F1);

FifoInit(&F2);

FifoInit(&F3);

Stack S; //Объявление и инициализация стеков

StackInit(&S);

FifoSort(pa, &F1, &F2, &F3, n); //распределили запросы по очередям

while (StackEmpty(&S) ||!FifoEmpty(&F1) || !FifoEmpty(&F2) || !FifoEmpty(&F3) || P1.Time!=0 || P2.Time!=0)

{

if (!FifoEmpty(&F1)) //если нулевая очередь не пуста

{

FifoGet(&F1,&tmp);

if (P1.Time==0 && P2.Time == 0) //процессоры пусты

P1=tmp;

if (P1.Time!=0 && P2.Time==0) //пуст второй

P2=tmp;

if (P1.Time==0 && P2.Time!=0) //пуст первый, не пуст второй

P1=tmp;

if (P1.Time!=0 && P2.Time!=0 && tmp.P<P1.P)

{

StackPut(&S, P1); //поместили в стек то что было

P1=tmp; //записали новый процесс

}

else if (P1.Time!=0 && P2.Time!=0 && tmp.P<P2.P)

{

StackPut(&S, P2); //поместили в стек то что было

P2=tmp; //записали новый процесс

}

}

else

{

if (P1.Time!=0)

P1.Time--;

else

{

if(!StackEmpty(&S))

StackGet(&S, &P1);

else

if (!FifoEmpty(&F1))

FifoGet(&F1, &P1);

}

if (P2.Time!=0)

P2.Time--;

else

{

if(!StackEmpty(&S))

StackGet(&S, &P2);

else

if (!FifoEmpty(&F1))

FifoGet(&F1, &tmp);

}

}

if (!FifoEmpty(&F2))

{

FifoGet(&F2,&tmp);

if (P1.Time==0 && P2.Time == 0)

P1=tmp;

if (P1.Time!=0 && P2.Time==0)

P2=tmp;

if (P1.Time==0 && P2.Time!=0)

P1=tmp;

if (P1.Time!=0 && P2.Time!=0 && tmp.P<P1.P)

{

StackPut(&S, P1);

P1=tmp;

}

else

if (P1.Time!=0 && P2.Time!=0 && tmp.P<P2.P)

{

StackPut(&S, P2);

P2=tmp;

}

}

else

{

if (P1.Time!=0)

P1.Time--;

else

{

if(!StackEmpty(&S))

StackGet(&S, &P1);

else

if (!FifoEmpty(&F2))

FifoGet(&F2, &tmp);

}

if (P2.Time!=0)

P2.Time--;

else

{

if(!StackEmpty(&S))

StackGet(&S, &P2);

else

if (!FifoEmpty(&F2))

FifoGet(&F2, &tmp);

}

}

if (!FifoEmpty(&F3))

{

FifoGet(&F3,&tmp);

if (P1.Time==0 && P2.Time == 0)

P2=tmp;

if (P1.Time!=0 && P2.Time==0)

P2=tmp;

if (P1.Time==0 && P2.Time!=0)

P1=tmp;

if (P1.Time!=0 && P2.Time!=0 && tmp.P<P1.P)

{

StackPut(&S, P1);

P1=tmp;

}

else

if (P1.Time!=0 && P2.Time!=0 && tmp.P<P2.P)

{

StackPut(&S, P2);

P2=tmp;

}

}

else

{

if (P1.Time!=0)

P1.Time--;

else

{

if(!StackEmpty(&S))

StackGet(&S, &P1);

else

if (!FifoEmpty(&F3))

FifoGet(&F3, &tmp);

}

if (P2.Time!=0)

P2.Time--;

else

{

if(!StackEmpty(&S))

StackGet(&S, &P2);

else

if (!FifoEmpty(&F3))

FifoGet(&F3, &tmp);

}

}

}

return 0;

}

В систему поступает 5 запросов:

|  |  |  |
| --- | --- | --- |
| Время | Объекты | Задачи |
| 0 | F1 | (Name1, 4) |
| F2 |  |
| F3 |  |
| S |  |
| P1 |  |
| P2 |  |
| 1 | F1 |  |
| F2 | (Name2, 2) |
| F3 |  |
| S |  |
| P1 | (Name1, 3) |
| P2 |  |
| 2 | F1 |  |
| F2 |  |
| F3 | (Name3, 2) |
| S |  |
| P1 | (Name1, 2) |
| P2 | (Name2, 1) |
| 3 | F1 |  |
| F2 |  |
| F3 | (Name4, 2) |
| S |  |
| P1 | (Name1, 1) |
| P2 | (Name3, 1) |
| 4 | F1 |  |
| F2 | (Name5, 3) |
| F3 |  |
| S | (Name4, 1) |
| P1 |  |
| P2 |  |
| 5 | F1 |  |
| F2 |  |
| F3 |  |
| S |  |
| P1 | (Name5, 2) |
| P2 | (Name4, 1) |
| 6 | F1 |  |
| F2 |  |
| F3 |  |
| S |  |
| P1 | (Name5, 1) |
| P2 |  |
| 7 | F1 |  |
| F2 |  |
| F3 |  |
| S |  |
| P1 |  |
| P2 |  |