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-211

Чувилко Илья Романович

Проверил:

Синюк Василий Григорьевич

Белгород 2022 г.

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

Задание

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.

Выполнение работы:

Номер варианта	Номер модуля для стека	Номер модуля для очереди	Номер задачи
23	1	3	4

Реализация стека:

Содержимое заголовочного файла:

```
#ifndef CODE_STACK_H
#define CODE_STACK_H

#include <malloc.h>

const int StackSize = 1000;
const int StackOk = 0; // Все ОК
const int StackOver = 1; // Стек переполнен
const int StackUnder = 2; // Стек пуст
int StackError; // Переменная ошибок
typedef int BaseType; // Определить тип элемента стека

typedef struct Stack
```

```
{
  int Buf[StackSize];
  unsigned ptr; // Указывает на элемент, являющийся вершиной стека
};

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


#endif //CODE_STACK_H
```

Содержимое исполняемого файла:

```
#include "Stack.h"

// Инициализация стека
void InitStack(Stack *s) {
  s->ptr = 0;
}

// Проверка: стек пуст?
int EmptyStack(Stack *s) {
  return s->ptr == 0;
}

void PutStack(Stack *s, int x) {
  if (s->ptr > StackSize) {
    StackError = StackOver;
    return;
  }

  s->Buf[++s->ptr] = x;
  StackError = StackOk;
}

// Извлечь элемент из стека
void GetStack(Stack *s, BaseType *E) {
  if (EmptyStack(s)) {
    StackError = StackUnder;
    return;
  }

  --s->ptr;
  StackError = StackOk;
}

// Прочитать элемент из вершины стека
void ReadStack(Stack *s, BaseType *E) {
  *E = s->Buf[--(s->ptr)];
  StackError = StackOk;
}
```

Реализация очереди:

Содержимое заголовочного файла:

```
#ifndef CODE_QUEUE_H
#define CODE_QUEUE_H

#include "../../../Lab 5/code/List7/List7.h"  // Смотреть лаб. раб. №5

const int FifoOk = LIST_OK;
const int FifoUnder = ListUnder;
const int FifoOver = LIST_NOT_MEM;
int FifoError; // Переменная ошибок

typedef List Fifo;

void InitFifo(Fifo *f); // 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?
void DoneFifo(Fifo *f); // Разрушить очередь

#endif //CODE_QUEUE_H
```

Содержимое исполняемого файла:

```
#include "Queue.h"

#include <malloc.h>

void InitFifo(Fifo *f, unsigned SizeMem) {
  InitList(f, SizeMem);
}

void PutFifo(Fifo *f, BaseType E) {
  EndList(f);
  PutList(f, E);
  f->ptr++;
  FifoError = ListError;
}

void GetFifo(Fifo *f, BaseType *E) {
  EndList(f);
  GetList(f, E);
  f->ptr--;
  FifoError = ListError;
}

void ReadFifo(Fifo *f, BaseType *E) {
  *E = f->PMemList[f->ptr];
}

int EmptyFifo(Fifo *f) {
  return f->N == 0;
}
```

```
void  DoneFifo(Fifo *f) {
   DoneList(f);
}
```

Выполнение задачи 4

Условие:

Система состоит из двух процессоров P1 и P2, трех очередей F0, F1, F2 и двух стеков S1 и S2 (рис.10).

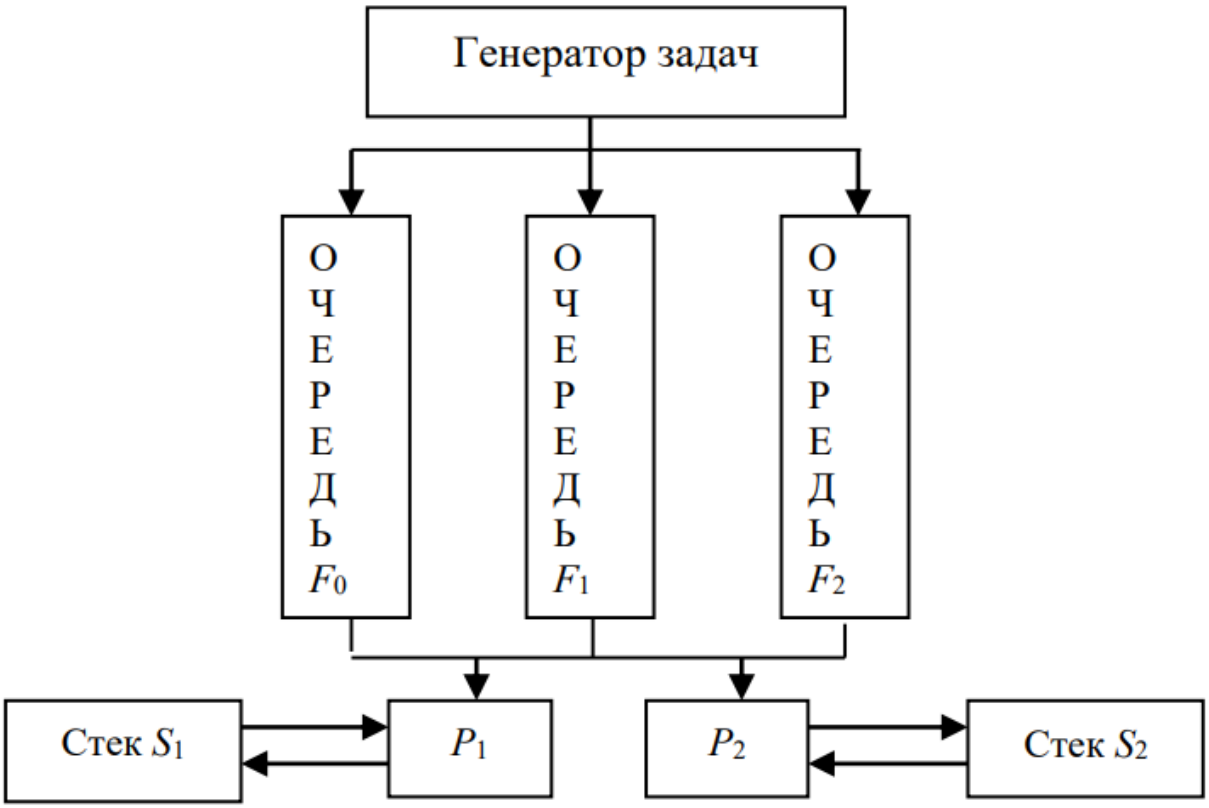

Рис.10. Система задчи 4

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди **F0**. Задача из очереди **F0** поступает в свободный процессор **P1** или **P2**, если оба свободны, то в **P1**. Если очередь **F0** пуста, то обрабатываются задачи из очереди **F1**. Задача из очереди **F1** поступает в свободный процессор **P1** или **P2**, если оба свободны, то в **P1**. Если очереди **F0** и **F1** пусты, то обрабатываются задачи из очереди **F2**. Задача из очереди **F2** поступает в свободный процессор **P1** или **P2**, если оба свободны, то в **P1**. Если процессоры заняты и поступает задача с более высоким приоритетом, чем обрабатываемая в одном из процессоров, то задача из процессора помещается в соответствующий стек, а поступающая — в процессор. Задача из стека помещается в соответствующий процессор, если он свободен, и очереди с задачами более высокого приоритета пусты.

```
#include <iostream>

#include "Stack/Stack.h"
#include "Queue/Queue.h"

 struct TInquiry
```

```
{
  char Name[10]; // имя запроса
  unsigned Time; // время обслуживания
  char P; /* приоритет задачи: 0 — высший,
 1 — средний, 2 — низший */
} typedef TInquiry;

void InputString(char *a, int size) {
  char c = 0;
  for (int i = 0; i < size; i++) {
    scanf("%c", &c);
    if (c == ' ') {
      a[i] = '\0';
      return;
    }
    a[i] = c;
  }
}

void CheckFreeProcessor(int *P, Fifo *f0, Fifo *f1, Fifo *f2) {
  if (!EmptyFifo(f0))
    GetFifo(f0, P);
  else if (!EmptyFifo(f1))
    GetFifo(f1, P);
  else if (!EmptyFifo(f2))
    GetFifo(f2, P);
  else {
    printf("\n\nThe work of the program is over");
    exit(1);
  }
}

void OutputStatus() {
  printf("Processors status: %d %d");

}

// Базовый тип будет: TInquiry
int main() {
  Fifo f0, f1, f2;
  InitFifo(&f0);
  InitFifo(&f1);
  InitFifo(&f2);
  int n; // Количество входных запросов
  printf("Enter the number of input requests");
  scanf("%d", &n);

  // Ввод задач
  for (int i = 0; i < n; i++) {
    TInquiry T;
    InputString(T.Name, 10);
    scanf("%d %c", &T.Time, &T.P);

    switch (T.P) {
      case 0:
        PutFifo(f0, T);
        break;
      case 1:
        PutFifo(f1, T);
        break;
```

```
        case 2:
          PutFifo(f2, T);
          break;
      }
    }

    // Выводятся состояние задач на момент окончания работы одного из процессоров
    int P0 = 0, P1 = 0;
    for (int i = 0; i < n; i++) {
      if (P0 == 0) {
        CheckFreeProcessor(&P0, &f0,&f1,&f2);
      } else if (P1 == 0) {
        CheckFreeProcessor(&P1, &f0,&f1,&f2);
      }

      int timeToNextStep = std::min(P0, P1);
      P0 -= timeToNextStep;
      P1 -= timeToNextStep;
    }

    return 0;
}
```

Вывод: в ходе работы изучены СД типа «линейный список», получены навыки их программной реализации и использования.