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»

по теме: **«Структуры данных «стек» и «очередь» (C)»**

Выполнил/а: ст. группы ПВ-231
Чупахина София Александровна
Проверил:
Акиньшин Даниил Иванович

Белгород, 2024

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

Задания:

1. Для СД типа «стек» и «очередь» определить:
   (a) Абстрактный уровень представления СД:
      i. Характер организованности и изменчивости,
      ii. Набор допустимых операций.
   (b) Физический уровень представления СД:
      i. Схему хранения;
      ii. Объем памяти, занимаемый экземпляром СД;
      iii. Формат внутреннего представления СД и способ его интерпретации;
      iv. Характеристику допустимых значений;
      v. Тип доступа к элементам.
   (c) 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 $0, 1, 2, \ldots, N$. Во втором — для каждого момента времени указываются имена объектов (очереди — $F_1, F_2, \ldots, F_N$; стеки — $S_1, S_2, \ldots, S_M$; процессоры — $P_1, P_2, \ldots, P_K$), а в третьем — задачи (имя, время), находящиеся в объектах.

Содержание

Задание 1:

Опишем СД типа «стек» и «очередь».

Абстрактный уровень:

Задание 1.1.1:

Как СД типа «стек», так и СД типа «очередь» являются линейными СД. Как и рассматриваемые ранее СД типа «массив» и «линейный список», они отображают некую упорядоченную последовательность элементов, и для каждого из них, кроме первого и последнего, существует предыдущий и следующий. СД типа «стек» и «очередь» могут быть реализованы для любого типа элементов. Специфика СД типа «стек» и «очередь» в том, что для них невозможна перезапись и чтение элемента по произвольному индексу (как для массива), а также включение и исключение элемента на произвольную позицию (как для линейных списков). В СД типа стек включать, исключать и считывать элемент можно только с одной определенной позиции, которая называется вершиной стека. Таким образом, последний включенный элемент будет ближе всего к вершине стека, из которой проводится исключение, и потому при исключении он будет извлечен первым. Это свойство дало стеку сокращенное название LIFO (от англ. Last In, First Out). В СД типа очередь элементы всегда включаются с одной позиции, которая называется хвостом очереди, и исключаются с другой позиции, которая называется головой очереди. Обычно роль вершины, головы или хвоста отдается первому или последнему элементу массива. Только что включенный с хвоста элемент отделяется от головы всеми теми элементами, которые были включены в список до него. По аналогии со стеком, очередь имеет указывающее на это свойство сокращенное название FIFO (от англ. First In, First Out).

«Стек» и «очередь» являются динамическими СД: несмотря на невозможность перезаписи и осуществление включения и исключения только с одной позиции (эти позиции совпадают для стека и не совпадают для очереди) в нем могут меняться как сами элементы, так и их количество. Количество элементов ограничено только способом реализации структуры; этот аспект подробнее рассмотрим в задании 1.2.

Задание 1.1.2:

СД типа «стек» и «очередь» не реализованы в языках программирования по умолчанию, то есть являются производными структурами данных. Тем не менее, для каждой из них существуют операции, которые должны быть реализованы по определению. Для СД типа «стек» это операции:

1. Инициализация.
2. Включение элемента.
3. Исключение элемента.
4. Чтение элемента.
5. Проверка пустоты стека.

6. Уничтожение стека.

А для СД типа «очередь» — следующие операции:

1. Инициализация.
2. Включение элемента.
3. Исключение элемента.
4. Проверка пустоты очереди.
5. Уничтожение очереди.

Физический уровень:

Задание 1.2.1:

Для СД типа «стек» и «очередь» может использоваться как прямоугольная, так и связная схема хранения. При использовании прямоугольной схемы для хранения элементов стека или очереди используется массив; тогда последовательности битов, представляющие нулевой, первый, второй и так далее элемент, идут друг за другом по возрастанию индекса подряд, без разделителей. Точно так же для реализации стека или очереди может использоваться связная схема хранения — тогда элементы этих СД будут храниться в связном списке. В таком случае значение каждого элемента включено в структуру типа «запись», которая хранит также указатель на следующий элемент (для односвязного списка) или указатели на следующий и на предыдущий элемент (для двусвязного списка). В любом случае, над СД типа «стек» и «очередь» реализуются также и функции, ограничивающие доступ к элементами массива или списка по умолчанию и обеспечивающие выполнение ключевых условий: включение, исключение или чтение элемента только с вершины стека; включение элемента только с хвоста и исключение только с головы для очереди.

Задание 1.2.2:

Количество памяти, занимаемой СД типа «стек» или «очередь», зависит от того, какое количество памяти занимает его базовый тип, есть ли необходимость хранить дополнительную информацию (указатели на другие элементы для связной схемы хранения) о каждом элементе, и сколько элементов содержит СД. Если базовый тип занимает T байт, для каждого элемента необходимо также хранить P (P также может быть равно 0) указателей типа *unsigned long*, занимающего 8 байт, а список содержит K элементов, то количество памяти, занимаемой экземпляром любой из этих СД, будет равно $(T + P) * K$ байт.

Задание 1.2.3:

Если СД типа «стек» или «очередь» с длиной K использует последовательную схему хранения, то значение этой структуры хранится как K идущих подряд последовательностей битов, кодирующих значения элементов СД в соответствии с правилами для данного базового типа.

Для связных же вариантов реализации каждый узел (структура с одним полем «значение элемента» и несколькими полями «указатели») хранится независимо друг от друга, и значения в полях узла также могут храниться в памяти компьютера не подряд (либо в порядке, отличном от заданного). Значение элемента переводится в двоичный код в зависимости от базового типа; указатели кодируются как положительные целые числа — переводятся в двоичную систему счисления и дополняются нулями слева до размера в 8 байт.

Задание 1.2.4:

Диапазоны допустимых значений СД типа «стек» и «очередь» связаны с диапазонами допустимых значений их базовых элементов. Если рассматривать стек или очередь некоторой фиксированной длины K, то количество возможных значений для экземпляров этих СД равно возведенному в степень K количеству возможных значений для их базового элемента. Соответственно, для СД типа «стек» или «очередь» количество возможных значений равно сумме количеств возможных значений этих СД с длинами от 0 до max, где max — максимальная длина стека или очереди (однозначно определенная в случаях, когда эти СД реализованы на массиве). Обозначая количество допустимых значений как CAR (кардинальное число), получим формулу $CAR(STACK) = CAR(FIFO) = CAR(BaseType)^0 + CAR(BaseType)^1 + \ldots + CAR(BaseType)^max$.

Задание 1.2.5:

СД типа «стек» и «очередь» имеют ограниченный доступ к элементам. Для стека чтение осуществляется только с его вершины, для очереди чтение не осуществлено вовсе, и узнать значение элемента можно только после его исключения из нее. Соответственно, чтобы получить некоторый элемент стека, нужно исключить из него все элементы, которые отделяют его от вершины, а чтобы получить некоторый элемент очереди, нужно исключить из нее все элементы, которые отделяют его от головы. С этой точки зрения логичнее будет сказать, что доступ к элементам в этих СД последователен.

Логический уровень:

СД типа «стек» и «очередь» не являются встроенными, и потому сначала необходимо реализовать их, выбрав один из способов отображения. Только после этого их можно описать на логическом уровне (представить на языке программирования). Приведем здесь описания для той реализации, которая будет выполнена в задании 2 этой лабораторной работы. Стек отображается на последовательный линейный список, и его вершиной является первый элемент этого списка; кольцевая очередь отображается на массив в динамической памяти.

Индивидуальное задание; вариант 21

Модуль 10 (для стека): Стек на ПЛС. Вершина стека – первый элемент ПЛС.

Реализация на языке С:

```
#if !defined(__STACK10_H)
#define __STACK10_H
```

```
#include "list6.h"// Смотреть лаб. раб. №5
const StackOk = ListOk;
const StackUnder = ListUnder;
const StackOver = ListNotMem;
int StackError; // Переменная ошибок
typedef List Stack;
void InitStack(Stack *s, unsigned Size); /* Инициализация стека */
void PutStack(Stack *s, void *E); // Поместить элемент в стек
void GetStack(Stack *s, void *E); // Извлечь элемент из стека
int EmptyStack(Stack s); // Проверка: стек пуст?
void ReadStack(Stack s, void *E); /* Прочитать элемент из вершины стека */
void DoneStack(Stack *s); // Уничтожить стек
#endif
```

Модуль 11 (для очереди): Очередь (кольцевая) на массиве в динамической памяти.

```
#if !defined(__FIFO11_H)
#define __FIFO11_H
const FifoOk = 0;
const FifoUnder = 1;
const FifoOver = 2;
int FifoError; // Переменная ошибок
typedef void *BaseType;
typedef struct Fifo
{
BaseType *Buf;
unsigned SizeBuf; // Максимальная длина очереди
unsigned SizeEl; // Размер элемента очереди
unsigned Uk1; // Указатель на «голову» очереди
unsigned Uk2; // Указатель на «хвост» очереди
unsigned N; // Количество элементов очереди
};
void InitFifo(Fifo* f, unsigned SizeEl, unsigned SizeBuf);
// Инициализация очереди
void PutFifo(Fifo *f, void *E); /* Поместить элемент в очередь */
void GetFifo(Fifo *f, void *E); // Извлечь элемент из очереди
void ReadFifo(Fifo *f, void *E); // Прочитать элемент
int EmptyFifo(Fifo *f); // Проверка, пуста ли очередь?
```

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

#endif

Задача 2: Система состоит из процессора P, трех очередей F_0, F_1, F_2 и стека S. В систему поступают запросы. Запрос можно представить записью.

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 F_0. Если она пуста, можно обрабатывать задачи из очереди F_1. Если и она пуста, то можно обрабатывать задачи из очереди F_2. Если все очереди пусты, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если обрабатывается задача с низшим приоритетом и поступает задача с более высоким приоритетом, то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны.

Задание 2:

Разделим код выполнения этого задания на два файла: заголовочный и реализации. В заголовочном файле зададим константы с кодами трех основных ошибок, которые могут возникнуть в ходе работы со списками: ListNotMem — для выделения места под новый элемент нет места в списке свободных элементов в массиве MemList; ListUnder — попытка получить доступ к элементу, в то время как пуст либо список, либо текущий указатель; ListEnd — в ходе перебора элементов списка был достигнут его конец. Под хранение кода ошибки отводится переменная. После этого дадим название используемым типам данных: BaseType — тип, элементы которого хранит список (в нашем случае это целые числа int); ptrel — беззнаковое целое число, индекс элемента массива MemList, где хранится некоторый элемент; element — запись, состоящая из значения элемента и индекса, по которому хранится следующий элемент; List — структура, хранящая индексы начального и текущего элемента, а также их общее количество. Только после этого будут объявлены прототипы функций для работы со списками.

Реализация функций будет вынесена в отдельный файл со следующим содержанием.

Примечание: функция PutList подразумевает, что элемент вставляется после элемента, на который указывает рабочий указатель, поэтому, чтобы вставить элемент в самое начало списка, рабочий указатель должен быть равен нулю, при условии, что нулю не равен стартовый указатель (если же стартовый указатель равен 0, это запускает отдельный алгоритм вставки в пустой список).

Для нормальных (не вызывающих ошибки и аварийного завершения) сценариев большинства функций (за исключением тех, что работают не со списками, а непосредственно с массивом MemList, на основе которого они хранятся, а также уничтожающей список функции DoneList),

можно составить автоматизированные тесты и вынести их в отдельный файл тестирования. Будем тестировать функции по порядку; после того, как они протестированы, их можно использовать в тестах дальнейших функций.

Запустив программу, можем самостоятельно убедиться, что все тесты прошли успешно:

Задание 3:

Теперь, когда все функции, обязательные для СД типа «линейный список», реализованы, можем решить задачу, в ходе которой нам предстоит работать с линейными списками. Прототип функции, решающей задачу добавим в заголовочный файл, затем вставим основной код в файл реализации, а в файле с автоматизированными тестами рассмотрим несколько сценариев: список содержит один элемент (будем считать, что тогда он удовлетворяет условию); список содержит несколько элементов, и шаг между ними одинаков и соответствует заданному; список содержит несколько элементов, но на одном из них шаг отклоняется от заданного, и часть последовательности после него перестает соответствовать условию.

Примечание: случай, когда последовательность пуста, будем считать ошибкой: программа аварийно завершает выполнение с кодом ListUnder.

Запустив программу, можем убедиться, что и этот тест полностью пройден:

Вывод:

В ходе лабораторной работы дали характеристику СД типа «стек» и «очередь», форматам их представления, реализовали по одному из них для каждой СД из них (стек на базе последовательного линейного списка с вершиной в первом элементе и кольцевую очередь на базе динамического массива), написали ряд базовых функций для работы со стеками и очередями в этом формате.