Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

Отчет по лабораторной работы №4.

Студент : **Шавиш Тарек**

Группа : **ИУ7И-31Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Руководитель **_____СилантьеваА.В.**

2023

Условие задачи

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека. Реализовать стек:

а) массивом;

б) списком.

Все стандартные операции со стеком должны быть оформлены подпрограммами.

При реализации стека списком в вывод текущего состояния стека добавить просмотр адресов элементов стека и создать свой массив свободных областей (адресов освобождаемых элементов) с выводом его на экран.

Вывести результаты сравнения двух стеков (кол-во занимаемой памяти, время выполнения операций)

Техническое задание

Используя операции со стеком реализовать поиск пути в лабиринте, который представлен матрицей.

Входные данные

Пункты меню, количество добавляемых и удаляемых элементов, карта с лабиринтом, введенный элемент стека.

Выходные данные

Текущее состояние стека, массив свободных областей, карта с лабиринтом и найденный путь в ней, результаты сравнения двух стеков.

Возможные аварийные ситуации

Некорректный ввод, невозможность найти путь в лабиринте.

Структуры данных

Структура хранения лабиринта

```
typedef struct maze
{
   char **matrix;
   int x;
   int y;
   int i_enter;
   int j_enter;
} maze_t;
```

matrix – указатель на матрицу, хранящую сам лабиринт
x, y – количество столбцов и строк матрицы
i_enter, j_enter – индексы, откуда начинается поиск пути

Структура стека, реализованного связанным списком:

```
typedef struct list_element
{
   int i;
   int j;
   int direction;
   struct list_element *next_elem;
} list_element_t;
```

i, j – индексы клетки, из которой можно продолжить путь
direction – направление пути
next_elem – указатель на следующий элемент связанного списка

```
typedef struct list
{
   list_element_t *ptr;
} stack_list_t;
```

ptr – указатель на первый элемент списка

Структура стека, реализованного массивом:

```
typedef struct array_elem
{
   int i;
   int j;
   int direction;
} array_element_t;
```

i, j – индексы клетки, из которой можно продолжить путь
direction – направление пути

```
typedef struct array
{
   array_element_t *ptr;
   int size;
} stack_array_t;
```

ptr – указатель на начало массива
size – размер массива

Функции

1. `int new_matrix(maze_t *const maze)`
    - **Декларация:** `int new_matrix(maze_t *const maze);`
    - **Описание:** Выделяет память для матрицы лабиринта на основе указанных размеров (`maze->x` и `maze->y`).

2. `void free_memory(maze_t *const maze)`
    - **Декларация:** `void free_memory(maze_t *const maze);`
    - **Описание:** Освобождает выделенную память для матрицы лабиринта.

3. `int init_stacks(stack_array_t *const array, stack_list_t *const list, const int max_stack)`
    - **Декларация:** `int init_stacks(stack_array_t *const array, stack_list_t *const list, const int max_stack);`
    - **Описание:** Инициализирует стек (либо на основе массива, либо на основе списка) с указанным максимальным размером стека.

4. `int peek(stack_array_t *array, stack_list_t *list, int *i, int *j, int *direction)`
    - **Декларация:** `int peek(stack_array_t *array, stack_list_t *list, int *i, int *j, int *direction);`
    - **Описание:** Получает верхний элемент стека без его удаления.

5. `int pop(stack_array_t *array, stack_list_t *list)`
    - **Декларация:** `int pop(stack_array_t *array, stack_list_t *list);`
    - **Описание:** Удаляет верхний элемент из стека.

6. `int push(stack_array_t *array, stack_list_t *list, const int i, const j, const int direction)`
    - **Декларация:** `int push(stack_array_t *array, stack_list_t *list, const int i, const int j, const int direction);`
    - **Описание:** Добавляет новый элемент в вершину стека.

7. `void free_list(stack_list_t *list)`
    - **Декларация:** `void free_list(stack_list_t *list);`
    - **Описание:** Освобождает память, выделенную для стека на основе связанного списка.

Функция Find_check_Function

1. `static int checking_for_corners(maze_t *const maze, int i, int j, stack_array_t *array, stack_list_t *list, int *stack_size, const int direction)`
    - **Декларация:** `static int checking_for_corners(maze_t *const maze, int i, int j, stack_array_t *array, stack_list_t *list, int *stack_size, const int direction);`
    - **Описание:** Проверяет и добавляет соседние ячейки в стек в случае, если это допустимые ходы. Используется для исследования лабиринта.

2. `static void check_space(maze_t *maze, const int i, const int j)`
    - **Декларация:** `static void check_space(maze_t *maze, const int i, const int j);`
    - **Описание:** Проверяет, является ли ячейка в лабиринте пустой и не содержит ли она цифру. В таком случае отмечает ячейку как 'T'.

3. `static int find_start_direction(const maze_t *const maze, const int i, const int j)`
    - **Декларация:** `static int find_start_direction(const maze_t *const maze, const int i, const int j);`
    - **Описание:** Определяет начальное направление движения на основе наличия свободных мест вокруг текущей позиции.

4. `static int direction_move(maze_t *maze, int *i, int *j, const int direction)`
    - **Декларация:** `static int direction_move(maze_t *maze, int *i, int *j, const int direction);`
    - **Описание:** Обновляет текущую позицию в соответствии с указанным направлением и проверяет, не наступили ли тупики или допустимые ходы.

5. `int find_way(int stack_type, stack_array_t *array, stack_list_t *list, maze_t *maze, int *stack_size, const int max_stack)`
    - **Декларация:** `int find_way(int stack_type, stack_array_t *array, stack_list_t *list, maze_t *maze, int *stack_size, const int max_stack);`
    - **Описание:** Основная функция, использующая стек (либо на основе массива, либо на основе списка) для поиска выхода из лабиринта. Итеративно исследует лабиринт до нахождения выхода или опустошения стека.

Алгоритм

Сначала формируется матрица, в которой хранится лабиринт. Если клетка, в которой в данный момент находится курсор, является развилкой, то эта клетка помещается в стек. Если курсор попадает в тупик, то из стека достается верхний элемент, и движение продолжается оттуда, то есть из последней развилки. Если стек пуст, то это признак того, что найти путь к выходу невозможно.

Тесты

Время

Добавление элементов

Количество элементов	Список	Массив
10	12500 тиков	3650 тиков
100	85000 тиков	28000 тиков
1000	600000 тиков	290000 тиков

Удаление элементов

Количество элементов	Список	Массив
10	7500 тиков	3150 тиков
100	37000 тиков	25200 тиков
1000	380000 тиков	250000 тиков

Печать стека

Количество элементов	Список	Массив
10	240000 тиков	210000 тиков
100	1500000 тиков	1490000 тиков
1000	12500000 тиков	13000000 тиков

Занимаемая память

Количество элементов	Список	Массив
10	40 байт	160 байт
100	400 байт	1600 байт
1000	4000 байт	16000 байт

Выводы по проделанной работе

Стек, реализованный с помощью связанного списка, лучше, чем реализация с помощью массивов, зависит от времени.

Таким образом, можно сделать вывод, что если нужно реализовать такую структуру данных как стек, то лучше использовать массив, а не связанный список.

Контрольные вопросы

Что такое стек?

Стек – структура данных, в которой можно обрабатывать только последний добавленный элемент (верхний элемент). На стек действует правило LIFO — последним пришел, первым вышел.

Каким образом и сколько памяти выделяется под хранение стека при различной его реализации?

При хранении стека с помощью списка, то память всегда выделяется в куче. При хранении с помощью массива, память выделяется либо в куче, либо на стеке (в зависимости от того, динамический массив или статический). Для каждого элемента стека, реализованного списком, выделяется на 4 или 8 байт (на большинстве современных ПК) больше, чем для элемента массива. Эти дополнительные байты занимает указатель на следующий элемент списка. Размер указателя (4 или 8 байт) зависит от архитектуры.

В этой программе я использовал указатели с массивами, так что я выделил память и освободил ее, поэтому массивы будут использовать больше памяти для хранения

Каким образом освобождается память при удалении элемента стека при различной реализации стека?

При хранении стека связанным списком, верхний элемент удаляется путь освобождением памяти для него и смещения указателя, указывающего на начало стека. При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека.

Что происходит с элементами стека при его просмотре?

Элементы стека уничтожаются, так как каждый раз достается верхний элемент стека.

Каким образом эффективнее реализовывать стек? От чего это зависит?

Реализовывать стек эффективнее с помощью массива. Он выигрывает как во времени обработки, так и в количестве занимаемой памяти. Вариант хранения списка может выигрывать только в том случае, если стек реализован статическим массивом. В этом случае, память для списка ограничена размером оперативной памяти (так как память выделяется в куче), а память для статического массива ограничена размером стека. Так же, если не известен размер стека, то в таком случае стоит использовать списки.