Отчёт

1. **Дисциплина: Типы и структуры данных**
2. **Лабораторная работа №4**
3. **по теме: «Работа со стеком»**
4. Работу выполнил:
5. студент группы ИУ7-32Б
6. Иванов Всеволод

# Описание условия задачи

Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного списка.

Оценить преимущества и недостатки каждой реализации, получить представление о механизмах выделения и освобождения памяти при работе с динамическими структурами данных.

# Описание технического задания

Программа для поиска пути в лабиринте, заданного матрицей

Область применения – матрицы лабиринтов

Сроки выполнения – 2 недели

Основой для разработки программы послужила потребность в реализации стека различными способами. Для обработки в этой программе был выбран лабиринт в формате матрицы, из которого в стеке хранится маршрут в виде последовательности координат клеток пути.

Для реализации задачи использовались стек свободной памяти и стек пути в виде связанных списков, и стек в виде массива.

Программа должна:

* Содержать указание формата и диапазона данных при вводе
* Содержать указание операций, производимых программой
* Контролировать правильность ввода данных
* Указание операции, производимой программой: добавление элемента в стек, удаление элемента из стека, вычисление (обработка данных)
* Иметь пояснения при выводе результата
* Отображать результаты сравнения времени и памяти работы программы при использовании стека в виде связанного списка и массива
* Вывод на экран списка адресов при удалении стека

# Этап постановки

Исходными данными является лабиринт в виде матрицы (в которой 0 – стена, а 1 - проход). Результатом является стек пути прохождения лабиринта, и лабиринт с отмеченным в нём маршрутом, а также сравнение времени добавления и удаления элемента из различных реализаций стека.

Обращение к программе происходит выбором функций программы путём введения их кодов в консоли. Вывод результатов работы программы также производится в консоль.

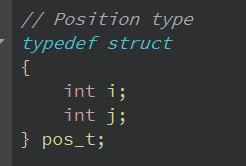
Возможными ошибками являются:

* Ввод матрицы некорректных размеров (менее 3 в одной из сторон)
* Ввод символов в поле размеров
* Ввод несуществующего файла для считывания матрицы
* Ввод символов в поле команды
* Ввод несуществующей команды
* Отсутствие решения лабиринта (начало и конец недостижимы друг для друга или находятся в стене)

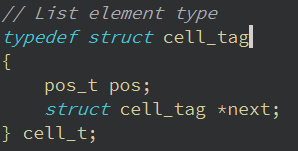
Особых требований для технических средств, аппаратной совместимости, программных средств не имеется.

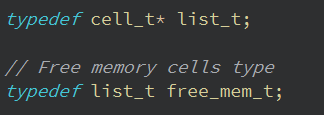
# Описание СД

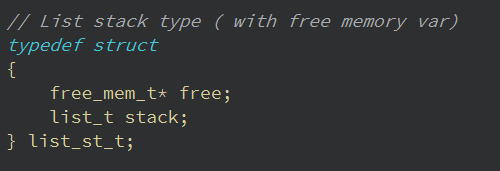
Для хранения позиции в матрице использован тип записи pos\_t, содержащий:

* Целые числа i, j, хранящие номер столбца и ряда ячейки

Для хранения элементов связанного списка была создана структура cell\_t, содержащая:

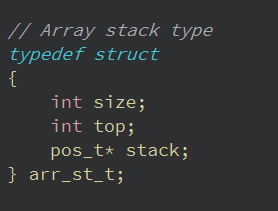
* Структуру pos типа pos\_t, хранящая позицию
* Указатель next на следующий элемент связанного списка

Стек на связанном списке реализован указателем на “головной” элемент стека - list\_t. На данной структуре также простроен стек области свободной памяти free\_mem\_t

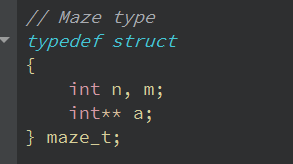
Стек на связанном списке с привязанной областью свободной памяти реализован структурой list\_st\_t, содержащей:

* Указатель free на стек свободной памяти free\_mem\_t
* Стек stack типа list\_t

Для хранения стека на массиве использована структура arr\_st\_t, хранящая:

* Целое число size – размер выделенной памяти под стек
* Целое число top – индекс вершины стека
* Указатель stack – динамический массив элементов pos\_t

Для хранения лабиринта была использована структура maze\_t, содержащая:

* Целые числа n и m, хранящие размер лабиринта
* Указатель на указатель на целые числа a – хранящий элементы матрицы лабиринта

# Описание алгоритма

Генерация лабиринта происходит путём выбора случайной клетки с нечётными координатами, откуда начинается создание проходов шагами по 2 клетки в случайном не пройденном ранее направлении. Все посещённые нечётные клетки заносятся в стек. В случае захода в тупик, ветвление продолжается с последней точки в стеке аналогичным образом. Заполнение продолжается до тех пор, пока стек не опустеет. После завершении генерации в левом и правом краю на нечётных координатах создаются входы в лабиринт.

Прохождение лабиринта начинается с нахождения двух входов в лабиринт. Движение начинается с одного из них. Для прохождения использовался алгоритм “Левой руки”. При движении из клетки выбирается путь, ведущий по левой стене (то есть повороты имеют следующий приоритет: по левую сторону, прямо, по правую сторону) и очередная клетка заносится в стек. В случае, если движение зашло в тупик, из стека снимаются позиции до тех пор, пока не будет встречена позиция, в которой остались не посещённые развилки, после чего движение продолжается. Также в матрице отмечаются посещённые клетки, которые более не посещаются. В момент, когда достигается точка выхода, поиск прекращается и функция возвращает стек пути. В случае, если стек становится пустым, считается, что пути между двумя точками нет.

Оценка времени работы стеков производится путём замера времени многократного занесения и удаления из них элементов. После этого общее время делится на количество повторов операции.

# Набор тестов

Негативные тесты

|  |  |  |
| --- | --- | --- |
| Ввод | Проверяемый случай | Вывод |
| 50 | Использование несуществующей команды | Ошибка: команда не существует |
| afa | Ввод символов | Ошибка: неправильный формат ввода |
| -1 2 | Ввод некорректных размеров лабиринта | Ошибка: некорректный размер лабиринта |
| Йц.1 | Открытие несуществующего или не текстового файла | Ошибка в открытии файла |
| maze.txt  3 3  0 0 0  0 a 0  1 0 0 | Символы в файле лабиринта | Ошибка: неправильный формат файла |
| 0 0 0  1 0 1  1 0 0 | Непроходимый лабиринт | Ошибка: лабиринт непроходим |

Тепличные тесты

|  |  |  |
| --- | --- | --- |
| Ввод | Проверяемый случай | Вывод |
| Команда: 0  5 7 | Генерация лабиринта размерами 5x7 | Вывод сгенерированного лабиринта  Вывод прохождения лабиринта и стеков на связанном списке и на массиве |
| Команда: 1  maze.txt | Считывание лабиринта из файла | Вывод считанного лабиринта  Вывод прохождения лабиринта и стеков на связанном списке и на массиве |
| Команда: 2 | Вызов тестового режима | \*Переход в тестовый режим\* |
| \*тестовый режим\*  Команда: 0 | Выход из тестового режима | Выход  \*завершение работы программы\* |
| \*тестовый режим\*  Команда: 10  1 2 | Добавление элемента в стек на связанном списке | Вывод стека  Вывод области свободной памяти |
| \*тестовый режим\*  Команда: 11 | Удаление элемента из стека на связанном списке | Вывод удалённого элемента  Вывод стека  Вывод области свободной памяти |
| \*тестовый режим\*  Команда: 20  1 2 | Добавление элемента в стек на массиве | Вывод стека |
| \*тестовый режим\*  Команда: 21 | Удаление элемента из стека на массиве | Вывод удалённого элемента  Вывод стека |
| Команда: 30 | Сравнение времени и памяти операций на разных стеках | Время и память, затраченные на добавление и удаление элементов из стека  Сравнение в процентах эффективности стеков |

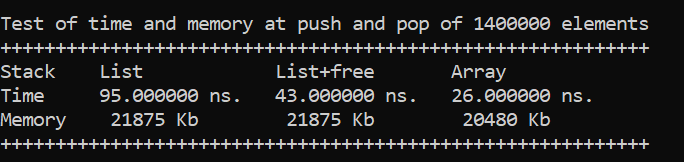
Граничные тесты

|  |  |  |
| --- | --- | --- |
| Ввод | Проверяемый случай | Вывод |
| Команда: 0  3 3 | Генерация лабиринта минимальных размеров | Вывод сгенерированного лабиринта  Вывод прохождения лабиринта и стеков на связанном списке и на массиве |

# Сравнение эффективностей

(Примечание с тесте List+free используется уже заполненная область свободной памяти, таким образом при push выделения памяти не происходит, вместо чего она просто берётся из уже зарезервированной)

Заполнение и удаление стеков на 1 400 000 элементов:



Сравнение реализаций стеков со стандартным стеком на связанном списке:

|  |  |  |
| --- | --- | --- |
| Стек | Время | Память |
| Связанный список | 100% | 100% |
| Связанный список + область свободной памяти (заполненная) | 46% | 100% |
| Массив | 27% | 93% |

Заполнение и удаление стеков на 2 500 000 элементов:



Сравнение реализаций стеков с стандартным стеком на связанном списке:

|  |  |  |
| --- | --- | --- |
| Стек | Время | Память |
| Связанный список | 100% | 100% |
| Связанный список + область свободной памяти (заполненная) | 45% | 100% |
| Массив | 24% | 52% |

# Ответы на вопросы

**1. Что такое стек?**

**Стек - это структура данных, являющаяся последовательным списком, в котором в один момент доступ можно получить только “верхнему” элементу. Стек функционирует по правилу LIFO: .**

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

**При реализации стека в виде массива, для его хранения отводится непрерывная область памяти. Память в размере sizeof(data) \* len байт. Длинна массива обычно отводится с запасом и расширяется при его достижении.**

**При реализации стека в виде связанного списка, для хранения каждого его элемента отводится отдельная область памяти. Элементы в нём связанны при помощи указателей на следующий элемент. Таким образом, под стек отводится (sizeof(data) + sizeof(pointer)) \* len байт. Выделенная память при этом может быть фрагментированной.**

**3. Каким образом освобождается память при удалении элемента стека при различной реализации стека?**

**При удалении элемента из стека на связанном списке память данного элемента может освобождаться сразу, или помещаться в область свободной памяти до востребованности.**

**При удалении элемента из стека на массиве в нём изменяется индекс верхушки стека. Занимаемая память может уменьшаться в случае, когда этот индекс достигает установленной границы относительно размера занимаемой памяти (например при < 1/3)**

**4. Что происходит с элементами стека при его просмотре?**

**Просмотр элемента стека возможен только в случае, если он находится на вершине стека. В таком случае его просмотр возможен путём удаления из стека**

**5. Каким образом эффективнее реализовывать стек? От чего это зависит?**

В случае, когда требуется эффективность по памяти лучше реализация стека массивом, так как время добавления и удаления будут меньше, чем в списке. Также такая реализация может дать выигрыш в занимаемой памяти в некоторых случаях.

В случае, когда требуется использовать больше памяти лучше использовать реализацию на связанном стеке, так как ему не требуется непрерывность памяти, поэтому его размер ограничен только размером свободной оперативной памяти.

# Выводы по проделанной работе

Из проведённого анализа следует вывод, что даже при использовании уже зарезервированной памяти из области свободной памяти стек на связанном списке проигрывает стеку на массиве примерно в 2 раза (в случае, если память под стек не выделена заранее выигрыш будет в 4 раза). Выигрыш по занимаемой памяти у стека на массиве варьируется от 1 до 2 раз, в зависимости от размера стека (т.к. в массиве при достижении границы стека выделяется память в 2 раза более нынешней). Однако, стек на массиве имеет недостаток в том, что выделяемая под него память должна быть непрерывна. Связанный список в свою очередь может быть фрагментированным. Таким образом он может хранить большее количество элементов