МГТУ им. Н. Э. Баумана

Отчет по лабораторной работе №4 по предмету «Типы и структуры данных»
«Работа со стеком»

Выполнила Параскун София,
ИУ7-34Б
Проверила
Никульшина Т. А.

Москва, 2020 г.

1. Описание условия задачи

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек массивом и 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 список или массив свободных элементов стека и создать свой список или массив свободных областей (адресов освобождаемых элементов) с выводом его на экран.

Проверить правильность расстановки скобок трех типов (круглые, квадратные, фигурные) в выражении.

2. Описание ТЗ, включающего внешнюю спецификацию

a) описание исходных данных и результатов

Основная программа получает на вход выбранный пользователем пункт меню (целое число). В соответствии с ним производятся операции взаимодействия со стеком.

Изначально мы имеем пустой стек, который заполняется пользователем по мере использования программы. При добавлении элемента пользователь вводит строку, которая может содержать только скобки или не содержать скобок вообще.

В зависимости от выбранного режима результатом является сообщение о корректности или некорректности расставленных скобок, время добавления/удаления элемента в стек на массиве и на списке, состояние стека с адресами используемой/освобожденной памяти.

b) Способ обращения к программе

Программа «app.exe» запускается через консоль, после чего можно увидеть меню программы, состоящее из 5 пунктов:
1 - Показать состояние стека,
2 - Добавить в стек,
3 - Удалить из стека,
4 - Проверить расстановку скобок (с очищением стека),
0 - Выход из программы.

Все вводы и выводы также осуществляются в консоли.

c) Описание возможных аварийных ситуаций и ошибок пользователя

№ ситуации	Аварийная ситуация	Реакция программы
1	Ввод некорректного пункта меню	Сообщение «Wrong values! Try again.», повторение ввода
2	Добавление элемента в заполненный стек	Сообщение «Stack is full! Clear or delete it.», возврат в главное меню
3	Ввод слишком большого количества символов при добавлении в стек	Сообщение «Too much symbols!», возврат в главное меню
4	Удаление элемента из пустого стека	Сообщение «Stack is empty! Nothing to delete.», возврат в главное меню
5	Показать состояние пустого стека	Сообщение «Stack is empty!», возврат в главное меню

3. Описание внутренних СД

Для начала пропишем используемые в программе константы.

```
#define MAX_LEN 5
#define MAX_STR_LEN 5
```

```
struct stack_all
{
    int stack_len;
    char stack_mass[MAX_LEN];
    struct stack_slot *stack_list;
};
```

Стек описывается структурой stack_all, где stack_len – размерность стека, stack_mass – реализация стека на массиве, *stack_list – указатель на крайний элемент стека.

Структура, которой описывается элемент стека называется stack_slot, где value – значение элемента стека, *next- указатель на следующий элемент стека.

```
struct stack_slot
{
    char value;
    struct stack_slot *next;
};
```

```
struct mem_slot
{
    struct stack_slot *stack_slot;
    int busy;
    struct mem_slot *next;
};
```

Также используется структура mem_slot для описания занятых/освобожденных сегментов памяти.

Память для массива выделяется статически, для структур stack_slot и mem_slot – динамически.

4. Описание алгоритма

Добавление в стек (если стек не заполнен): при добавлении в массив новый элемент записывается в конец массива, количество элементов увеличивается на 1. При добавлении в список создается соответствующая структура, в значение которой кладется элемент, а указатель на следующий приравнивается текущему указателю на список. После чего указателю на список присваивается адрес созданного элемента.

Удаление из стека (если стек не пустой): при удалении из массива, значению удаляемого элемента присваивается „\0“, количество элементов уменьшается на 1. При удалении из списка указателю на список присваивается значение указателя на следующий элемент удаляемой записи, после чего освобождается память по указателю удаляемого элемента.

Обработка данных: для этого создается вспомогательный стек, в который записываются встреченные в основном стеке закрывающиеся скобки. Если же встречается открывающаяся скобка, то проверяется последний элемент вспомогательного стека. При совпадании вида скобок, элемент извлекается из вспомогательного стека и проверка продолжается, в ином случае результату проверки присваивается значение неверного.

5. Описание используемых функций

В используемых ниже функциях stack – общая структура стека, mem – список, хранящий адреса используемых/освобожденных элементов.

*void show_stack(struct stack_all stack, struct mem_slot *mem)* – показывает состояние стека: заполненность, хранящиеся элементы, используемая/освобожденная память.

*void add_to_stack(struct stack-all *stack, struct mem_sot **mem)* – добавляет введенные пользователем элементы в стек.

*void delete_from_stack(struct stack_all *stack, struct mem_slot **mem)* – удаляет из стека последний элемент.

*void check_brace(struct stack_all *stack, struct mem_slot **mem)* — производит проверку расставленных скобок, освобождая весь стек.

6. Результаты эффективности

Рассмотрим результаты работы при разных размерах стека. Время указано в тактах.

Добавление и удаление не зависит по времени от размера стека, так как затрагивается только «верхний» элемент.

Размер: N	Добавление	Удаление
Массив	34	28
Список	9724	9722

Сравним обработку данных и память для предельно заполненного стека:

Размер: 5	Массив	Список
Обработка данных	858	20342
Память	5	80
Размер: 30		
Обработка данных	1272	90100
Память	30	480
Размер: 70		
Обработка данных	1702	133378
Память	70	1120
Размер: 100		
Обработка данных	2474	244316
Память	100	1600
Размер: 200		
Обработка данных	4024	281044
Память	200	3200

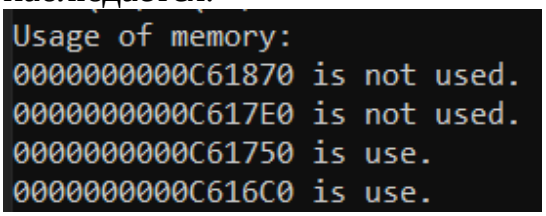
Как мы видим, выигрыша по времени при реализации списком не получилось, а по памяти мы получили огромные потери.

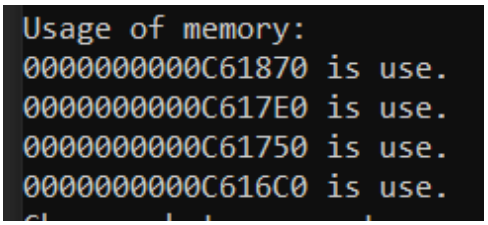
Но что же будет при разной заполненности стека? По времени выигрыш массива очевиден, поэтому рассмотрим используемую память.

Заполненность	5%	8%	15%	30%
Размер	При реализации массивом/стеком			
100	100/80	100/128	100/240	100/560
1000	1000/800	1000/1280	1000/2400	1000/5600
10000	10000/8000	10000/12800	10000/24000	10000/56000
100000	100000/80000	100000/12800	100000/24000	100000/560000

Как мы видим, реализация стека списком дает выигрыш по памяти при 5-6% заполненности, что немаловажно, когда под стек необходимо выделить очень много памяти статически (!). При динамическом выделении памяти под массив это преимущество сойдет на нет, но тогда будет тратится много времени на перезаписывание массива в новый блок памяти.

Также заметим, что фрагментации при повторном выделении памяти не наблюдается.





7. Выводы по проделанной работе

В ходе этой лабораторной удалось поработать с таким типом данных как стек. Также удалось реализовать операции взаимодействия с ним: добавление-удаление элементов, обработка данных стека.

Стек был реализован двумя способами — массивом и односвязным списком. После проведения оценки эффективности, стало понятно, что массив намного экономнее по памяти (на один элемент стека в массиве уходит 1 б, в списке 16 б), при добавлении и удалении эффективнее в среднем в 300 раз по времени, НО нельзя забывать, что это применимо к статическому массиву, память под который фиксирована. В случае динамического массива значительная часть времени уходила бы на выделение нового участка памяти и переписывание старых данных в него, это могло привести к относительному равенству по времени добавления/удаления элементов.
Что касается времени обработки данных, здесь не играет роль, каким образом выделена память под массив. Он сильно эффективнее по времени обработки данных (в зависимости от размера стека от 25 до 70 раз), так как помимо работы со стеком-списком мы работаем еще с дополнительным стеком, который хранит закрывающиеся скобки, он тоже описан списком.

Ответы на контрольные вопросы

1. Что такое стек?

Стек — структура данных, в которой можно обрабатывать только «верхний» элемент. Действия осуществляются по принципу «Last In, First Out» - последним пришел, первым вышел.

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

При реализации стека массивом память выделяется статически или динамически (на усмотрение программиста). При реализации списком память выделяется динамически. Для отдельного элемента стека при реализации списком выделяется больше памяти, чем при реализации массивом, так как необходимо помимо значения элемента хранить указатель на следующий элемент.

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

Массивом: считываются данные об удаляемом элементе, после чего указатель конца сдвигается на одну позицию влево.

Списком: считываются данные об удаляемом элементе, после чего указатель на последний элемент сдвигается к предыдущему элементу, и память из-под удаляемого элемента освобождается.

4. Что происходит с элементами стека при его просмотре?

При просмотре стека осуществляется считывание информации о верхнем элементе, его удаление и переход к следующему, пока не будет достигнута нижняя граница.

5. Каким образом эффективнее реализовать стек? От чего это зависит?

Эффективнее реализовывать стек массивом, так как это оказывается оперативнее по времени и выигрышнее по памяти.