Работа 3. Изучение особенностей размещения объектов в динамической памяти

Теоретический материал

Для выполнения работы вам будет необходимо использовать материал по следующим темам:

1. Массивы и указатели (материал из предыдущей работы)

- Общая информация о работе с массивами: https://metanit.com/c/tutorial/2.13.php
- Общий материал по указателям: https://metanit.com/c/tutorial/5.1.php
- Связь указателей и массивов: https://metanit.com/c/tutorial/5.5.php

2. Динамическое выделение памяти

- Функции для работы с динамической памятью(функции `malloc`, `realloc, free`): https://metanit.com/c/tutorial/5.8.php

3. Работа с объектами в памяти

- Работа с памятью (функция `memcpy`): https://metanit.com/c/tutorial/8.3.php

Задание

В этом задании вам предстоит изучить работу с функциями `memcpy`, `malloc`, `realloc`, `free`, также провести замеры времени выполнения операций изменения размера выделенной памяти.

В данной работе конкретные значения, содержащиеся в элементах массивов, не имеют значения, но должны быть проинициализированы любым удобным способом.

Задание (базовой уровень)

1. **Создание массивов**
   Создайте два массива `A` и `B`, каждый объемом от 100 МБ до 1 ГБ, используя

функцию `malloc`. Сохраните начальные значения указателей на массивы `A` и `B`. Объем вводит пользователь

2. **Увеличение размера массивов**
    - Увеличьте размер массива `B` в 2 раза, используя функцию `realloc`.
    - Измерьте время, затраченное на выполнение этой операции, и выведите его на экран.
    - Сохраните и выведите значения указателя на массив `B` до и после увеличения размера.
3. Повторите описанные шаги из п. 2 для массива `A`, также увеличив его размер в 2 раза. Замерьте время выполнения функции `realloc` в этом случае и также выведите результат на экран. Сохраните и выведите значения указателя на массив `A` до и после увеличения.
4. **Копирование фрагмента массива****
    - Введите числа `S` (приращение числа элементов) и `T` (позиция для вставки).
    - Увеличьте размер массива `B` на `S` элементов.
    - Переместите содержимое части массива `B`, начинающееся с позиции `T`, в конец массива.
    - Скопируйте `S` элементов из начала массива `A` в массив `B`, начиная с элемента `B[T]`.
5. **Уменьшение размера массивов**
    - Уменьшите размер второго массива `B` в 2 раза, снова используя функцию `realloc`.
    - Измерьте время выполнения и выведите его на экран.
    - Сохраните и выведите значения указателя на массив `B` до и после уменьшения размера.
    - Затем уменьшите размер массива `A` в 2 раза. Замерьте время выполнения функции `realloc` и выведите результат на экран. Выведите значения указателя на массив `A` до и после уменьшения.
6. **Сравнение указателей и анализ времени выполнения**
   Сравните указатели до и после каждого вызова функции `realloc`. Сделайте выводы о том, изменились ли адреса массивов после изменения их размера и насколько отличалось время выполнения функции `realloc` при увеличении и уменьшении размеров.
7. **Освобождение памяти**
   Выведите на экран произвольный элемент массива A, выполните освобождение памяти, выделенной под массивы `A` и `B`, попытайтесь снова “прочесть” и вывести на экран значение элемента массива после того, как его память была освобождена. Объясните результат.

Результаты отчета

1. Приведите полученные значения времени выполнения функции `realloc` для увеличения и уменьшения размеров массивов.
2. Представьте таблицу или список со значениями указателей до и после каждого вызова `realloc`.
3. Сделайте вывод о том, как изменения размера массива влияют на указатель (изменился ли адрес после вызова `realloc`) и как время выполнения функции зависит от операции увеличения или уменьшения размера массива.

Подсказки:

Для замера времени выполнения используйте функции из стандартной библиотеки времени, например:

```
#include <time.h>

clock_t start, end;
double elapsed_time;

// Засекаем время до вызова функции
start = clock();

/* Измеряемая операция */

// Засекаем время после завершения функции
end = clock();

// Вычисляем разницу времени в миллисекундах
elapsed_time = (double)(end - start) / CLOCKS_PER_SEC * 1000;
```

В случае, если исследуемый блок кода выполняется слишком быстро, и разница между end и start равна или близка к нулю, попробуйте модифицировать алгоритм так, чтобы искомый фрагмент кода выполнялся много раз подряд с одними и теми же параметрами, и измерьте суммарное время. При этом обратите внимание, чтобы в настройках компиляции не использовалась оптимизация (обычно для этого надо указать ключ -O0 в командной строке gcc, для других компиляторов возможны другие варианты, уточняйте в руководстве по среде разработки).

Задание (продвинутый уровень)

Общая постановка задачи

Создайте программу на языке C, создающую последовательно несколько динамических массивов с разными начальными размерами, и увеличивающую каждый из них в несколько этапов вплоть до определенного объёма. Выполните анализ

производительности динамического увеличения массивов в зависимости от их начального размера и *политики увеличения объема* (то есть числа последовательных приращений его размера и величины приращения).

Задачи:

1. Создайте массивы с заданными начальными размерами.
2. Реализуйте следующие политики увеличения объема массива:
    - Увеличение на фиксированный объем (например, 1 КБ или 1 МБ за раз).
    - Увеличение в 2 раза от предыдущего размера.
3. Заполняйте массивы данными различного объема – например, по одному символу или сразу большими блоками данных (для заполнения блоками используйте вспомогательный массив размером, равным размеру блока, и многократно копируйте его в целевой массив с помощью memcpy()) и измеряйте время, затраченное суммарно на каждую *политику увеличения объема* и размер блока (т.е., суммарное время от создания массива до окончания его заполнения для каждой комбинации начальный размер+политика увеличения+размер блока).
4. Сохраните результаты замеров времени в виде таблиц, а также представьте данные в виде графиков для наглядного анализа производительности при различных политиках увеличения объема и начальных размерах массивов.

Результат:
Подготовьте отчет с таблицами и графиками, демонстрирующими зависимости времени заполнения от начального размера массива и политики увеличения. Сделайте выводы о том, какие стратегии увеличения и начальные размеры оказывают наименьшее влияние на производительность.

Параметры и условия для каждого варианта

1. **Вариант 1**
    - Начальные размеры массивов: 1 МБ, 100 МБ
    - Полиси увеличения: 1 МБ за раз, удвоение размера
    - Объемы заполнения: 1 символ, 10 МБ, 500 МБ
2. **Вариант 2**
    - Начальные размеры массивов: 10 КБ, 500 МБ
    - Полиси увеличения: 512 байт за раз, 1.5x от текущего размера
    - Объемы заполнения: 1 символ, 1 МБ, 250 МБ
3. **Вариант 3**
    - Начальные размеры массивов: 5 МБ, 50 МБ
    - Полиси увеличения: 256 байт за раз, 5 МБ за раз
    - Объемы заполнения: 5 МБ, 100 МБ, 1 ГБ
4. **Вариант 4**
    - Начальные размеры массивов: 10 МБ, 300 МБ

    - Полиси увеличения: 500 КБ за раз, 2.5x от текущего размера
    - Объемы заполнения: 100 символов, 10 МБ, 300 МБ
5. **Вариант 5**
    - Начальные размеры массивов: 100 КБ, 400 МБ
    - Полиси увеличения: 2 КБ за раз, удвоение размера
    - Объемы заполнения: 1 символ, 50 МБ, 200 МБ
6. **Вариант 6**
    - Начальные размеры массивов: 100 КБ, 10 МБ
    - Полиси увеличения: 1 КБ за раз, 100 КБ за раз
    - Объемы заполнения: 1 символ, 100 КБ, 5 МБ
7. **Вариант 7**
    - Начальные размеры массивов: 500 КБ,, 50 МБ
    - Полиси увеличения: 128 байт за раз, 3x от текущего размера
    - Объемы заполнения: 10 символов, 1 МБ, 10 МБ
8. **Вариант 8**
    - Начальные размеры массивов: 25 МБ, 250 МБ
    - Полиси увеличения: 512 байт за раз, 4 МБ за раз
    - Объемы заполнения: 50 символов, 5 МБ, 50 МБ
9. **Вариант 9**
    - Начальные размеры массивов: 2 МБ, 100 МБ
    - Полиси увеличения: 2 КБ за раз, 20 МБ за раз
    - Объемы заполнения: 1 символ, 2 МБ, 100 МБ
10. **Вариант 10**
    - Начальные размеры массивов: 1 КБ, 10 МБ
    - Полиси увеличения: 128 байт за раз, удвоение размера
    - Объемы заполнения: 1 символ, 500 КБ, 2 МБ