Работа с памятью

Memory model, stack, heap,
static array, dynamic array

Задачи занятия

- Поработать с разными видами памяти в C++.
- Научиться использовать статический массив.
- Научиться использовать динамический массив.
- Порешать задачи на константные указатели и увидеть “заразность” const.
- Попробовать ловить утечки памяти

Статический массив

Когда можно и нужно его использовать?

- Количество элементов известно на этапе компиляции или хотя бы верхняя граница;
- Размер массива меньше размера стека;
- Хватает области видимости переменной;

Инициализация

```
int arr[3];                 // {3 x garbage}
int arr[3] = {}             // {0, 0, 0}
int arr[3] = {1, 2}         // {1, 2, 0}
int arr[3] = {1, 2, 3}      // {1, 2, 3}
int arr[] = {1, 2}          // {1, 2}
```

Использование

- Доступ к элементу arr[i];
- Можно передавать в функции с потерей длин измерений
- Нельзя возвращать из функции (почему?)

Динамическая память

- T* new T / delete T*
- T* new T[size_t] / delete[] T*
- void* malloc(size_t) / void free(void *)

Динамическая память всегда должна быть освобождена после использования

Инициализация массивов

- int* arr = new int[length](); //default ctor
- std::fill(arr, arr+length, INITIAL_VALUE)
- std::fill_n(arr, length, INITIAL_VALUE)

Использование

- Указатель на динамическую память может быть возвращен из функции.
- При возвращении указателя на динамическую память, нужно четко определить кто его будет освобождать
- Каждому new должно соответствовать delete, иначе получим утечку памяти.

const

- Ключевое слово, помечающее данные неизменяемыми (immutable)
- Неизменяемость - сильное свойство, часто упрощающее понимание и написание программ
- const ‘заразен’. const type нельзя передать в функцию, принимающую type. (Но если очень хочется и по-другому никак - то можно.)
- Смодифицируйте const char* = “foo”;

Утечка памяти (memory leak)

- Что это такое?
  - Выделили память
  - Забыли освободить или потеряли указатель
  - Память выделена, но не используется. Она может быть недоступна другим процессам до завершения программы

Valgrind

- Используем valgrind для проверок утечек памяти
- sudo apt-get install valgrind
- valgrind --tool=memcheck --leak-check=yes my_program

Типичные ошибки

- Несимметричное использование оператора new и delete, new[] и delete[] приводит к undefined behavior.
- Утечки памяти
- Использование указателя после delete
- Чтение или запись за границами выделенной памяти
- Использование неинициализированной памяти

Что может произойти?

```
int* a = new int[2];
int* b = new int[2];

a[0] = 5;
a[1] = 7;
delete[] a;
std::cout << a[0] << std::endl;
std::cout << a[1] << std::endl;

b[0] = 5;
b[1] = 7;
delete b; // no [] after delete
std::cout << b[0] << std::endl;
std::cout << b[1] << std::endl;
```

Ответ

-2144604096

1

5

7

Результат может отличаться на разных системах, но показан почти худший вариант.

Что может произойти?

```
…
delete[] a;
delete b;
…
a[0] = 5;
a[1] = 7;
std::cout << a[0] << std::endl;
std::cout << a[1] << std::endl;

b[0] = 5;
b[1] = 7;
std::cout << b[0] << std::endl;
std::cout << b[1] << std::endl;
```

Ответ

5

7

5

7

Результат может отличаться на разных системах, но показан наихудший вариант.

Задача 1.

Показать, что адреса стека убывают, а адреса кучи растут.

Задача 2.

Используя статический массив вычислить для числа N(0 <= N < 21). Сколько существует способов забраться на лестницу из N ступеней, если можно шагать на каждую по очереди или через одну.

Задача 3.

Реализовать конкатенацию неизменяемых массивов (см. memcpy):

```
const char* concat(
    const char a[],
    const size_t a_size,
    const char b[],
    const size_t b_size,
    size_t &concat_size
);
```

Задача 4.

Создать матрицу интов MxN делая не более 2 аллокаций динамической памяти с красивой индексацией в виде:

Matrix[i][j]

Не использовать перегрузку операторов.

Нужно использовать операторы new, delete и арифметику указателей.

Задача 5.

Проверить на наличие утечек памяти задачу 3. Исправить их, если они обнаружатся. Если утечек нет - создайте их и поизучайте вывод программы valgrind.

Задача 6.

Если вы дружите с ООП в C++ можете реализовать простенький scoped_ptr. Что это такое и как оно работает:

http://stackoverflow.com/questions/569775/smart-pointers-boost-explained

http://www.boost.org/doc/libs/1_56_0/libs/smart_ptr/scoped_ptr.htm

http://google.com

Задача 7.

Если вы гуру C++, то можете реализовать простенький shared_ptr. Что это такое и как оно работает вы, наверное, и так знаете, но на всякий случай:

http://google.com