Creative Software Programming Practice (week-4-2)

Every assignment will be announced on **Thursday** and should be sumitted by next **Tuesday**.

We have practice classes on Wednesdays and Thursdays.
The contents of the practice class are different from the assignments and aim to be completed on the same day.

Assignments will be publish on Thursday and must be submitted by the next Tuesday.

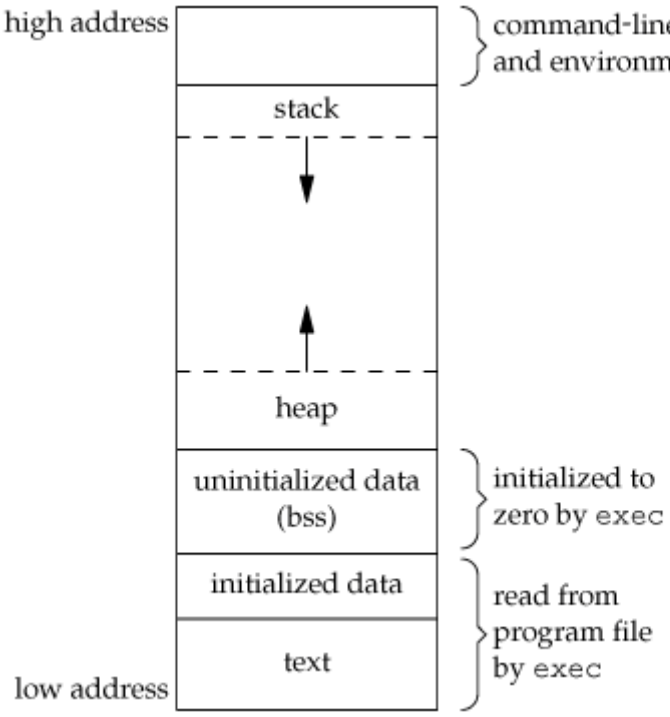
In this week **Handed out will be Sep 24, 2020, Due Sep 29, 2020**

Topics

1. Dynamic memory allocation
2. Reference

1. Dynamic memory allocation

When you run a C/C++ program, OS allocates memory space for the program like below picture:



- Stack: local variables, function parameters, return address
- Heap: for dynamic memory allocation
- bss, data: global variables, static variables
- text: compiled binary code

In traditional C, you can allocate new memory with `malloc` and release with `free` like below.

```
T* var = (T*)malloc(sizeof(T) * n);
free(var);
```

In C++ support `new` / `delete` for memory allocation.

```
    T* var = new T[n];
    delete[] T;
```

`malloc` and `new` allocates N ($N \times sizeof(T)$ for `new`) bytes of memory block and return the pointer of the block. `free` and `delete` deallocates the allocated memory block.

```
#include <iostream>

void function(int* array) {
    array[0] = 34;
    std::cout << "The address of array: " << array << std::endl;
    std::cout << "The value of array[0]: " << *array << std::endl;
}

int main() {
    int size = 5;
    int* array = new int[size];

    std::cout << "The address of array: " << array << std::endl;
    std::cout << "The value of array[0]: " << *array << std::endl;

    function(array);

    std::cout << "The address of array: " << array << std::endl;
    std::cout << "The value of array[0]: " << *array << std::endl;

    delete[] array;

    return 0;
}
```

2. Reference

The pointer is not a reference, and just pass the integer value(memory address) and handle it a special value. But in C++, there is a real reference and you can pass it to function or anywhere.

```
int a = 3;
int& b = a;
b = 2;
assert(a==3);
```

```
#include <iostream>

void func_ref(int& a) {
    a += 3;
}
void func_val(int a) {
    a += 3;
}
void func_ptr(int* a) {
    *a += 3;
}

int main() {
```

```
    int a;

    std::cout << "Function val" << std::endl;
    a = 3;
    std::cout << "The value of a: " << a << std::endl;
    func_val(a);
    std::cout << "The value of a: " << a << std::endl;

    std::cout << "Function ref" << std::endl;
    a = 3;
    std::cout << "The value of a: " << a << std::endl;
    func_ref(a);
    std::cout << "The value of a: " << a << std::endl;

    std::cout << "Function ptr" << std::endl;
    a = 3;
    std::cout << "The value of a: " << a << std::endl;
    func_ptr(&a);
    std::cout << "The value of a: " << a << std::endl;

    return 0;
}
```

3. Pratice

Implement memory reallocation function. (*c already realloc function but implement your own*)

It takes an N memory pointer and returns it to an M (where $N < M$) memory pointer. But the data it contains must be guaranteed. Follow below skeleton codes.

```
#include <iostream>
#include <cassert>

template <typename T>
T* realloc(T* pointer, size_t org_size, size_t new_size) {
    // TODO
    // return new allocated memory
}

int main() {
    int* array = new int[32];

    for (int i = 0; i < 32; i++) {
        array[i] = i;
    }

    array = realloc(array, 32, 64);

    // data guarantee
    for (int i = 0; i < 32; i++) {
        assert(array[i] == i);
    }
    // also you can assign re-allocated area
    for (int i = 0; i < 32; i++) {
        array[32+i] = i;
    }
```

```
    return 0;
}
```

```
```