COMP 345 Fall 18 Week 4

Haotao Lai (Eric)
h_lai@encs.concordia.ca

Lab Instructor

Section: B-X   9999   --W----   20:30 22:20 H929

Name: Haotao Lai (Eric)

Office: EV 8.241

Email: h_lai@encs.concordia

Website: http://laihaotao.me/ta

Content

- pointer
- struct, class, inheritance
- constructor, destructor
- const, static

Pointer

A pointer contains the memory address of a portion of memory that in turn contains a specific value.

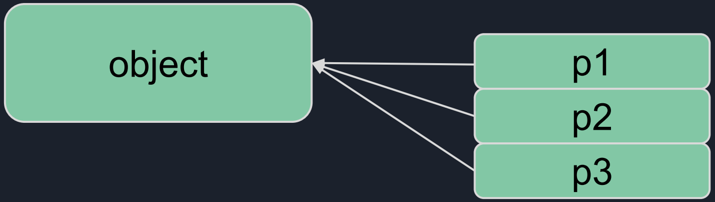
Dereferencing operator: *, e.g. *p refers to the object pointed to by the pointer.
Address operator: &, e.g. &i refers to the address of the first memory cell holding an object.

```
int main() {
   int len = 3;
   int *iPtr = new int[len];
   for (int i = 0; i < len; i++) {
        *iPtr[i] = i;
   }
   // do something with iPtr
}
```

Two common problems when play with pointer

Dangling Pointer

Several pointer-variables point the the same object, at some point, the object get deleted via one of the pointer, but the rest don’t know, keep manipulating the object.



Example code:

```
delete p3;
// do something with p1 or p2
```

Memory Leak

You allocate a piece of memory, but for some reasons, you never release it.

Example code:

```
int foo( ) {
    int *i_ptr = new int(1);
    If ( *i_ptr == 1 ) {
        // memory leak occurs, forget to delete
        return 2;
    }
    delete i_ptr;
    Return 0;
}
```

Possible Solution

Two possible solution to the previous two problems:

- A possible solution will be smart pointer
- Another solution will be “write code carefully” by figure out the ownership of the pointer

For those who interested in smart pointers can checkout the following tutorial slide set:

http://laihaotao.me/ta/comp345/smart_pointer.pdf

Pointer

How the memory exactly works for the following code

```
void foo() {
    int *p1, *p2;
    p1 = new int;
    *p1 = 42;
    p2 = p1;
}
```

Stack Memory and Heap Memory

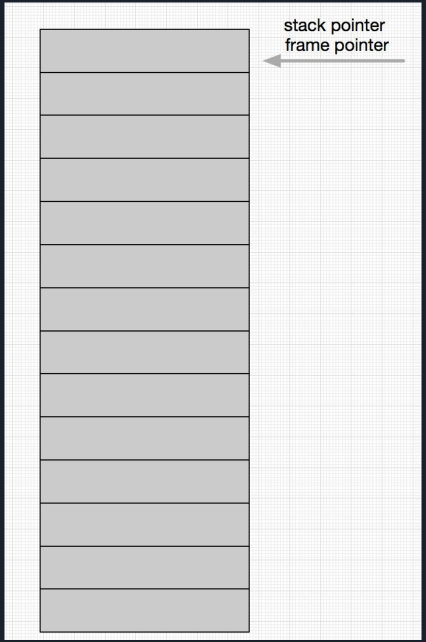
What we talk about here is NOT data structure stack or heap!

Stack Memory: store local variables for code block, create when a new code block is entered (eg. function)

Heap Memory: store the dynamic variables, when you use the keyword “new”, variables will be allocated on the heap, the programmer have to manage the memory on the heap.

Stack Memory

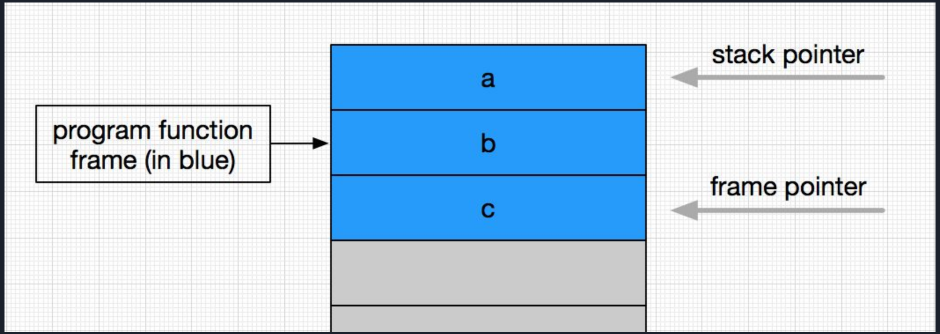
```
int add(int a, int b) {
    return a + b;
}

int program() {
    int a, b, c;
    a = 10;
    b = 20;
    c = add(a, b);

    return 0;
}
```



ps: assume in this example, “program” function is the “main” function.

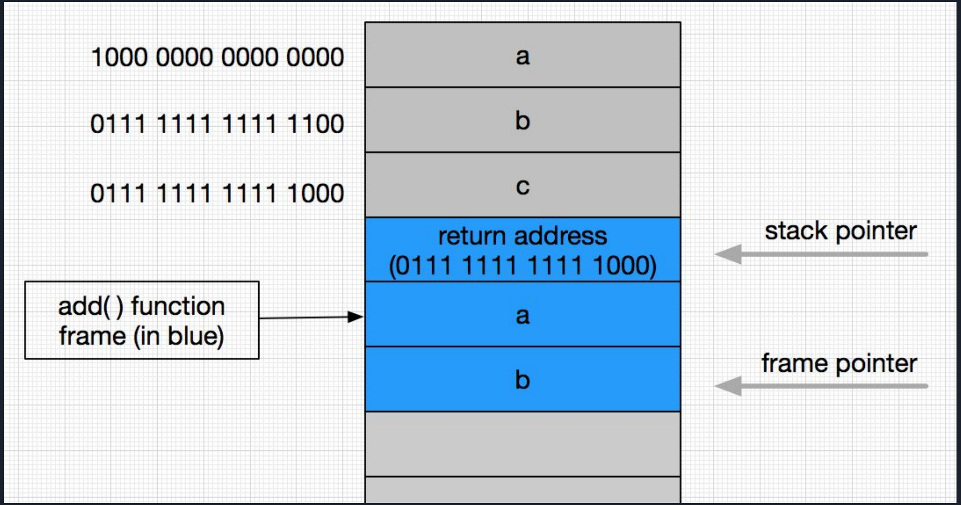
Stack Memory

```
int add(int a, int b) {
    return a + b;
}

int program() {
    int a, b, c;
    a = 10;
    b = 20;
    c = add(a, b);

    return 0;
}
```



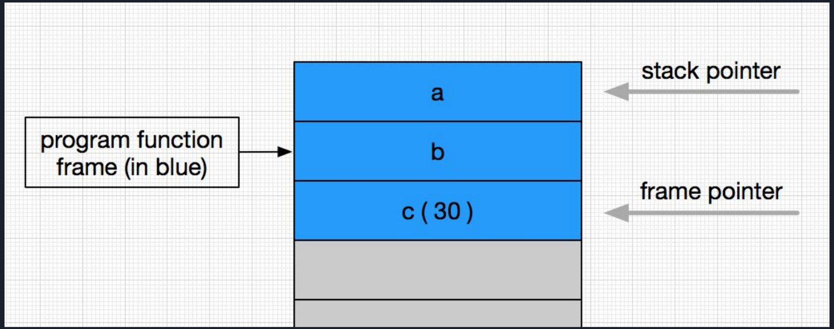
Stack Memory

```
int add(int a, int b) {
    return a + b;
}

int program() {
    int a, b, c;
    a = 10;
    b = 20;
    c = add(a, b);

    return 0;
}
```



Stack Memory

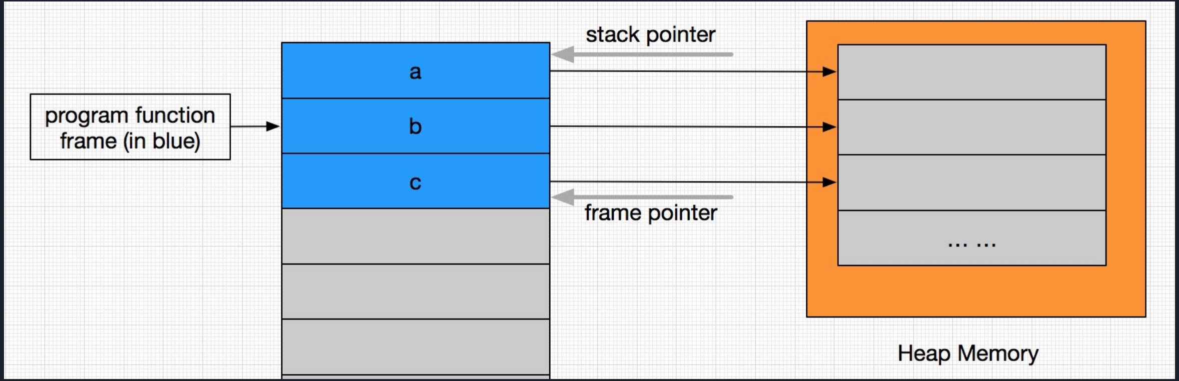
```
int add(int a, int b) {
    return a + b;
}

int program() {
    int a, b, c;
    a = 10;
    b = 20;
    c = add(a, b);

    return 0;
}
```



The add function frame is no longer existing any more!

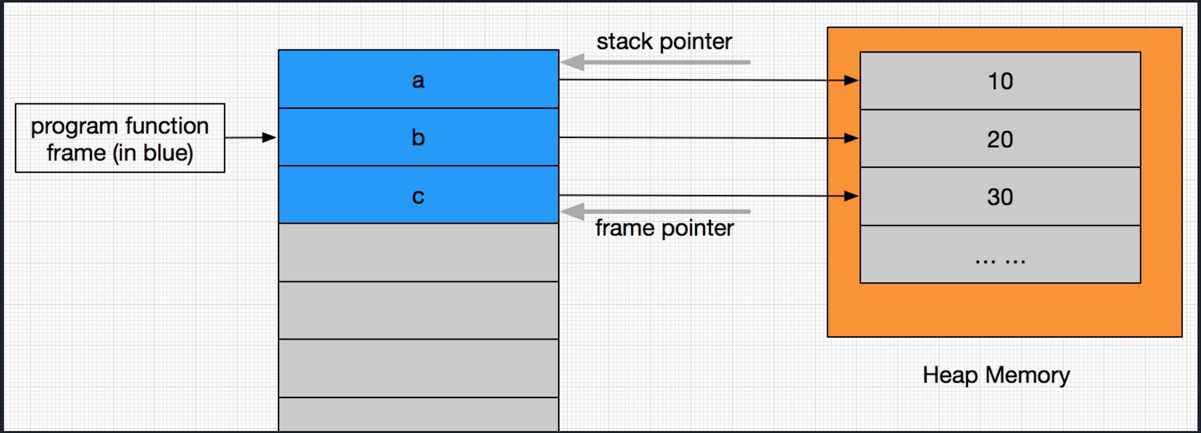
Heap Memory

```
int add(int a, int b) {
    return a + b;
}

int program() {
    int *a = new int;
    int *b = new int;
    int *c = new int;
    *a = 10;
    *b = 20;
    *c = add(*a, *b);
    delete a, b, c;
    return 0;
}
```



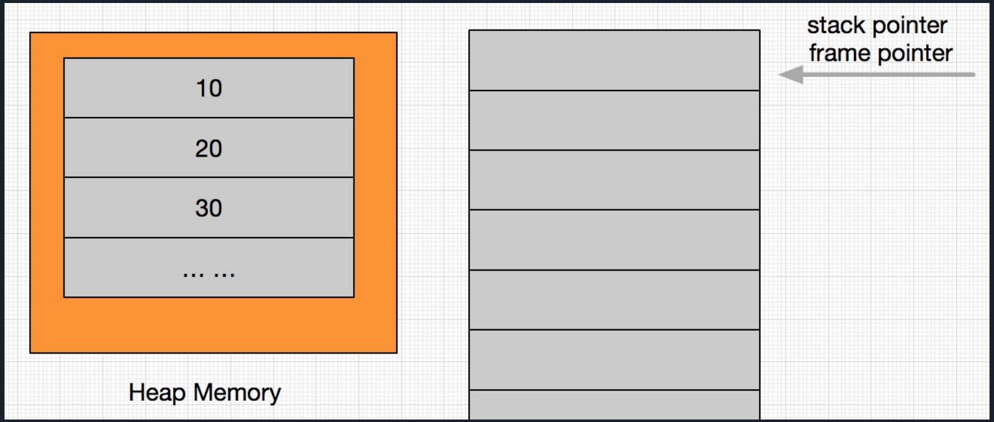
Heap Memory

```
int add(int a, int b) {
    return a + b;
}

int program() {
    int *a = new int;
    int *b = new int;
    int *c = new int;
    *a = 10;
    *b = 20;
    *c = add(*a, *b);
    delete a, b, c;
    return 0;
}
```



Heap Memory

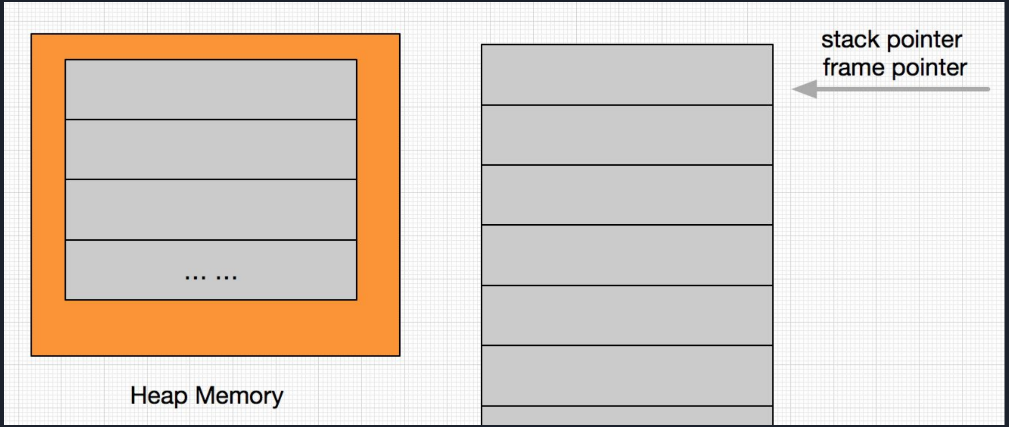
```
int add(int a, int b) {
    return a + b;
}

int program() {
    int *a = new int;
    int *b = new int;
    int *c = new int;
    *a = 10;
    *b = 20;
    *c = add(*a, *b);
    // delete a, b, c;
    return 0;
}
```



Memory leak! When the program function goes out of scope only the pointer variables which are local variables to the program function got deleted, the memory where these pointers refer to will still be there since the programmer didn't release them explicitly.

Heap Memory

```
int add(int a, int b) {
    return a + b;
}

int program() {
    int *a = new int;
    int *b = new int;
    int *c = new int;
    *a = 10;
    *b = 20;
    *c = add(*a, *b);
    delete a, b, c;
    return 0;
}
```



Memory released properly

Question?