CS 225 Spring 2019 :: TA Lecture Notes
1/23 Heap Memory

By Wenjie

- **Heap Memory**
  - Starts from the low memory values, grows up (opposite of Stack)
  - When the Heap meets the Stack, we are "out of memory"
  - Only create heap memory using keyword `new`:
    - allocates heap memory
    - calls the object's constructor
    - returns a pointer to the memory
  - Only free heap memory using keyword `delete`:
    - calls the object's destructor
    - marks memory as free
  - delete the objects we created when we no longer use them
    - Heap memory is never automatically reclaimed
    - If we don't free memory on heap, we are leaking memory. We cannot access it and we cannot reclaim it.
    - It's a good practice to set deleted variable to NULL or nullptr. It's a special value that means memory location "0", and c++ throws an error if one tries to access it.

heap1.cpp

```
1  int main() {
2       int *p = new int;
3  //pointer on stack, int on heap
4       Cube *c = new Cube(10);
5  //pointer on stack, object on heap
6       delete c;   c = nullptr;
7  //delete and set null
       delete p;   p = nullptr;
       return 0;
   }
```

CS 225 Spring 2019 :: TA Lecture Notes

1/23 Heap Memory

By Wenjie

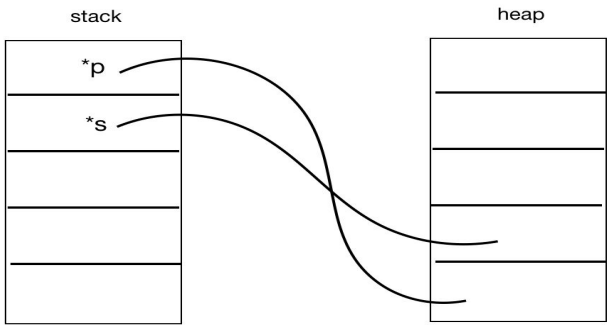

Fig. A stack pointer that points to heap memory blocks

copy.cpp

```
#include <iostream>
using std::cout;
using std::endl;

int main() {
  int  i =  2,  j =  4,  k =  8;
  int *p = &i, *q = &j, *r = &k;

  k = i;
  cout << i << j << k << *p << *q << *r << endl;
  //          2     4      2       2        4       2

  p = q;
  cout << i << j << k << *p << *q << *r << endl;
  //          2     4      2       4        4       2

  *q = *r;
  cout << i << j << k << *p << *q << *r << endl;
  //          2     2      2       2        2       2
}
```

CS 225 Spring 2019 :: TA Lecture Notes
1/23 Heap Memory

By Wenjie

- **Reference Variable**
  - Reference variable is an alias to an existing variable. It never creates new memory, and it needs to be initialized when declared, and it can never be redeclared.
  - When we modify the reference variable, that would also modify the variable being aliased.

reference.cpp

```
1  #include <iostream>
2
3  int main() {
4    int i = 7;
5    int & j = i;   // j is an alias of i
6
7    j = 4;
8    std::cout << i << " " << j << std::endl;
9   //                   4               4
10  // i and j are the same thing, they change together
11
12   i = 2;
13   std::cout << i << " " << j << std::endl;
14  //                   2               2
15   return 0;
16 }
```

- **The use of "&" operator**
  - A declaration of a reference variable would be like:

```
1  Int a = 3
2  int & b = a;         //declaring a reference variable
```

  - However, this process should not be confused with the case of getting the memory address of a variable which would also involved with "&" operator, as the example shown below:

```
1  Cube c;
2  std::cout << "Mem address storing c: " << &c << std::endl;
```

CS 225 Spring 2019 :: TA Lecture Notes
1/23 Heap Memory

By Wenjie

- Similarly, the use of "*" operator would also be either declaration of a pointer variable or the dereferencing of a variable's to get its value. As we see more through cs225, we will have better sense of knowing which context of situations that we are at.