Object Oriented Programming

Additional Handout

Call stacks, heaps and pointers

Andrew Rice
Nov 2019
Each cell is a ‘byte’

32-bit architecture
=> 4 bytes to a word

Address
(usually written in hexadecimal)
e.g. 0x07C

Each row is a ‘word’

Call stack

Heap
```c
void f(int x) {
    char c = 'a';
    long l = 1234;
    int i = 10;
}

f(4);
```

This example is in C/C++
void f(int x) {
    char c = 'a';
    long l = 1234;
    int i = 10;
}

f(4);
```c
void f(int x) {
    char c = 'a';
    long l = 1234;
    int i = 10;
}

f(4);
```
`void f(int x) {
    char c = 'a';
    long l = 1234;
    int i = 10;
}

f(4);`

1234 is bigger than one byte

1234 & 0xFF = 210
1234 >> 8 = 4
void f(int x) {
    char c = 'a';
    long l = 1234;
    int i = 10;
}

f(4);
1  void f(int x) {
2      char c = 'a';
3      long l = 1234;
4      int i = 10;
5  }
6
7  f(4);
void f() {
    int i = 1;
    int j = 2;
    int k = 3;
    int* p = &i;
    int* q = &k;
}

* on a LHS means 'its a pointer'

& on a RHS means 'take the address of'
```c
void f() {
    int i = 1;
    int j = 2;
    int k = 3;
    int* p = &i;
    int* q = &k;
    int* r = p + 1;
}
```

We can do arithmetic on pointers (based on the datatype size)
```c
void f() {
    int i = 1;
    int j = 2;
    int k = 3;
    int* p = &i;
    int* q = &k;
    int* r = p + 1;
    int l = *r;
}
```

* on the RHS means ‘dereference’ i.e. follow the pointer.
void f() {
    int i = 1;
    int j = 2;
    int k = 3;
    int* p = &i;
    int* q = &k;
    int* r = p + 1;
    int l = *r;
    int m = *(q + 1);
}

Nothing will stop you making mistakes!
In C++ you can choose whether you want to allocate on the stack or the heap.
Items on the stack exist only for the duration of your function call

Items on the heap exist until they are deleted
Now go back to the main notes!
In Java primitive types go on the stack

Everything else goes on the heap
This example is in Java

```java
test() {
    int i = 3;
    int[] a = new int[] {1, 2};
    String s = "a";
}
```

Java delete’s for us automatically. This is called Garbage Collection.
static void test() {
    int i = 3;
    int[] a = new int[] {1, 2};
    String s = "a";
}

‘a’ and ‘s’ are references. These are like pointers but you can’t do arithmetic on them.

When you say s.toUpperCase() you are ‘dereferencing’ s and calling the method toUpperCase on it.
References in C++ are a completely different concept!
static void test() {
    int i = 3;
    int* k = &i;
    int& j = i;
}

This example is in C++
static void test() {
    int i = 3;
    int* k = &i;
    int& j = i;
}
1 static void test() {
2     int i = 3;
3     int* k = &i;
4     int& j = i;
5 }

& on the LHS means ‘reference’
Recap for Java

- Primitive types on the stack
- Everything else on the heap
- References are values on the stack that ‘point’ to somewhere on the heap
- References are like pointers but you can’t do arithmetic on them
- Java references are not much like C++ references