CS162: Introduction to Computer Science II

Pointers

### Overview of Memory

- A computer’s memory is divided into numbered memory locations
- When you declare a variable, eg.
  ```
  int x
  ```
  You “name” a memory location and store the value of the variable in that location
- Behind the scenes, the compiler will put x at address 1000

### Pointers

There are 2 operators with pointers:

- **The dereference operator (*)**:
  - Eg. \(*p\) returns the contents at the memory location pointed to by the pointer \(p\)
- **The addressof operator (&)**
  - Eg. \(\&x\) returns the memory address of variable \(x\)
Pointers

```c
int x = 1;
int *p1 = &x;
```

```
1 . . .
p1
```

Note: `cout << x` would print out 2

Pointers

Another example:

```c
int x = 100;
int *p1 = &x;
int *p2 = p1;
*p2 = 200;
```

```
100 . . .
p1
```

```
p2
```

All the couts will print 200
Pointers

Tip: If you declare a pointer, don’t leave it uninitialized eg.

\[ \text{int } *p; \]

Instead, explicitly initialize it to something meaningful eg.

\[ \text{int } *p = \text{<something> } ; \]

If you can’t think of something meaningful, use NULL

\[ \text{int } *p = \text{NULL}; \]

Dynamic Memory Allocation

- The `new` operator returns a pointer to a dynamically allocated variable

  \[ \text{int } *p = \text{new int}; \]
  \[ *p = 10; \]

- or equivalently

  \[ \text{int } *p = \text{new int(10)}; \]

- The `new` operator reserves space on the heap for an object of the specified type (in the example above, it is an int)

- The heap is a special area of memory specifically for dynamically allocated variables

Dynamic Memory Allocation

- Dynamically allocating memory for an int is rather silly.

- Typically you dynamically allocate memory for an object eg.

  \[ \text{Wordlist } *w = \text{new Wordlist("words.txt")} \]
  \[ \text{Character } *c = \text{new Character("Bob", 100, 10, 10);} \]

- The new operator will invoke the constructor of the object

Dynamic Memory Allocation

What happens if you forget to delete memory that you dynamically allocated with new?

- You get a memory leak

- The chunk of memory you dynamically allocated is never returned to the heap

- If this happens enough times, your program may run out of memory

Dynamic Memory Allocation

Beware of dangling pointers!

\[ \text{int } *p = \text{new int(10);} \]
\[ \text{delete } p; \]
\[ /* p is now a dangling pointer */ \]
\[ \text{std::cout } \ll *p \ll \text{std::endl;} \]

Causes a crash!
Dynamic Memory Allocation

Tip: Set dangling pointers to NULL
```cpp
int *p = new int(10);
delete p;
p = NULL;
if( p != NULL )
    std::cout << *p << std::endl;
```

Dynamic Memory Allocation

- Dynamic memory allocation is great for arrays when you don't know the size ahead of time
- Eg. suppose size is a parameter passed in to the function
  ```cpp
double* d = new double[size];
```
- The pointer d points to the first element of the array

Dynamic Memory Allocation

To delete the array, use
```cpp
delete [] d
```

WARNING: DO NOT USE
```cpp
delete d
If d is an array
```