Memory Management and Pointers

CSC 340

February 17, 2016

1

Pointer Basics

- Type of variable that stores a memory address (of a single object or an array of objects)
- Pointers "point" to a location in memory

```
int age = 37;
int * pointer = &age;
```

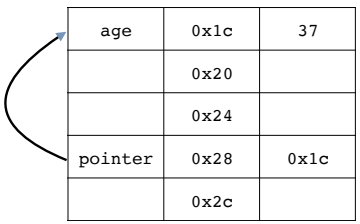
- Pointers must always point to a legal memory address before you attempt to use them
  - If they don't, you'll get segmentation (or other memory) faults

2

2

Visualizing Pointers

```
int age = 37;
int * pointer = &age;
```



age	0x1c	37
	0x20	
	0x24	
pointer	0x28	0x1c
	0x2c	

3

3

Declaring and Initializing Pointers

4

- Pointer variables must be declared to have a pointer type
- The * operator identifies the variable as a pointer variable

```
// NULL pointer
double *d_pointer = NULL;
Person *p_pointer = NULL;
// Multiple declarations (what are the types?)
int *p1 = NULL, *p2 = NULL, v1, v2;
```

4

The & and * operators

5

- & - The address of operator
  - Returns the address of a variable, which can be assigned to a pointer variable

```
Person john, *pointer = NULL;
pointer = &john;
```

- * - dereferencing operator (or the value of operator)

```
cout << *pointer;
(*pointer).first_name = "John";
```

5

The -> operator

6

- Used to access a data or function member in an object, through a pointer
- Is a short hand version of dereferencing * and the member-of . combination

```
// Equivalent:
(*person).first_name = "John";
person->first_name = "John";
```

6

Pointer Assignment

- When two pointers are of the same data type, you can assign one pointer to the other:

```
Person john, *one_person, *other_person;
one_person = &john;
other_person = one_person;
```

- Both pointers now point to the same object
- What does this do?

```
*other_person = *one_person;
```

- https://gist.github.com/jrob8577/567eba3ffcfaec58c080

The new and delete operators

- Pointers can be used to point to memory space dynamically allocated by the new operator

```
Person * person_array = new Person[10];
```

  - The new operator may fail (we will learn about exception handling later)
  - Once created successfully, person_array can be treated as a fixed size array

- When dynamically allocated variables are no longer needed, delete them to return memory to the freestore

```
delete person_array;
```

- Undefined pointer variables are also called stale pointers. Dereferencing a stale pointer is usually disastrous.

Pointer arithmetic

- You can add and subtract with pointers
  - The ++ and — operators can be used
  - Two pointers of the same type can be subtracted to the obtain the number of indexed variables between them (the pointers should be in the same array!)

10

Pointer arithmetic example

```
int * array = new int[10];

// Need to init
for( int i = 0; i < 10; i ++ )
{
    *(array + i) = i;
}

for( int i = 0; i < 10; i++ )
{
    // Same as array[i]
    cout << *(array + i) << " ";
}
```


11

Pointers and Functions

- Pointers can be passed to functions
  - Similar to call by reference
- Pointers can be returned from functions
  - Create an array in the function, and return it
  - The caller is responsible for deleting the array
  - Should not return the address of a local object

12

Reference Variables

- A reference variable is a pointer constant that always dereferences implicitly
  ```
  int longname = 10;
  int & count = longname;
  ```
- count can be treated as an alias of longname
- Must be initialized at declaration
- Often not necessary

Other uses for pointers

- Linked data structures
  - Linked Lists
  - Tree
  - Graphs
- Inheritance and Polymorphism