



#### Pointers in C++



- A pointer is a variable that stores the memory address of another entity (e.g. variable)
- Use the \* operator to declare a pointer

```
int *ptr1, *ptr2; //not initialized yet!
```

■ Use the & operator to get the address of a variable

```
ptr1 = &x; // x was previously declared
```

■ Use the \* operator to go from the pointer to the data being pointed to

```
*ptr1 = 10; // x = 10 now
```

#### Pointers in C++



```
int a = 5;
                       What are the values of the following?
int *ptr = &a;
a. ptr
```

- b. \*ptr
- c. ptr == a
- d. ptr == &a
- e. &ptr
- f. \*a
- g. \*&a
- h. \*\*&ptr

#### + Dynamically allocated memory



■ The new keyword allocates memory from the heap

```
string *strPtr = new string("hello");
int *intPtr = new int(5);
```

- Tip: Don't use new when a stack-allocated variable can be used instead!
- Tip: If you do use new make sure you use delete
- Tip: If you have multiple pointers to same piece of memory
  - Beware of stale pointers (point to already freed memory)
  - Beware of double deleting (deleting same piece of memory twice)

### + Call-by-value



- Java and C++ use call-by-value when passing parameters
- Call-by-value means the value of the input arguments are copied into the formal parameters

```
int main() {
  int x = 5;
  int y = 7;

swap(x,y);

input arguments
```

```
void swap(int m, int n){
  int temp = m;
  m = n;
  n = temp;
}
```

x and y aren't swapped!

+ Call-by-reference



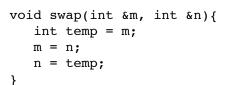
■ C++ allows call-by-reference

```
int main() {
   int x = 5;
   int y = 7;
   swap(5,7);
}
```

```
void swap(int &m, int &n){
  int temp = m;
  m = n;
  n = temp;
}
```

x and y are now swapped!

## + Call-by-reference



- A reference variable is different from a pointer variable
- In this context, the & operator makes m and n reference variables
- i.e., m and n refer to actual variables and not copies!

#### \*Binary Search Example



```
int binarySearch(int val, vector<int> arr, int lo, int hi)
{
    if( lo > hi) { return -1; }
    int mid = (lo+hi)/2;
    if(arr[mid] == val) {
        return mid;
    }
    else if(val < arr[mid]) {
        return binarySearch(val, arr, lo, mid-1);
    }
    else {
        return binarySearch(val, arr, mid, hi);
    }
}</pre>
```

+ Binary Search Example



new function prototype

int binarySearch(int val, const vector<int>& arr,int lo, int hi);

- The & operator means no copying of input arguments
- const means this function will not change (mutate) this input parameter
  - Only const methods can be called on arr

#### + More Pointers in C++



- The dereference operator \* has low precendence
- Pointers to objects (e.g. vector<int>\* vecPtr)
  - Use parenthesis to enforce order: (\*vecPtr).push\_back(5)
  - Alternatively, use the -> operator: vecPtr->push\_back(5)

#### Classes in C++



- The Big Three: destructor, copy constructor, operator=
- These special functions are already written for you!
- Rule-of-thumb: If you need to overwrite one of these, overwrite them all!
- Destructor
  - Called when object goes out of scope or is deleted
  - Frees resources (e.g. memory, closes files)

# + Copy Constructor



- Constructs a new object from an existing object
- The copy constructor is called when,

```
IntCell copy = original;
IntCell copy(original);
an input parameter to a call-by-value function
an object returned by value
```

It would not be called in this instance:

```
IntCell copy;
copy = original;
```

#### operator=



- Assignment for two already constructed objects
- Example usage,

```
IntCell first(3);
IntCell scnd;
scnd = first;
```