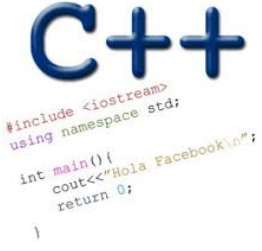
C++ PROGRAM MEMORY MODEL, POINTERS AND REFERENCES

Problem Solving with Computers-I

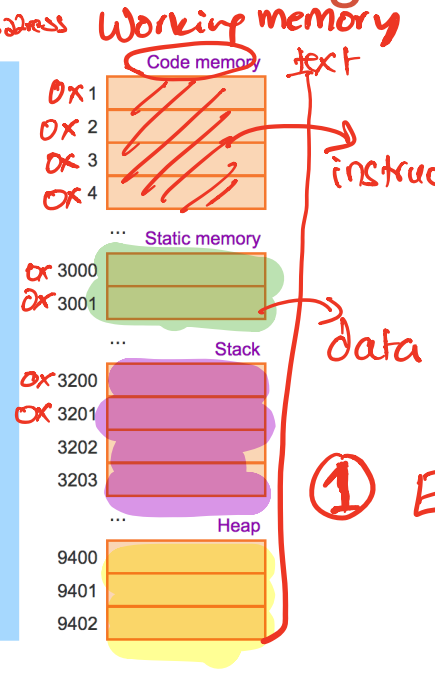


Learning Goals

- Review basics of classes
  - Defining classes and declaring objects (last lecture)
  - Access specifiers: private, public (last lecture)
  - Different ways of initializing objects and when to use each:
    - Default constructor
    - Parametrized constructor
    - Parameterized constructor with default values
    - Initializer lists
- Develop a mental model of how programs are represented in memory.
- Understand pointer and reference mechanics and how they are used to pass parameters to functions

C++ Memory Model a.k.a Program's Memory Regions

```cpp
#include <iostream>
using namespace std;

// Program is stored in code memory

int myGlobal = 33;     // In static memory

void MyFct() {
   int myLocal;        // On stack
   myLocal = 999;
   cout << " " << myLocal;
}

int main() {
   int myInt;              // On stack
   int* myPtr = nullptr; // On stack
   myInt = 555;

   myPtr = new int;        // In heap
   *myPtr = 222;
   cout << *myPtr << " " << myInt;
   delete myPtr; // Deallocated from heap

   MyFct(); // Stack grows, then shrinks

   return 0;
}
```



Java, Python: removing data happens automatically

C++: some of the data is removed automatically but not always

① Everything has a memory address

The code regions store program instructions. myGlobal is a global variable and is stored in the static memory region. Code and static regions last for the entire program execution.

Pointers

- Pointer: A variable that contains the address of another variable
- Declaration: *type* * pointer_name;

```
int* p;
```

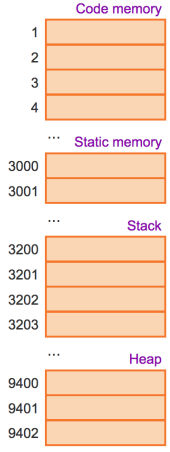
// junk

```
complex* c;
```

// junk

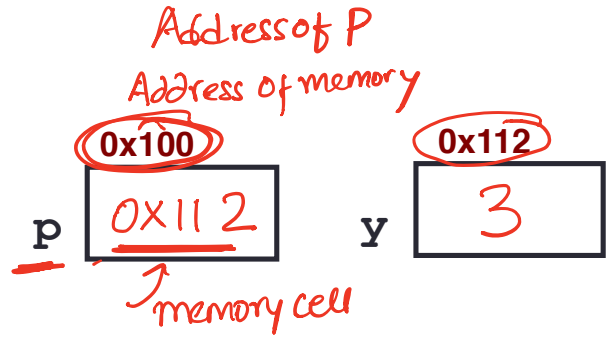
```
int x;
```

// junk

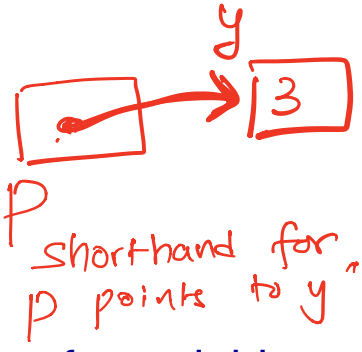


How to make a pointer point to something

```
int* p;
int y = 3;
p = &y; //p points to y
```

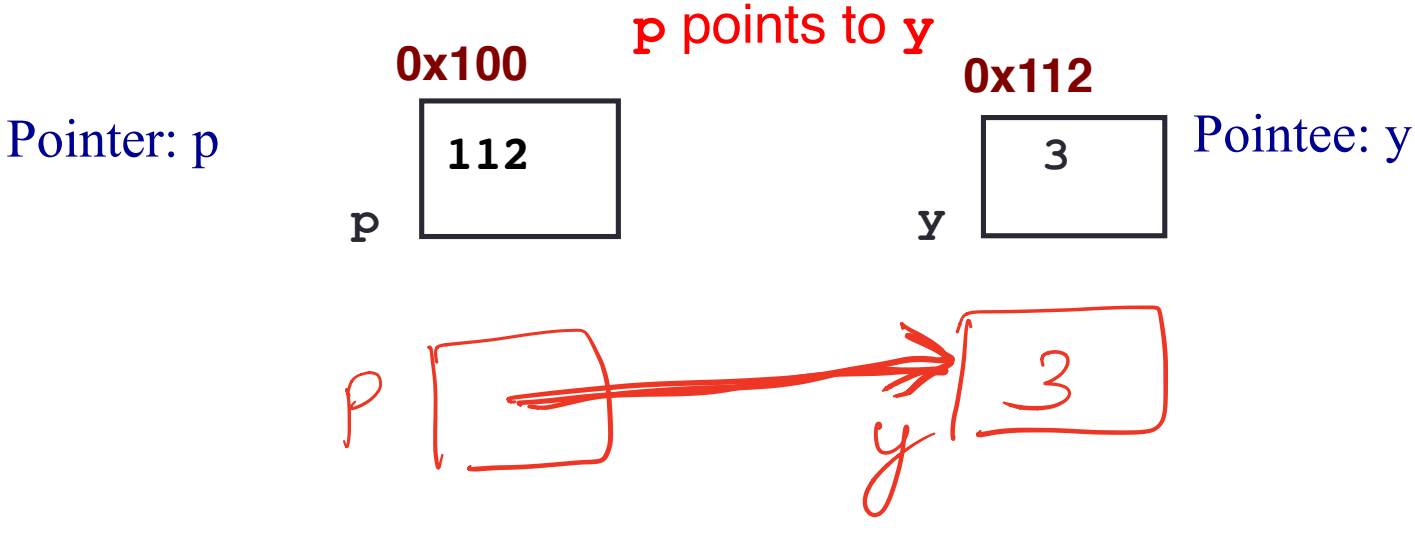



```
cout << &p;  // prints 0x100
cout << &y;  // 0x112
cout << p ;  // 0x112
```



To access the location of a variable, use the address operator '&'

Pointer Diagrams:
Diagrams that show the relationship between pointers and pointees

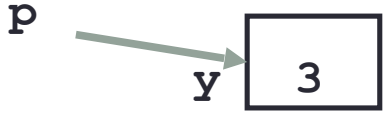


You can change the value of a variable using a pointer !

```
int* p, y;
y = 3;
p = &y;
cout << *p; // Dereferencing p to get the value of y
*p = 5;
```

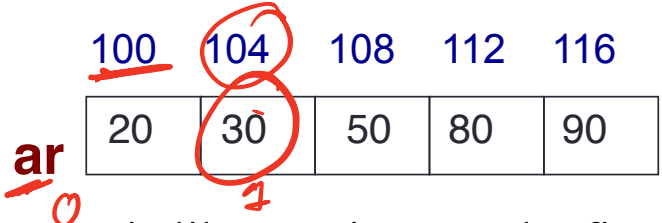
Two ways of changing the value of a variable

- Change the value of y directly:



- Change the value of y indirectly (via pointer p):

Arrays and pointers



- `ar` is like a pointer to the first element
- `ar[0]` is the same as `*ar`
- `ar[2]` is the same as `*(ar+2)`

*(ar+1)
100 + 1*4
= 104
int *ar

- Use pointers to pass arrays in functions
- Use *pointer arithmetic* to access arrays more conveniently

Next time

- Dynamic Memory Management in C++