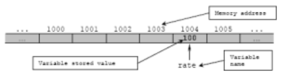
CS 142
C++ Pointers

Rhodes College

Memory Management

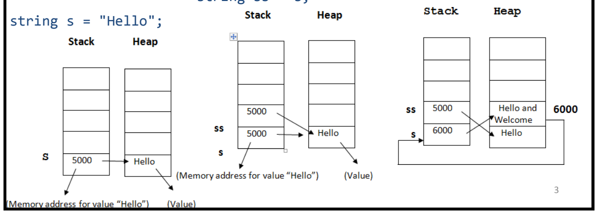


- Imagine all your RAM in your computer as one big continuous array, indexed from 0 to # of bytes-1.
- Each program on your computer is given a chunk of memory to work with when it runs by the OS.
- This area is called "the stack," and every compiled programming language manages this area of memory in mostly the same way.
  - When a program is compiled, the compiler has to determine exactly how many bytes of memory all the variables within EACH FUNCTION will take up.

Memory Management

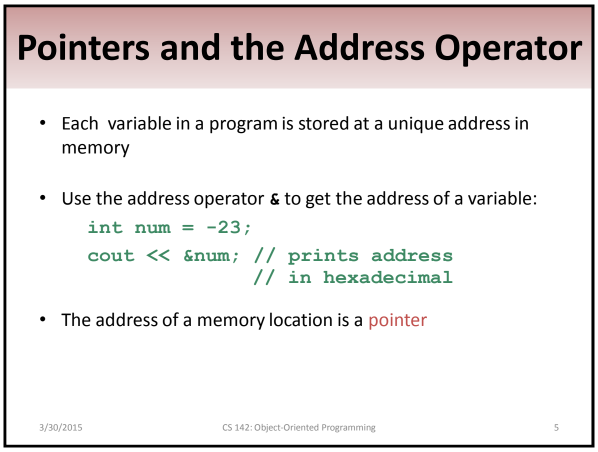
- It is impossible to store variables that grow and shrink on the stack, or variables whose size cannot be determined at compile time. (size of activation records are determined at compile time and always must stay the same).
  - Ex: Two strings of indeterminate length.

```
string s = "Hello";
string ss = s;
s = s + " " + "and Welcome";
```



Memory Management

- Variables that grow and shrink or whose size we don't know until run time are stored in a different area of memory called the HEAP or FREE STORE.
  - Automatic variables on the stack are managed for us.
  - Variables on the heap we manage ourselves – we control where we want to put them, when we want them to exist, and when we want them to go away.
- Since this is a completely different area of memory, we need a completely different programming idea to put a variable here.
- We need something called a POINTER.

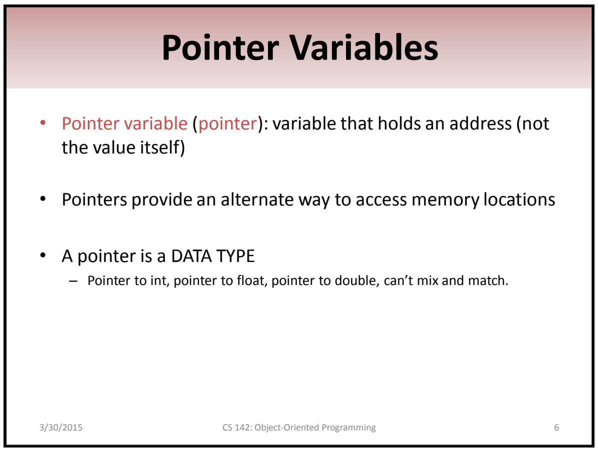


Pointers and the Address Operator

- Each variable in a program is stored at a unique address in memory
- Use the address operator **&** to get the address of a variable:

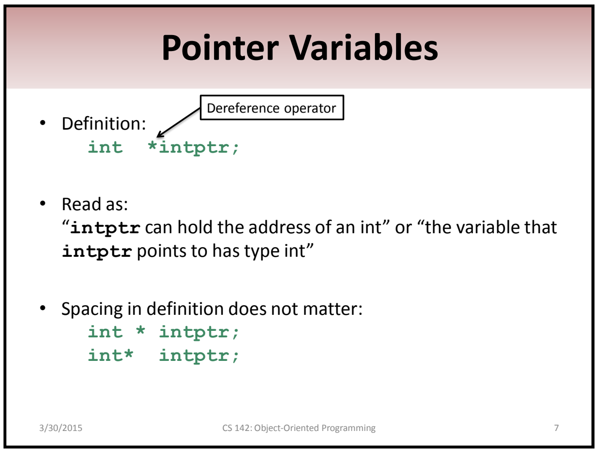
```
int num = -23;
cout << &num; // prints address
              // in hexadecimal
```

- The address of a memory location is a pointer



Pointer Variables

- Pointer variable (pointer): variable that holds an address (not the value itself)
- Pointers provide an alternate way to access memory locations
- A pointer is a DATA TYPE
  - Pointer to int, pointer to float, pointer to double, can't mix and match.



Pointer Variables

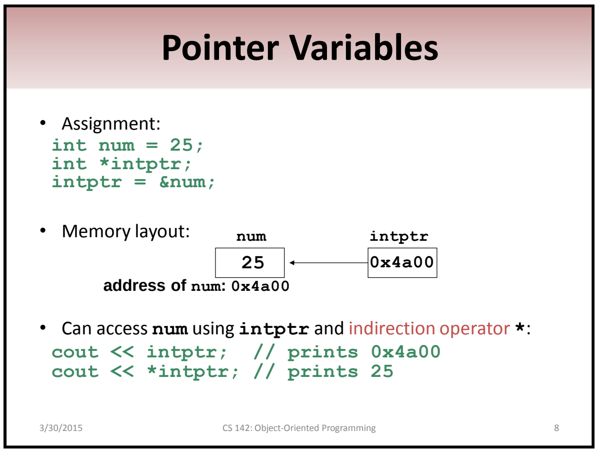
- Definition:

Dereference operator

```
int  *intptr;
```

- Read as:
  "**intptr** can hold the address of an int" or "the variable that **intptr** points to has type int"
- Spacing in definition does not matter:

```
int * intptr;
int*  intptr;
```



Pointer Variables

- Assignment:

```
int num = 25;
int *intptr;
intptr = &num;
```

- Memory layout:

$$\begin{tikzcd} \overset{\texttt{num}}{\boxed{25}} & \overset{\texttt{intptr}}{\boxed{\texttt{0x4a00}}} \arrow[l] \end{tikzcd}$$

address of **num**: **0x4a00**

- Can access **num** using **intptr** and indirection operator *****:

```
cout << intptr;  // prints 0x4a00
cout << *intptr; // prints 25
```

Initializing Pointers

- Can initialize to NULL or 0 (zero)
```
int *ptr = NULL;
```
- Can initialize to addresses of other variables
```
int num, *numPtr = &num;
int val[ISIZE], *valptr = val;
```
- Initial value must have correct type
```
float cost;
int *ptr = &cost; // won't work
```

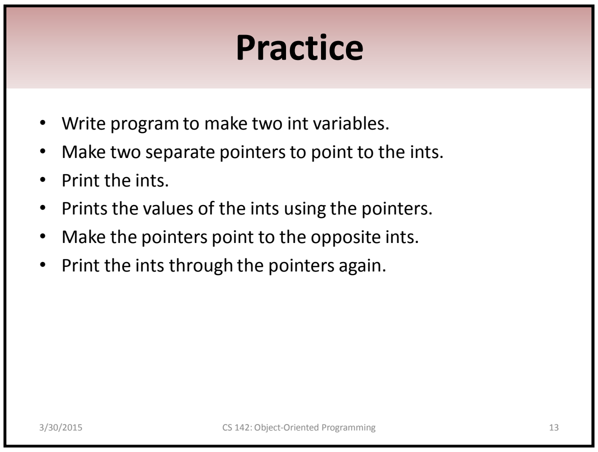
Comparing Pointers

- Relational operators can be used to compare addresses in pointers
- Comparing addresses in pointers is not the same as comparing contents pointed at by pointers:
```
if (ptr1 == ptr2)    // compares
                     // addresses
if (*ptr1 == *ptr2) // compares
                     // contents
```

Demo

- simple_ptr_ex.cpp in Public directory under C++ -> pointers

MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.
FakePlus.com



Practice

- Write program to make two int variables.
- Make two separate pointers to point to the ints.
- Print the ints.
- Prints the values of the ints using the pointers.
- Make the pointers point to the opposite ints.
- Print the ints through the pointers again.