

CSC111

Admin

- Exams -> handed back in labs this week
 - Please check the addition on your exam
 - Mark appeals for the midterm exam must be done in person at office hours
- Reference sheets – please come to office hours to pickup (deadline Feb 21st)
- Office hours back to normal (Tuesday + Friday)

Pointers

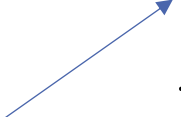
all pointer types take up
8 bytes of memory



Primitive types so far		Pointer types	
Declaration	Can hold	Declaration	Can hold
<code>char c;</code>	A character	<code>char* cptr;</code>	An address of a variable of type char
<code>int i;</code>	An integer value	<code>int* iptr;</code>	An address of a variable of type int
<code>double d;</code>	A floating point value	<code>double* dptr;</code>	An address of a variable of type double

New operator: **&** (address of operator)

```
int x;  
x = 4;  
&x;  
int* iptr;  
iptr = &x;
```

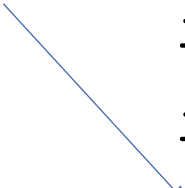
*address of
operator* 

New operator: * (dereference operator)

- Confusing because the * has multiple uses
 - multiplication: `int x = 3 * 2;`
 - declaring a pointer variable: `int* iptr;`
 - value pointed to by: `*iptr;`

```
int x;  
x = 4;  
int* iptr;  
iptr = &x;  
*iptr;
```

*dereference
operator*



Can be read as:
“value pointed to by ...”

the value pointed to by `iptr` is `x`

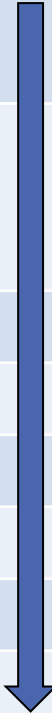
```
int z = *iptr;  
*iptr = 20;
```

`z` gets the value 4

`x` gets the value 20

Adding * (dereference) and & (address of) to precedence table

Precedence	Description	Associativity
Highest	Operations enclosed in brackets (), ++/-- postfix	left to right
	+/- unary operator, ++/-- prefix, (type) cast, !, & (address of), * (dereference)	right to left
	*, /, %	left to right
	+, -	left to right
	<, <=, >, >=	left to right
	==, !=	left to right
	&&	left to right
		left to right
Lowest	=, +=, -=, *=, /=, %=	right to left



```
#include <stdio.h>
```

```
void add1( int* num_ptr);
```

← Function PROTOTYPE

```
int main ( ) {  
    int x = 12;
```

```
    int* ptr = &x;
```

```
    add1(ptr);
```

```
    add1(&x);
```

} Function CALLs
must pass **expected argument**

```
    return 0;
```

```
}
```

```
/* Purpose: adds 1 to number pointed to by num_ptr
```

```
* Parameters: int* num_ptr – address of an int
```

```
*/
```

} Documentation

```
void add1 ( int* num_ptr) {
```

```
    *num_ptr = *num_ptr + 1;
```

} Function DEFINITION

```
}
```


Double pointers

- Pointers, that point to another pointer are valid
 - Can theoretically have many levels of indirection

```
#include <stdio.h>
```

```
void add1( int* num_ptr);
```

```
int main ( ) {
```

```
    int a = 7;
```

```
    int* b = &a;
```

```
    int** c = &b;
```

```
    printf("%d\n", **c);
```

```
    return 0;
```

```
}
```

Why Pointers

- Allow for the 'return' of multiple values from a function
 - Demo 3
- Needed for some more data structures we will be discussing soon
- Needed to by some functions to pass back information (like input from the user)
- Used for file handling

```
int* ptr2 int** ptr3;
```

```
*ptr1 + *ptr2;
```

[illegible]

```
ptr1 + 1;
```

Code question - Demo 4