Introduction to C
Introduction to C

- A simple C Program
  - Variable Declarations
  - `printf()`
- Compiling and Running a C Program
- `sizeof` Program
  - `#include`
- What is `True` in C?
  - `if` example
- Another C Program
  - `#define`
  - `scanf()`
Introduction to C

- Another C Program (continued)
  - *for loop*
  - Promotion

- Other C topics
  - Increment and Decrement Operators
  - Casting
  - Operator Precedence
  - Value of Assignment Operator
Variables

- Variable names correspond to memory locations in memory. Every variable has a **type**, a **name** and a **value**.

```
int i;
i = 4;
```

(\textit{the address of } i ) \quad \& i
Two components of `printf` statement:
- Formatting template `{within quotes}`
- Argument list - variables separated by commas.
printf

```c
int main()
{
    ...
    printf("%d %f %c\n", i, fvar, ch);
}
```

Formatting template:
- Argument list matches up with `\%`
- Some of the argument types:
  - `\%d` integers
  - `\%f` floating-point numbers
  - `\%c` characters
printf

Width of variable printing:

- `%4d` – decimal integers at least 4 digits wide
- `%5f` – floating point at least 5 digits wide
- `%6.2f` – floating point at least 6 digits wide with at least 2 after the decimal point

```c
int main()
{
    ...
    printf("%4d %5f %6.2f\n", i, fvar, f2var);
}
```
A Simple C Program

/* Example of a simple C Program */
#include <stdio.h>
int main()
{
    int i;
    char c, s;
    i = 2303;
    c = 'C';
    s = 'S';
    printf("\nHello\n");
    printf(" %c%c %d Students!!\n", c, s, i);
    return 0;
}
Compiling and Running simple

%ls
simple.c
%gcc simple.c
%ls
a.out simple.c
%./a.out

Hello CS 2303 Students!!
%

Alternate Version

%ls
simple.c
%gcc -o simple simple.c
%ls
simple simple.c
%./simple
/* Fig. 7.17: fig07_17.c
   Demonstrating the sizeof operator */
#include <stdio.h>

int main( void )
{
    char c;
    short s;
    int i;
    long l;
    float f;
    double d;
    long double ld;
    int array[20]; /* create array of 20 int elements */
    int *ptr = array; /* create pointer to array */

Figure 7.17 (part 1)
sizeof Operator

```
17    printf("    sizeof c = %d\tsizeof(char) = %d"
18        \n        sizeof s = %d\tsizeof(short) = %d"
19        \n        sizeof i = %d\tsizeof(int) = %d"
20        \n        sizeof l = %d\tsizeof(long) = %d"
21        \n        sizeof f = %d\tsizeof(float) = %d"
22        \n        sizeof d = %d\tsizeof(double) = %d"
23        \n        sizeof ld = %d\tsizeof(long double) = %d"
24        \n        sizeof array = %d"
25        \n        sizeof ptr = %d\n",
26                sizeof c, sizeof(char), sizeof s, sizeof(short), sizeof i,
27                sizeof(int), sizeof l, sizeof(long), sizeof f,
28                sizeof(float), sizeof d, sizeof(double), sizeof ld,
29                sizeof(long double), sizeof array, sizeof ptr);
30
31    return 0; /* indicates successful termination */
32
33 } /* end main */
```

Figure 7.17 (part 2)

from typelen.c

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>12</td>
</tr>
</tbody>
</table>
/* check to see what conditional does with negative integers */

int main ()
{
    int i = 0; // zero is the only value for false in C
    if (i) printf("%d = true\n", i);
    else
        printf("%d = false\n", i);
    i = 4;
    if (i) printf("Positive integer %d = true\n", i);
    else
        printf("Positive integer %d = false\n", i);
    i = -4;
    if (i) printf("Negative integer %d = true\n", i);
    else
        printf("Negative integer %d = false\n", i);
    return 0;
}

$./a.out
0 = false
Positive integer 4 = true
Negative integer -4 = true
```c
#define SIZE 5
int main ()
{
    int i, start, finish;
    float celsius;
    scanf("%d", &start);
    finish = start + SIZE;
    for (i=start; i<finish; i++)
    {
        celsius = (5.0/9.0) * (i - 32.0);
        printf("%3d %6.1f\n", i, celsius);
    }
    return 0;
}
```
```c
#define SIZE 5
int main ()
{
    int i, start, finish;
    float celsius;

    scanf("%d", &start);
    finish = start + SIZE;
    for (i=start; i<finish; i++)
    {
        celsius = (5.0/9.0)*(i - 32.0);
        printf("%3d %6.1f\n", i, celsius);
    }
    return 0;
}
```
```c
#define SIZE 5
int main ()
{
    int i, start, finish;
    float celsius;

    scanf("%d", &start);
    finish = start + SIZE;
    for (i=start; i<finish; i++)
    {
        celsius = (5.0/9.0)*(i - 32.0);
        printf("%3d %6.1f\n", i, celsius);
    }
    return 0;
}
```

```
$./a.out
30
30 -1.1
31 -0.6
32 0.0
33 0.6
34 1.1
```
Other C Topics

- Increment and decrement operators
- Casting operator \((\text{type})\)
- Operator precedence
- Danger – mistake in the value of the assignment operator
- Variable scope
- Switch
- Conditional operator `?:`

Will cover later !!
## Increment and Decrement Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Sample expression</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>++a</td>
<td>Increment a by 1, then use the new value of a in the expression in which a resides.</td>
</tr>
<tr>
<td>++</td>
<td>a++</td>
<td>Use the current value of a in the expression in which a resides, then increment a by 1.</td>
</tr>
<tr>
<td>--</td>
<td>--b</td>
<td>Decrement b by 1, then use the new value of b in the expression in which b resides.</td>
</tr>
<tr>
<td>--</td>
<td>b--</td>
<td>Use the current value of b in the expression in which b resides, then decrement b by 1.</td>
</tr>
</tbody>
</table>

**Fig. 3.12**

Increment and decrement operators
Casting

- Cast is a unary operator.
- Cast is often useful when an iteration index is used in mixed type arithmetic.
- Later, it will be important to make sure arguments passed are properly matched between called and calling routines.

Example:
```c
int total, count;
float average;
...
average = (float) total / count;
```

When in doubt, be conservative and use cast to be sure!
## Fig 4.16 Operator Precedence

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>++ (postfix)</code></td>
<td>right to left</td>
<td>postfix</td>
</tr>
<tr>
<td><code>-- (postfix)</code></td>
<td>right to left</td>
<td>postfix</td>
</tr>
<tr>
<td><code>+</code></td>
<td>right to left</td>
<td>unary</td>
</tr>
<tr>
<td><code>-</code></td>
<td>right to left</td>
<td>unary</td>
</tr>
<tr>
<td><code>!</code></td>
<td>left to right</td>
<td>cast</td>
</tr>
</tbody>
</table>
| `
`                | left to right | multiplicative |
| `*`                | left to right | arithmetic |
| `/`                | left to right | arithmetic |
| `%`                | left to right | arithmetic |
| `<`                | left to right | relational |
| `<=`               | left to right | relational |
| `>`                | left to right | relational |
| `>=`               | left to right | relational |
| `==`               | left to right | equality   |
| `!=`               | left to right | equality   |
| `&&`               | left to right | logical AND |
| `||`               | left to right | logical OR |
| `?:`               | right to left | conditional |
| `=` `+=` `-=` `*=` `=/` `%=` | right to left | assignment |
| `,`                | left to right | comma      |
The value of assignment is the same as the contents deposited into the variable type on the left.

Note: There are several potential dangers here - especially when the programmer creates new types!!

Bad Examples (for now):

if ( i = 0 )  
if ( i = 4 )

What is the problem ??
This presentation covers many important C topics **quickly** including:

- Declaration of variable types
  - memory allocation by type
  - The address of a variable &
- `printf()` , `scanf()`
- C arithmetic (operators, precedence, casting, promotion, assignment value)
- C booleans (true and false)
- `if`
- Preprocessor directives
  - `#define`, `#include`
- `for`

You are now ready to due lab 1 and once we cover functions everyone should be able to due Program 1.