Introduction to C
Introduction to C

- A '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.

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

(\textit{the address of } i) \& i

4

printf

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

- Two components:
  - Formatting template {within quotes}
  - Argument list - variables separated by commas.
int main()
{

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

\textbf{Formatting template:}
\begin{itemize}
\item Argument list matches up with \texttt{\%}
\item Some of the argument types:
  \begin{itemize}
  \item \texttt{\%d} integers
  \item \texttt{\%f} floating-point numbers
  \item \texttt{\%c} characters
  \end{itemize}
\end{itemize}
printf

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

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
/* Example of a simple C Program */
int main()
{
    int i;
    float var;
    char c, s;
    i = 2303;
    c = 'C';
    s = 'S';
    printf("\nHello");
    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
Figure 7.17 (part 1)
Figure 7.17 (part 2)

sizeof operator

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

sizeof c = 1
sizeof s = 2
sizeof i = 4
sizeof l = 4
sizeof f = 4
sizeof d = 8
sizeof ld = 8
sizeof array = 80
sizeof ptr = 4
```

from typelen.c

char 1
short 2
int 4
long 4
long long 8
float 4
double 8
long double 12
/* 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
Another C Program

```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;
}
```
Another C Program

```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
", i, celsius);
    }
    return 0;
}
```
Another C Program

```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
```

example of 'promotion'
Other C Topics

- Increment and decrement operators
- Casting operator (type)
- Operator precedence
- Danger :: the value of the assignment operator
- Variable scope
- Switch
- Conditional operator ?:
### 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 / counter;
```

*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>++</code> (postfix)</td>
<td>right to left</td>
<td>postfix</td>
</tr>
<tr>
<td><code>--</code> (postfix)</td>
<td>right to left</td>
<td>unary</td>
</tr>
<tr>
<td><code>*</code> / <code>%</code></td>
<td>left to right</td>
<td>multiplicative</td>
</tr>
<tr>
<td><code>+</code> - <code>!</code></td>
<td>left to right</td>
<td>arithmetic</td>
</tr>
<tr>
<td><code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code></td>
<td>left to right</td>
<td>relational</td>
</tr>
<tr>
<td><code>==</code> <code>!=</code></td>
<td>left to right</td>
<td>boolean</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>left to right</td>
<td>logical AND</td>
</tr>
<tr>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>?:</code></td>
<td>right to left</td>
<td>conditional</td>
</tr>
<tr>
<td><code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code></td>
<td>right to left</td>
<td>assignment</td>
</tr>
<tr>
<td><code>,</code></td>
<td>left to right</td>
<td>comma</td>
</tr>
</tbody>
</table>
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!!

Examples (for now):

\[
\begin{align*}
\text{if ( } i = 0 \text{ )} & \quad \text{if ( } i == 0 \text{) } \\
\text{if ( } i = 4 \text{) } & \quad \text{if ( } i == 4 \text{) }
\end{align*}
\]

What is the problem ??
Review/Summary

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.