Programming in C for Engineers
Lecture 2

Statements

Dr. Ahmed M. Attiya
Statements

• Much of our programming will consist of writing statements.
• Examples include declaration statements, assignment statements, and function calls, each of which is terminated by a semicolon.
• One thing to keep in mind when looking at a program is that the code is processed from the top down.
Declaration Statements

Declaration statements are when we declare variables for use. In C89, variables must be declared at the top of the block in which they are used.

Example:

    int cost;
    int age = 42;

Both of these statements allocate memory for use. The second statement also assigns an initial value to the variable.
Assignment Statements

Assignment statements have a left side and a right side. The right side could contain a single value, a complicated expression, or a function call. In each case, the right side will ultimately reduce to a single value, which is then assigned to the variable named on the left side.

Example:

```c
int a, b;

a = 14;
b = a + 10;
```
Assignment

Many times we wish to modify the value of a variable (e.g., add, subtract, etc.).

Example: We want to add 5 to the value of a. We can use

\[ a = a + 5; \]

Remember: The statement

\[ \text{some\_variable} = \text{expression}; \]

is assigning the value of \text{expression} to \text{some\_variable}; this isn't an equation in the mathematical sense.
Operators – Assignment

The following assignment operators are available in C:

\[\begin{align*}
&\text{+=} & \text{addition} \\
&\text{-=} & \text{subtraction} \\
&\text{*=} & \text{multiplication} \\
&\text{/=} & \text{division}
\end{align*}\]

These are an alternative to what we saw on the previous slide.

Example: Instead of
\[a = a + 5;\]
we could use
\[a += 5;\]
Assignment examples

```c
int a = 3;
int b = 5;

printf("a has a value of \%d\n", a);

a = a + b;
printf("now a has a value of \%d\n", a);

/* we could have done this instead */
a += b;
printf("now a has a value of \%d\n", a);
```

**Output**

- a has a value of 3
- now a has a value of 8
- now a has a value of 13
Functions

Functions are collections of computer code that perform a task. In order to use a function, we call it by name and provide it with whatever information is necessary to perform the task.

Example: We can determine the sine of $x$ using

$$y = \sin(x);$$

instead of writing the code for the mathematical function

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

We’ll discuss how to write our own functions in a future lecture.
Input / Output Functions

We need to tell the compiler where to find the information it needs to include the function code in the program. This is why when we use `printf()` we put `#include <stdio.h>` at the top of the file.

Example

```c
#include <stdio.h>

int main(void)
{
    int some_variable = 2;
    printf("some_variable is %d\n", some_variable);
}
```

Output:

```
some_variable is 2
```
More about printf()

We would be severely limited if we had to know exactly what our programs would print at the time we wrote the code.

We can use variables, whose values may be unknown at the time we write our programs, in our calls to printf().

We use **format specifiers** to indicate where our variable will be located in the printed output and how it will appear.
printf() Format specifiers

#include <stdio.h>

int main(void)
{
    int a = 23, b = 1000;

    printf("a is %d\n", a);
    printf("b is %d\n\n", b);

    printf("a is %5d\n", a);
    printf("b is %5d\n", b);
}

produces

    a is 23
    b is 1000

    a is    23
    b is   1000
printf() Format specifiers

We use \f for floating point numbers.

```c
#include <stdio.h>

int main(void)
{
    double c = 123.456789;

    printf("%f\n", c);
    printf("%6.4f\n", c);
    printf("%10.4f\n", c);
}

produces

123.456789
123.4568
123.4568
```
# Format specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>Integer</td>
<td>Signed decimal integer</td>
</tr>
<tr>
<td>%u</td>
<td>Integer</td>
<td>Unsigned decimal integer</td>
</tr>
<tr>
<td>%f</td>
<td>Float</td>
<td>Signed value of the form ([-]dddd.dddd). The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.</td>
</tr>
<tr>
<td>%e</td>
<td>Float</td>
<td>Signed value of the form ([-]d.dddd or e[+/-]ddd)</td>
</tr>
<tr>
<td>%c</td>
<td>Character</td>
<td>Single character</td>
</tr>
<tr>
<td>%s</td>
<td>String</td>
<td>Expects char *. Prints characters until a null-terminator is encountered or precision is reached</td>
</tr>
</tbody>
</table>
printf() Flags

\n (newline)
\t (tab)
\v (vertical tab)
\f (new page)
\b (backspace)
scanf()

Read formatted data from stdin
Reads data from stdin and stores them according to the parameter format into the locations pointed by the additional arguments. The additional arguments should point to already allocated objects of the type specified by their corresponding format tag within the format string.

```c
/* scanf example */
#include <stdio.h>
int main ()
{
    char str [80];
    int i;
    printf ("Enter your family name: ");
    scanf ("%s",str);
    printf ("Enter your age: ");
    scanf ("%d",&i);
    printf ("Mr. %s, %d years old.
",str,i);
    return 0;
}
```
**gets()**

Get string from stdin

Reads characters from stdin and stores them as a string into `str` until a newline character (`\n`) or the End-of-File is reached.

The ending newline character (`\n`) is not included in the string.

```c
/* gets example */
#include <stdio.h>
int main()
{
    char string[256];
    printf("Insert your full address: ");
    gets(string);
    printf("Your address is: %s\n", string);
    return 0;
}
```
**atof() , atoi()**

**atof()** Converts string to double

**atoi()** Converts string to an integer

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main ()
{
    double n,m;
    double pi=3.1415926535;
    char szInput [256];
    printf ( "Enter degrees: " );
    gets ( szInput );
    n = atof ( szInput );
    m = sin (n*pi/180);
    printf ( "The sine of %f degrees is %f\n" , n, m );
    return 0;
}
```

/* atof example: sine calculator */
**cin and cout**

**cin is an input stream**

- cin object of type istream
- istream class is defined in file iostream
  - #include <iostream>

```cpp
int N(0); // number of datapoints
double sum(0), read_space;

while( cin >> read_space ) {
    sum += read_space;
    N++;
}

if( N ) {
    cout.setf( ios::fixed );
    cout.precision( 3 );
    cout << "Average: " << sum/N << endl;
}
```
cin and cout

cout is an output stream

- cout object of type ostream
- ostream class is defined in file iostream
  - #include <iostream>
- Member functions!
  - cout.setf( ios::fixed );
  - cout.precision(3);

```cpp
int N(0); // number of datapoints
double sum(0), read_space;

while( cin >> read_space ) {
    sum += read_space;
    N++;
}

if( N ) {
    cout.setf( ios::fixed );
    cout.precision(3);
    cout << "Average: " << sum/N << endl;
}
```