Statements

Darin Brezeale

The University of Texas at Arlington

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.

Operators – Assignment

The following assignment operators are available in C:

+=	addition
-=	subtraction
$*$=	multiplication
/=	division

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

Example: Instead of

```
    a = a + 5;
```

we could use

```
    a += 5;
```

Libraries of Functions

The C language (C89 standard) only has 32 keywords, none of which deal with common tasks such as printing, reading / writing files, performing complex mathematical operations, and so forth.

Modern C compilers include functions for performing these common tasks in the Standard C Library.

The function we will use most often in this course is `printf()`.

`printf()` example

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

```
#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.

More about `printf()` cont.

```
#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

Here are a few examples of the many format specifiers available.

```
#include <stdio.h>

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

    printf("%d %d\n", a, b);
    printf("%4d %d\n", a, b);
    printf("%04d %d\n", a, b);
    printf("%-4d %d\n", a, b);
}
```

produces

```
23 1000
  23 1000
0023 1000
23   1000
```

`printf()` Format specifiers

We use `f` for floating point numbers.

```
#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
```

`scanf()`

When we wish to prompt the user for a number or string, we can use `scanf()`.

```
#include <stdio.h>

int main(void)
{
    int age;

    printf("What is your age:  ");
    scanf("%d", &age);

    printf("You are %d years old.\n", age);
}
```

`scanf()` cont.

Note the following about `scanf()`:

- it doesn't tell the user what is wanted, so the `printf()` statement is needed for this
- it requires a format specifier for each variable to store values
- the format specifiers are largely the same as for `printf()`; the main exception is that variables of type `float` use `%f` and variables of type `double` use `%lf` (that's the letter L)
- variable names for numeric types must be preceded by an ampersand