Lecture 2. An Introduction to C

- **Everyone's first C program: `hello.c`**

```
/* Everyone's first
    C program. */
#include <stdio.h>

int main(void) {
    printf("Hello world!\n");
    return 0;
}
```

- **To compile, load, and execute `hello.c`:**

```
% lcc hello.c
% a.out
Hello world!
%
```

slanted font indicates what you type

- **Writing and running C programs involves at least 3 steps:**

1. **Using an *editor* (`emacs`) to create a *file* that contains the program (`hello.c`)**
2. **Using a *compiler* (`lcc`) to translate the program from C to 'machine language'**
3. **Issuing a *command* (`a.out`) to execute the machine-language program**

Usually — *OK, always* — you iterate these steps until step 3 is 'correct'

Dissecting hello.c

```
/* Everyone's first
    C program. */
```

`/*` and `*/` enclose *comments*, which document your program or parts of it. The compiler treats a comment as a single space

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

`#include` is a *preprocessor* directive, which causes the compiler to read in standard declarations from the *header file* `stdio.h`

```
int main(void) {
```

Introduces the `main` function, which is where execution begins. `int` is the type of the value returned by `main`, `void` indicates that `main` has no arguments, and the `{` begins the body of the function

```
printf("Hello world!\n");
```

Calls the *standard library* function `printf`, which prints the characters in its *string* argument. `\n` is an *escape sequence* for a new-line character

```
return 0;
```

`main` returns the integer 0, indicating that the program completed successfully

```
}
```

Ends the function `main`

Computing the Sum from 1 to n

```
/*
Compute the sum of the integers
from 1 to n, for a given n.
*/
#include <stdio.h>

int main(void) {
    int i, n, sum;

    sum = 0;
    printf("Enter n:\n");
    scanf("%d", &n);
    i = 1;
    while (i <= n) {
        sum = sum + i;
        i = i + 1;
    }
    printf("Sum from 1 to %d = %d\n", n, sum);
    return 0;
}
```

```
% lcc sum.c
% a.out
Enter n:
100
Sum from 1 to 100 = 5050
%
```

Dissecting sum.c

```
int i, n, sum;
```

This *declaration* introduces three *variables* that can store integers — values of type `int`

```
sum = 0;
```

This *assignment expression* changes the value stored in sum to 0

```
scanf("%d", &n);
```

Calls the *standard library* function `scanf` to read an integer (%d) and store it in n

```
i = 1;
```

Changes the value stored in i to 1

```
while (i <= n) {
    sum = sum + i;
    i = i + 1;
}
```

This *while loop* executes the loop body — the two statements between { and } — repeatedly as long as the value of `i` is less than or equal to the value of n

Expression Evaluation

```
sum = sum + i;
```

This assignment expression means:

> **add the value of `sum` to the value of `i`, then store that result *back* into the variable `sum`**

The meaning of this assignment — its *semantics* — might be clearer if written as

```
sum + i --> sum;
```

but that's not C (or any other language)

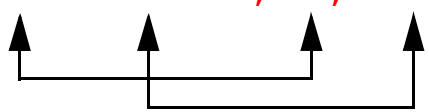
```
i = i + 1;
```

Stores the sum of `i` and 1 back into `i` — increments `i` by 1

```
printf("Sum from 1 to %d = %d\n", n, sum);
```

Calls `printf` to output its first argument; each *conversion specifier* `%d` causes the value of the corresponding following `int` argument to be printed instead

```
printf("Sum from 1 to %d = %d\n", n, sum);
```



Another Example: Printing a Random Pattern

```
/*
Print a NxN random pattern.
*/
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    int i, j, n, bit;

    scanf("%d", &n);
    for (i = 0; i < n; i = i + 1) {
        for (j = 0; j < n; j = j + 1) {
            bit = (rand()>>14)%2;
            if (bit == 0)
                printf(" ");
            else
                printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

```
% lcc pattern.c
% a.out
20
*   **  *        ******
* *   **  ********    *
  *   **  ***   * * *
 **  *      *      **
*      * * *** *** **
    **    * **  *   ****
      ***   *     *** *
*   * *   **  * ******
  * *    * *     *   *
* *   * * ******  **
*    *       **  * ****
**  *****  ***     **
   ***  *    * **  * **
 ***  *         *  **
* **        *****
   ****    **       **  *
     ***  **  **   *****
* **  *  **     **  ****
    **  *    * **** **
*    * * *         ***
%
```

Dissecting pattern.c

```
for (i = 0; i < n; i = i + 1) {
    …
}
```

This *for loop* executes its body (…) n times; it is equivalent to

```
i = 0;
while (i < n) {
    …
    i = i + 1;
}
```

```
for (i = 0; i < n; i = i + 1) {
    for (j = 0; j < n; j = j + 1) {
        …
    }
}
```

These two *nested for loops* execute the body of the inner loop $n \times n = n^2$ times

Dissecting pattern.c, cont'd

```
bit = (rand()>>14)%2;
```

This assignment expression

> **calls the standard function `rand`, which returns a 15-bit *random number*,**
> ***shifts* that number right by 14 bits,**
> **computes the *remainder* of dividing that number by 2;**
>
> **so, `bit` is assigned 0 or 1**

```
if (bit == 0)
    printf(" ");
else
    printf("*");
```

This *if-else statement* compares `bit` with 0;
it prints a space if `bit` is equal to 0, or an asterisk if `bit` is not equal to 0

For More Information

- **Check out the other texts on C programming (on reserve in the Eng. Library):**

  Kelley and Pohl, *C by Dissection: The Essentials of C Programming*, 3/e

  Kelley and Pohl, *A Book on C: Programming in C*, 3/e

  Roberts, *The Art and Science of C: An Introduction to Computer Science*

- **Check out the reference books (on reserve):**

  Harbison and Steele, *C: A Reference Manual*, 4/e

  Kernighan and Ritchie, *The C Programming Language*, 2/e

- **Cruise the sample programs on the COS 126 Help! Web page:**

  follow the 'Sample Programs' link to `hello.c`, `sum.c`, etc.