Introduction to C

This lecture is an introduction to writing computer programs in a popular computer language called C. Due to time constraints we will cover only the minimum subset of the C language that you will need to complete the lab assignments.
After this lecture you should be able to:

- *explain the following terms: file, file name, file type, compiler, source, executable, variable, statement, operator, precedence*
- *evaluate expressions involving integer variables, constants, and the operators described in this lecture*
- *write a simple C program including a main() function declaration, integer variable declarations, and the following statements: expression, if/else, and while*
- *predict the result of executing such a program*

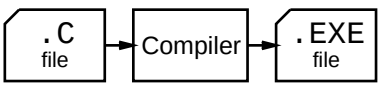
Compiling a C Program

All information in a computer is stored in hard disks (fixed inside the computer) and floppy disks (which can be removed). This information is organized in files. Each file has a unique name which is used to identify it. This file name is composed of a name (up to 8 characters long), a period and an extension (up to 3 characters long). The name usually has some mnemonic value while the extension identifies the type of file.

Exercise: What are the file names and extensions of the following file names: TC.EXE, LAB1.C, IOLIB.OBJ, CS.LIB)?

Since microprocessors don't understand C, we use a program called a compiler to translate ("compile") the the C program (the "source code") into a language that the machine can understand ("machine language").

Programs in C have the extension `.C` while machine-language programs have the extension `.EXE`. Therefore we can say that a compiler is a program that converts `.C` files into equivalent `.EXE` files.



Exercise: If you compile a C program in the file `ASG.C` what would be the file name of the resulting executable?

The Structure of a C program

Here's a simple C program:

```
main ()
{
   int i ;

   i = 0 ;
   while ( i < 4 ) {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
   }
}
```

If you compile and then execute this program the computer will print the integers between 0 and 3.

The first two lines are required by all C programs and define a function called `main`. The braces ("curly brackets") on the second line and the last line mark the start and end of the function.

The computer executes the statements in order, starting with the first statement in the function called `main` and ending after the last statement in the function `main` is executed.

The line `int i ;` is a statement that declares an `int`eger variable called `i`. A variable is an area of the computer's memory that is used to store numbers.

The line `i = 0 ;` is another statement – an expression statement. This particular expression statement sets the value of the variable `i` to zero.

The line `while ( i < 4 )` is yet another type of statement, a `while` statement. The computer repeatedly evaluates the expression within the

parentheses and executes the statements between the braces while this expression is true.

The line starting with `printf()` is another expression statement. In this case the expression statement contains only the name of another function which is to be executed. This particular function (`printf`) causes the value of the variable `i` to be printed on the screen.

The next line is another expression statement that increments the value of the variable `i`.

Exercise: What do you think would be printed out if the order of the two statements within the while 'loop' was interchanged?

The following sections give more detailed information about variables, expression statements, and the two basic types of control statements: `if/else` and `while` statements.

Variable Declarations

We will only need to use integer variables. These come in two sizes called `char` and `int`. Each size can be signed or unsigned and can take on the range of values shown below.

variable type	size (bits)	unsigned range	signed range
`char`	8	0 – 255	-128 – 127
`int`	16	0 – 65536	-32768 – 32767

Variables have to be "declared" at the beginning of the function where they are used. A variable is only in existence while statements in that function are being executed and the values of variables are forgotten when the function terminates.

Here are some examples of variable declarations:

```
int day, month, year ;

unsigned int cycles ;

char ppc ;
```

Exercise: What are the possible values for each of these variables?

The first character of a variable or function name must be a letter or underscore, followed by other letters, underscores or digits. Case is significant: `i` and `I` are two different variables.

There are certain reserved names that can't be used for variable names. Keywords such as `if`, `while`, `int`, etc. are reserved. A list of reserved keywords is available from the Turbo C help index screen (type F1 twice and select "Keywords").

Exercise: Make up three valid and three invalid variables names.

Since computers can only work with numbers, letters must be converted to numbers when they are read into the computer and converted back to characters when they are displayed. The standard mapping of characters to numbers is called "ASCII." For example, the number used to represent the letter `'a'` is 97 and the number for the space character is 32. ASCII encoding only defines values from 0 to 127 so a `char` variable is typically used to store the code for a character.

Expressions

Expressions describe how new values are computed from the values of existing variables and constants. Expressions are built up from variables, constants and operators.

Constants are similar to variables except that their values cannot be changed. Integer constants can be expressed as a number (e.g. 12). We can also specify the ASCII value of a particular character by surrounding that character with single quotes (e.g. `'e'`).

Operators are characters denoting operations to be performed on variables such as addition, comparison, and assignment.

For now, we will only study a few of the operators available in C. The following is a list of the most common operators and examples of expressions using them:

- the arithmetic operators, `* / + -` , result in the product, quotient, sum and difference of the values on the left and right. As usual, multiplication and division are performed before ("have higher precedence than") addition and subtraction. Otherwise operations are done left to right.

```
1 + 3 * 5 / 4
```

- parentheses are not really operators but are used to change the order in which parts of an expression are evaluated

```
( 2 + ' ' ) * 3
```

- comparison operators (`< > >= <= == !=`) compare the value on the left and right of the operator. the result is the value 0 if the comparison is false, 1 if it is true. Comparison operators have lower precedence than the operators described above.

```
( -1 < ( 3 != 2 ) ) * ( 5 > 1 )
```

- the assignment operator, `=`, assigns the value of the expression on the right to the variable on left. The result is the value that is assigned. Assignment operators have lower precedence than the operators given above.

```
b = 5
c = b - ( a = 3 )
```

We will discuss other operators as we need them. A complete list of operators and their precedence is available from the Turbo C help screen (type F1 twice and select "Precedence").

Exercise: What are the values of each of the above expressions (the ASCII value for a space is 32)?

Statements

Expression Statements

The most common C statement is an expression statement. It is simply an expression terminated by a semicolon. For example:

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

`if/else` statement

This statement causes only one of two groups of statements to be executed depending on the value of an expression. If the expression is non-zero the statements in the `if` portion are executed, otherwise the statements in the else portion (if any) are executed.

```
if ( <expression> ) {
   <statements>
} else {
   <statements>
}
```

Exercise: What is the value of 'i' after executing the following statements?

```
b = 4 ;
if ( b * b == 4 ) {
   i = 1 ;
} else {
   i = 0 ;
}
```

`while` statement

As explained previously, the `while` statement is used to repeatedly execute a group of statements while the controlling expression is non-zero.

```
while ( <expression> ) {
   <statements>
}
```

Exercise: How many times will the statement inside the '`while` loop' be executed?

```
b = 2 ;
while ( b <= 16 ) {
   b = b * 2 ;
}
```

Functions

C has many pre-defined functions to do formatted input and output, compute values of mathematical functions, manipulate character strings, etc.

Functions can be invoked within expressions just like any other value. The value of a function depends on the definition of the function. Many functions also have side-effects such as printing a string on the display.

Some examples of built-in functions are the `printf()` function for formatted output, `getchar()` to read a character from the keyboard, or `isdigit()` to test whether a character is a decimal digit.

In order to use C's built-in functions one or more "`#include`" lines must be used at the start of a program. These lines tell the compiler to load files ("include files") that define particular groups of built-in functions. For example, the line `#include <stdio.h>` lets you use the standard i/o functions in your program, while the line `#include <string.h>` lets you use the standard functions that manipulate strings.

The Turbo C on-line help can give you details on any particular function (enter the function name in the edit window, put the cursor on the function name and press control-F1).

Comments and White Space

"White space" characters include spaces, tabs and the invisible characters that mark the end of a line. White space is optional except where necessary to avoid ambiguity, such as between `int` and the variable name. In places where white space is required, you may use any type and any number of white space characters.

Comments are notes included in the source code to help readers understand the program. Comments are treated as white space by the compiler. Comments are delimited by the character pairs `/*` and `*/`.

Exercise: Could the first program in this lecture be rewritten all on one line?

Example Program

Here's another example of a C program:

```
/* Sample program for introduction to C
   Ed Casas
   97-9-4
*/

#include <stdio.h>

/* print the ASCII values of all characters that are
   digits (ASCII assigns codes for digits in numerical
   order). */

main ()
{
   int i ;

   i = 0 ;
   while ( i <= 127 ) {

      if ( i >= '0' && i <= '9' ) {
         printf ( "Character number %d is a digit.\n",
             i ) ;
      }

      i = i + 1 ;
   }
}
```

Exercise: What will be printed by the program above?