C Tutorial
Session #1

- History of C
- Why we use C
- First C program
- Compile and Run your program
- Functions, Basic Types, printf ()
- Type storage
- Strings and characters
- Operators
- Loops

History of C

- First developed at Bell Labs during the early 1970's
- Derived from a computer language named B
- Initially designed as a system programming language under UNIX
- Function based, weakly typed, formal syntax grammar and case-sensitive
- With its power comes the ability to create havoc

Why C

- Despite the presence of many other programming languages, C is still widely used as a system programming language
- C is extensively used in the area of Operating Systems, Compilers, Embedded Systems, and Scientific Computing
- Features: efficient, flexible, low level programming readily available

Prerequisites

- A Linux/Unix machine
- Have a basic idea about the file system
- Basic programming concepts

First C Program

```
#include <stdio.h>

int main()
   {
        printf("Hello World!\n");
        return (1);
   }
```

Line 1: #include <stdio.h>

- #include is a pre-processor directive
- stdio.h is a standard C library header file used for I/O
- Header files are inserted into your program source by the C preprocessor, before the actual compilation
- Header files typically contain declarations and definitions of functions, types, and constants that are used in your program.

Line 2: int main()

- This statement declares the main function
- A C program can contain many functions but must always have one main function
- main () function is the starting point of your program

Line 3: {

- This opening bracket denotes the start of the function or body of code
- All functions should start with { and end with }

Line 4: printf("Hello World!");

- printf is a function from a standard C library that is used to print strings to the standard output
- The "\n" is a special format modifier that tells the printf to output a new line.
- Function calls must end with a semi-colon

Line 5: return (1);

- This closing bracket denotes the end of the function.

Line 6: }

- This closing bracket denotes the end of the function.

Compile your program

- You can type in your shell:
  - gcc –c helloworld.c *(compile)*
  - gcc –o helloworld helloworld.c *(compile & link)*

Run your program

- In your shell, type:

  ./helloworld

More on functions

- A function is a self-contained module of code that can accomplish some task.
- Example:

```
int myfunction(int a)
{
    return (a + 1);
}
```

Function Structure

Return type Function name (arguments)

{

    Function body

}

Assignment 1

- Create a program that prints out:

  Hello everybody, my name is ….

Basic Types

- int – 2 byte integer
- long – 4 byte integer
- float – 2 byte floating point (real)
- double – 4 byte floating point
- char – single byte character
- unsigned char – single unsigned byte
  (note: unsigned is a qualifier that can be applied to other types as well)

Assignment 2

- printf function can print variables

  Eg. printf("the value of A is %d\n", A);

- Read the above example, and then write a program that computes the sum of 2 integers and displays the computation and result.

  Sample output:

  2 + 2 = 4

printf () Format Specifiers

Code	Format
%c	character
%d	signed integers
%E	scientific notation, with a uppercase "E"
%f	floating point
%s	a string of characters
%u	unsigned integer
%X	unsigned hexadecimal, with uppercase letters
%p	a pointer
%%	a '%' sign
\n	New line character
\" \'	Quotation characters
\l	Linefeed character
\0	Null ‘character’

Other optional details can be specified, such as the precision. A format specifier may look as follows:

% flags width .precision size type

Flag	Description
+	for a number, that a sign should always be included, even if it is positive
<sp ace >	space should be prepended to the output of a number if the sign is positive. Of course this flag is ignored if the previous flag is also used
#	leading zero be used for an octal number (format specifiers o and O), a leading 0x or 0X be used for a hexadecimal number (format specifiers x and X) or that a decimal point always be included for the floating point types
0	leading zeroes should be used to pad a number

Examples:

```
float scale = 1.57f;
int num2 = 123;
int prec = 4;

printf("Scale is %7.3f\n",scale); //Scale is   1.570
printf("Scale is %5.d\n",num2);   //Scale is   123
printf("Scale is %.5f\n",scale);  //Scale is 1.57000
printf("Num2 is %.5d\n",num2);    //Num2 is 00123
```

Code Example

```
#include <stdio.h>    /* include file for
                         standard i/o      */

long factorial (int);/* function prototype*/

//*****************************************
//  function: factorial
//  purpose:  compute the factorial of the
//            supplied number
//  inputs:   int nInput
//  return:   long factorial value
//*****************************************

long factorial(int nInput)
{
   int ctr;
   long result = 1;

   for( ctr = 1 ; ctr <= nInput ; ctr++ )
      result = result * ctr;

   return ( result );
}
```

```
//*****************************************************
//  function: main
//  purpose:  entry point of program
//            prompts for number from user and calls
//            factorial function then outputs the result
//            to the screen.
//  inputs:   none
//  return:   integer result 1 = success, 0 = failure
//*****************************************************

int main()
{
   int number;
   long fact = 1;

   printf
      ("Enter a number to calculate it's factorial\n");
   if (scanf("%d", &number) != 1)
      {
      printf (“Invalid number entered\n”);
      printf (“usage: factorial <num>\n”);
      return (0);
      }

   printf ("%d! = %ld\n", number, factorial(number));

   return 1;
}
```

Introducing Debugging

- Syntax Errors
- Semantic Errors
- Runtime Errors

Common Errors

```
#include <stdio.h>
int main(void)
(
    int n  int n2  int n3;
    /* this program has several errors
    n = 5;
    n2 = n * n;
    n3 = n2 * n2;
    printf("n =  %d  n squared = %d  n cubed = %d\n"  n  n2  n3);
    return 0;
)
```

Data and C

- Programs work with data.
- A common C program works like this
  - You feed data to your program.
  - Your program does something with the data.
  - Your program gives the result back to you.

Example Reading Input from Keyboard

```
int main( )
{
   float weight;
   scanf(“%f”  &weight)
   printf(“george’s weight is %f.\n” weight);
   return 0;
}
```

Float and Int

- Bits Bytes and Words.
- The integer

+/-	0	0	0	0	1	1	1

- The Float

+/-	.314159	1

Type Char

- The char type is used for storing characters such as letters and punctuation marks.
- Char type actually stored as integer (length 1 byte)
- Example

```
char broiled;   //declare a char variable.
broiled = 'T';  //correct
broiled = T;    //error
broiled = "T";  //error
```

Character strings

- An example of a string

  “I am a string.”
- A character string is a series of one or more characters.
- Strings are enclosed by double quotation marks.

Character strings(2)

- C has no special string type
- A string is an array of chars
- Characters in a string are stored in adjacent memory cells
- Standard C string functions depend on a null terminated string

Character strings (3)

- String declaration

```
char name[5];
```

- Notice the difference

```
char ch;
char name[5];
```

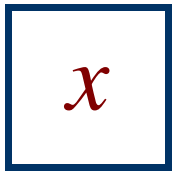
- Every char of name can be accessed as name[i]
- Arrays are indexed from 0 so the first character in a string is `string[0]`

Sample program

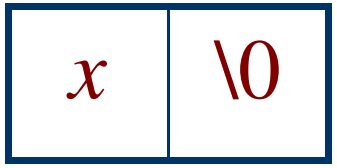
```
int main()
{
    char name[40];
    printf("what is your name?");
    scanf("%s"  name);
    printf("hello  %s.\n"  name);
    return 0;
}
```

Strings versus characters

- Character ‘x’



- String “x”



Common String functions

- Remember to include <string.h>
- *strlen* // returns the length of the string
- *strcpy* // string copy
- *strcmp* // string compare
- *strcat* // append one string to another
- *sprintf* // same as *printf* but prints to a string
- *sscanf* // same as *scanf* but reads from a string

Question

- What does strlen () return if applies to the following string? Why?

  “hello everybody\0 my name is dr. Evil.”

Operators

- Arithmetic

addition	`+`
subtraction	`-`
multiplication	`*`
division	`/`
modulus	`%`

integer addition is not the same as floating point, be careful with types

- Assignment `=`

```
eg. Number = 23;
```

- Augmented assignment

`+=` `-=` `*=` `/=` `%=` `&=` `|=` `^=` `<<=` `>>=`

eg.

```
Number += 5;
```

is equivalent to

```
Number = Number + 5;
```

Operators

- bitwise logic
  - NOT `~`
  - AND `&`
  - OR `|`
  - XOR `^`
- bitwise shifts
  - shift left `<<`
  - shift right `>>`
- boolean logic
  - Not `!`
  - And `&&`
  - Or `||`

Example:

```
int num1 = 1, num2 = 2, result;

result = num1 && num2; // result = 1
result = num1 & num2;  // result = 3
```

Note: there is no boolean type, non-zero is considered logically true

Operators

- equality testing
  - Equal to **==**
  - Not equal to **!=**
- order relations **< <= > >=**
- conditional evaluation `(expr) ? … result1 : result2;`

  example:

```
int num1 = 5;
result = num1==5 ? 1 : 2 // result = 1
```

  is equivalent to

```
if (num1 == 5) result = 1 else result = 2;
```

  note the difference between

```
num1 = 5;     // assignment
num1 == 5;    // logical test
```

- increment and decrement **++ --**

  order can be important: ++i and i++ are both valid

- object size **sizeof ()** – NOT the same as **strlen ()**

Loops

- for loop

```
for (i = 0; i < max; i++)
    {… body… }
```

- while loop

```
while (expression)
    {… body …}
```

- *do loop*

```
do
{… body… }
while (expression);
```

Example

```
int main( )
{
    int i = 0;
    while (i < 3)
    {
        printf(“%d “  i);
        i = i + 1;
    }
    return 0;
}
```

Question

What if we delete the line?

$i = i + 1;$

Answer…. Infinite loop!

Next Session

- Next Session
- Friday, April 13th
- MacLab, A-level
- Regenstein Library, Room AC001