C Programming
Basic Standard I/O

Standard Input/Output (I/O)

- Preconnected input and output channels between a computer program and its environment(typically a text terminal).
  - Standard input :
    - text input from keyboard
  - Standard output
    - text output written to display
  - Standard error :
    - another text output written to display for error messaging

Standard I/O library

- Library
  - A collection of subroutines (functions) used to develop software
- Standard library
  - Library that is made available in every implementation of a programming language
  - Same interface(parameter type) , same functionality in different systems
- Standard I/O library
  - Standard library for processing I/O

printf function

```
printf(control string,argument list);
```

- Control string contains
  - Literal text to be displayed
  - format specifiers
  - Special characters
- Arguments can be
  - Variable , function, expression, constant
  - # of argument list must match the # of format identifiers

printf example

```
 7 #include <stdio.h>
 8
 9 int main()
10 {
11     int    i = 2;
12     double f = 3.14;
13     char   c = '5';
14
15     printf("i = %d\n", i);
16     printf("f = %f\n", f);
17     printf("c = %c\n", c);
18
19     return 0;
20 }
```

```
Output :
i =           2
f =    3.141593
c =           5
```

`printf` format specifiers

Specifier	*Type*
`%c`	character
`%d`	decimal integer
`%o`	octal integer (leading 0)
`%x`	hexadecimal integer (leading 0x)
`%u`	unsigned decimal integer
`%ld`	long int
`%f`	floating point
`%lf`	double or long double
`%e`	exponential floating point
`%s`	character string

printf examples

```c
#include <stdio.h>

int main()
{
    int    i = 2;
    double f = 3.14159265358979323846;
    char   c = '5';

    printf("i = %10d\n", i);
    printf("f = %10f\n", f);
    printf("c = %10c\n", c);

    return 0;
}
```

```
output :
i =          2
f =   3.141593
c =          5
```

```c
#include <stdio.h>

int main()
{
    double pi = 3.14159265358979323846;

    printf("pi = %10f\n", pi);
    printf("pi = %10.2f\n", pi);
    printf("pi = %10.12f\n", pi);

    return 0;
}
```

```
output:
pi =   3.141593
pi =        3.14
pi = 3.141592653590
```

scanf function

- Accept formatted text input

```
#include <stdio.h>

int main()
{
    int n = 0;

    scanf("%d", &n);
    printf("entered n   = %d\n", n);
    printf("double of n = %d\n", n+n);
    printf("triple of n = %d\n", n+n+n);

    return 0;
}
```

```
Output :
27                          <---- keyboard input
entered n   = 27
double of n = 54
triple of n = 81
```

gets(), puts() functions

- line based string I/O functions
- Prototype
  - **char* gets(char *BUF);**
    - Read characters from standard input until a newline is found
  - **int puts(const char *s);**
    - Writes a string s to the standard output.

```
#include <stdio.h>

#define MAX_LINES 256

int main()
{
        char line[MAX_LINES];
        printf("string input :");
        gets(line);
        printf("the input string is : ");
        puts(line);

        return 0;
}
```

redirection

- Input redirection
  - Gets standard input from a file “inputFile.txt”
  - `program.exe < inputFile.txt`
- Output redirection
  - writes standard output to a file “outputFile.txt”
  - `program.exe > outputFile.txt`
- Combination
  - Gets standard input from a file “inputFile.txt” and writes standard output to a file “outputFile.txt”
  - `program.exe < inputFile.txt > outputFile.txt`

Exercise

- Write a program that converts meter-type height into [feet(integer),inch(float)]-type height. Your program should get one float typed height value as an input and prints integer typed feet value and the rest of the height is represented as inch type. (1m=3.2808ft=39.37inch)
  - Ex) 1.80meter -> 5feet 10.9inch
- use automatic type conversion
  - 1/2 = 0 (?) , 3/2 = 1 (?)

```
(ex)
int a;
float b;
b = 3.6/2.0;
a=b;
printf("a=%d, b=%f\n",a,b);
```

C Programming
Variables , Data Types

First Program

```
#include <stdio.h>
int main()
{
        /* My first program */
        printf("Hello World! \n");

        return 0;
}
```

```
Output :

 Hello World!
```

- C is case sensitive.
- End of each statement must be marked with a semicolon (;).
- Multiple statements can be on the same line.
- *White space (e.g. space, tab, enter, …)* is ignored.

First Program

```
#include <stdio.h>
int main()
{
        /* My first program */
        printf("Hello World! \n");

        return 0;
}
```

```
Output :

 Hello World!
```

- The C program starting point : **main().**
- **main() {}** indicates where the program actually starts and ends.
- In general, braces {} are used throughout C to enclose a block of statements to be treated as a unit.
- *COMMON ERROR: imbalanced number of open and close curly brackets!*

First Program

```
#include <stdio.h>
int main()
{
        /* My first program */
        printf("Hello World! \n");

        return 0;
}
```

```
Output :

 Hello World!
```

- #include <stdio.h>
  - Including a header file `stdio.h`
  - Allows the use of `printf` function
  - For each function built into the language, an associated *header file must be included.*
- printf() is actually a function (procedure) in C that is used for printing variables and text

First Program

```
#include <stdio.h>
int main()
{
        /* My first program */
        printf("Hello World! \n");

        return 0;
}
```

```
Output :

 Hello World!
```

- Comments
  - /* My first program */
  - Comments are inserted between “/*” and “*/”
  - Or, you can use “//”
  - Primarily they serve as *internal documentation for program structure and function*.

Why use comments?

- Documentation of variables, functions and algorithms
- Ex) for each function, explain input and output of the function, and **what** the function does.
- Describes the program, author, date, modification changes, revisions,…

Header Files

- Header files contain definitions of functions and variables
- Preprocessor `#include` insert the codes of a header file into the source code.
- Standard header files are provided with each compiler
- To use any of the standard functions, the appropriate header file should be included.
  - Ex) to use `printf()` function , insert `#include <stdio.h>`
- In UNIX, standard header files are generally located in the /usr/include subdirectory

Header Files

```
#include <string.h>
#include <math.h>
#include "mylib.h"
```

- The use of brackets <> informs the compiler to search the compiler's include directories for the specified file.

- The use of the double quotes "" around the filename informs the compiler to start the search in the current directory for the specified file.

Second Program

```
#include <stdio.h>
#define TAXRATE 0.10
int main () {
        float balance;
        float tax=0.0;     /* declaration + initialization */
        char rate='A';
        int credit_no=1;
        balance = 72.10;
        tax = balance * TAXRATE;
        printf("The tax on %.2f is %.2f\n",balance, tax);
        printf("CREDIT RATE : %d/%c\n", credit_no, rate);

        return 0;
}
```

```
Output :
The tax on 72.10 is 7.21
CREDIT RATE : 1/A
```

Names in C

- Identifiers (variable name)
  - Must begin with a character or underscore(_)
  - May be followed by any combination of characters, underscores, or digits(0-9)
  - Case sensitive
  - Ex) `summary, exit_flag, i, _id, jerry7`
- Keywords
  - Reserved identifiers that have predefined meaning to the C compiler. C only has 29 keywords.
  - Ex) `if , else, char, int, while`

Symbolic Constants

- Names given to values that cannot be changed.
- Use preprocessor directive `#define`

```
#define N 3000
#define FALSE 0
#define PI 3.14159
#define FIGURE "triangle"
```

- Symbols which occur in the C program are replaced by their value before actual compilation

Declaring Variables

- Variable
  - Named memory location where data value is stored
  - Each variable has a certain type (e.g. `int, char, float, …`)
  - Contents of a variable can change
  - Variables must be declared before use in a program
  - Declaration of variables should be done at the opening brace of a function in C. (it is more flexible in C++)

- Basic declaration format
  - *data_type var1, var2, …;*
  - *Examples)*

```
int i,j,k;
float length, height;
```

Data Types

- `char` : 1 byte, capable of holding one character (ascii code)
- `int` : 4 byte (on 32bit computer) integer
- `float` : single-precision floating point
- `double` : double-precision floating point

$min: -2^{8n-1}$
$max: 2^{8n-1} - 1$

type	size	min value	max value
char	1byte	$-2^7 = -128$	$2^7-1 = 127$
short	2byte	$-2^{15} = -32,768$	$2^{15}-1 = 32,767$
int	4byte	$-2^{31} = -2,147,483,648$	$2^{31}-1 = 2,147,483,647$
long	4byte	$-2^{31} = -2,147,483,648$	$2^{31}-1 = 2,147,483,647$

- Min/Max values are defined in `<limit.h>` header file

unsigned type

- Use when representing only positive numbers

Data type	size	min	max
unsigned char	1byte	0	2^8-1 = 255
unsigned short	2 byte	0	$2^{16}-1$ = 65,535
unsigned int	4byte	0	$2^{32}-1$ = 4,294,967,295

Negative integer representation

- signed
- first bit represents the sign of a number
- Rest of bits represent the value of a number
- Negative integer number
  - Represented as 2's complement

number	Bit representation
+5	00000101
1's complement of 5	11111010
2's complement of 5	11111011
-5	11111011

floating point

- real number : significant number + position of decimal point
- Decimal point(.) can be placed anywhere relative to the significant digits of the number
- This position is indicated separately in the internal representation
- Advantage of floating point representation
  - Support much wider range of values
  - Representing 314159265358979.3 vs 3.14159265358793

type	size	min	max
float	4 byte	(7 significant numbers) -1.0E+38	(7 significant numbers) 1.0E+38
double	8 byte	(15 significant numbers) -1.0E+308	(15 significant numbers) 1.0E+308

Ascii Code

Escape character

- Starts with backslash sign
- Indicate special meaning and interpretation

Escape character	meaning
\b	backspace
\t	tab
\n	newline
\r	formfeed
\"	double quote
\'	single quote
\\	back slash

code.c

```
 6  int main()
 7  {
 8      char c;
 9      int i;
10
11      c = 'a';
12      printf("%c %d \n", c, c);
13      c = 'A';
14      printf("%c %d \n", c, c);
15      c = '1';
16      printf("%c %d \n", c, c);
17      c = '$';
18      printf("%c %d \n", c, c);
19      c = '+';
20      printf("%c %d \n", c, c);
21
22      i = 'a';
23      printf("%c %d \n", i, i);
24      i = 'A';
25      printf("%c %d \n", i, i);
26      i = '1';
27      printf("%c %d \n", i, i);
28      i = '$';
29      printf("%c %d \n", i, i);
30      i = '+';
31      printf("%c %d \n", i, i);
32      return 0;
33  }
```

output:

```
a 97
A 65
1 49
$ 36
+ 43
a 97
A 65
1 49
$ 36
+ 43
```

getchar() , putchar()

- **int getchar()**
  - Defined in <stdio.h>,
  - Get one character input from keyboard and return the ascii value
- **int putchar(int c)**
  - Defined in <stdio.h>
  - prints one character provided as a parameter

```
#include <stdio.h>

int main()
{
        int c;

        printf("keyboard input (one character?)");

        c=getchar();

        printf("character input : %c\n",c);
        printf("ascii code : %d\n", c);

        return 0;
}
```

Output :

```
character input : A
ascii code : 65
```

korea.c

```
#include <stdio.h>

int main()
{
    short no_univ = 276;
    int population = 48295000;
    long budget = 2370000000000000L;

    printf("korea info\n");
    printf("univ no : %d\n", no_univ);
    printf("population : %d\n", population);
    printf("budget : %d\n", budget);

    return 0;
}
```

Output :

```
korea info
univ no : 276
putpulation:   48295000
budget:   -590360576
```

Overflow?

- (integer type) overflow
  - occurs when storing a value that is bigger than what can be stored.
  - Ex) 2,147,483,647 (= $2^{31}-1$) + 1 = ?

```
  01111111 11111111 11111111 11111111
+ 00000000 00000000 00000000 00000001
-----------------------------------------
  10000000 00000000 00000000 00000000
```

```
#include <stdio.h>

int main()
{
        int a=2147483647;

        printf("%d,%d\n",a,a+1);
        return 0;
}
```

C Programming
Operators

Expressions and Statements

- Expression
  - Combination of constants,variables,operators, and function calls
  - Ex)

```
a+b
3.0*x - 9.66553
tan(angle)
```

- Statement
  - An expression terminated with a semicolon
  - Ex)

```
sum = x + y + z;
printf("Dragons!");
```

Assignment Operator

- The equal sign = is an assignment operator
- Used to give a variable the value of an expression
  - Ex)

```
x=34.8;
sum=a+b;
slope=tan(rise/run);
midinit='J';
j=j+3;
x=y=z=13.0;
```

- Initialization
  - Ex)

```
int i=0;
```

Arithmetic operators

- Binary operators
  - Addition : +
  - Subtraction : –
  - Multiplication : *
  - Division : /
  - Modulus : % // only works for integers values
- Unary operators
  - + , –
- Integer division
  - 1/2 = 0 (?) , 3/2 = 1 (?)

Arithmetic operators

- In binary operators
  - If two operands are int type : the result is int type
  - If one or two operands are floating-point type : the result is floating-point type
    - `2 + 3.14 ⇒ 2.0 + 3.14 = 5.14`
    - `12.0/5 ⇒ 12.0/5.0 = 2.4`

increment/decrement

- Increment operator ++
  - **`i=i+1;`**
    - **`i++;`** **`// postfix form`**
    - **`++i;`** **`// prefix form`**
- decrement operator –
  - i=i–1;
    - i––; // postfix form
    - ––I; // prefix form
- Difference between **`i++`** and **`++i`** ?

prefix vs. postfix

- Difference shows up when the operators are used as a part of a larger expression
  - **++k** : k is incremented before the expression is evaluated.
  - **k++** : k is incremented after the expression is evaluated.
- Ex) difference?

```
int a;
int i=0, j=0;
a= (++i) + (++j);
```

```
int b;
int i=0, j=0;
b= (i++) + (j++);
```

Shorthand Operators

- General syntax
  - *variable = variable op expression;*

    is equivalent to

    variable op= expression;

- Common forms
  - **+=, -=, *=, /=, %=**

- Examples

```
j=j*(3+x); j *= 3+x;

a=a/(s-5); a /= s-5;
```

Precedence , Associativity of Operators

- Operator Precedence
  - determines the order in which operations are performed
  - operators with higher precedence are employed first.

precedence	operators
1st	unary + , unary -
2nd	binary * / %
3rd	binary + -

- Operator Associativity
  - if two operators in an expression have the same precedence, associativity determines the direction in which the expression will be evaluated.

```
* , / , %   : L -> R
+ , - (bin): L -> R
=           : R -> L
+ , - (unary) : R -> L
```

Precedence Examples

- Evaluation Order

```
    1 + 2 * 3 - 4
->  1 + 6 - 4
->  7 - 4
->  3
```

  - use **parenthesis** to force a desired order of evaluation
  - Ex)

```
(1 + 2) * (3 - 4)
```

Associativity Examples

- Left associativity

  `a / b * c  →  (a / b) * c`

- Right associativity

  `- + - a  →  - (+ (- a))`

Bitwise Operators

shift/logic	Op. name	usage	type	output
shift op.	left shift	`a<<n`	integer	Shift bits of `a` to left by n bit Newly created bits will be 0
	right shift	`a>>n`	integer	Shift bits of `a` to right by n bit Newly created bits will be 0
bit op.	bit AND	`a & b`	integer	AND of `a`'s and `b`'s each bit
	bit OR	`a | b`	integer	OR of `a`'s and `b`'s each bit
	bit XOR	`a ^ b`	integer	XOR of `a`'s and `b`'s each bit
	1's complement	`~a`	integer	1's complement of `a`

Truth/False Table

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	~a
0	1
1	0

Bitwise Operators Examples

- 11 = 0000 0000 0000 1011
- 17 = 0000 0000 0001 0001

- 11 << 2
- 0000 0000 0000 1011 << 2 = 0000 0000 0010 1100 = 44

- 17 >> 3
- 0000 0000 0001 0001 >> 3 = 0000 0000 0000 0010 = 2

```c
#include <stdio.h>

int main() {
    int a = 11;
    int b = 17;

    printf("%d << 2 = %d \n", a, a << 2);
    printf("%d >> 3 = %d \n", b, b >> 3);

    return 0;
}
```

output:
11 << 2 = 44
17 >> 3 = 2

example

```c
#include <stdio.h>

int main() {
    short a = 0x1f05;
    short b = 0x31a1;

    printf("%x & %x = %x \n", a, b, a&b);
    printf("%x | %x = %x \n", a, b, a|b);
    printf("%x ^ %x = %x \n", a, b, a^b);
    printf("~%x  = %x \n", a, ~a);

    return 0;
}
```

output:

```
1f05 & 31a1 = 1101
1f05 | 31a1 = 3fa5
1f05 ^ 31a1 = 2ea4
~1f05  = ffffe0fa
```

example

expression	value	result
a	0x1f05	0001 1111 0000 0101
b	0x31a1	0011 0001 1010 0001
~a	0xe0fa	1110 0000 1111 1010
a & b	0x1101	0001 0001 0000 0001
a | b	0x3fa5	0011 1111 1010 0101
a ^ b	0x2ea4	0010 1110 1010 0100

Relational Operators

Meaning	Symbol	Data Type	Return Value
Equal	a == b	integer or floating point	1(=true) if a is equal to b otherwise 0(=false)
not equal	a != b	integer or floating point	1(=true) if a is not equal to b otherwise 0(=false)
less than	a < b	integer or floating point	1(=true) if a is less than b otherwise 0(=false)
less than or equal to	a <= b	integer or floating point	1(=true) if a is less than or equal to b otherwise 0(=false)
greater than	a > b	integer or floating point	1(=true) if a is greater than b otherwise 0(=false)
greater than or equal to	a >= b	integer or floating point	1(=true) if a is greater than or equal to b otherwise 0(=false)

example

```c
#include <stdio.h>

int main() {
    int x = 10;
    int y = 11;

    printf("(%d > %d)  = %d\n", x, y, x > y);
    printf("(%d >= %d) = %d\n", x, y, x >= y);
    printf("(%d == %d) = %d\n", x, y, x == y);
    printf("(%d != %d) = %d\n", x, y, x != y);
    printf("(%d < %d)  = %d\n", x, y, x < y);
    printf("(%d <= %d) = %d\n", x, y, x <= y);

    return 0;
}
```

output:

```
(10 > 11)  = 0
(10 >= 11) = 0
(10 == 11) = 0
(10 != 11) = 1
(10 < 11)  = 1
(10 <= 11) = 1
```

Logical Operators

op name	expression	meaing
logical NOT	! a	If a is false, then 1(=true), otherwise 0(=false)
logical AND	a && b	If both a and b are true, then 1(=true), otherwise 0(=false)
logical OR	a || b	If either a or b is true, then 1(=true), otherwise 0(=false)

example

```
#include <stdio.h>

int main()
{
    int score;

    printf("Score?");
    scanf("%d",&score);
    if (score >= 90 && score <=100)
          printf("your grade is A.\n");
    if (score >= 80 && score < 90)
          printf("your grade is B.\n");
    if (score >= 70 && score < 80)
          printf("your grade is C.\n");
    if (score >=60 && score < 70)
          printf("your grade is D.\n");
    if (score < 60)
          printf("your grade is F.\n");

    return 0;
}
```

Automatic Type Conversion

- What happens when expression has mixture of different data types.
- Ex)

```
double x=1.2;
float y=0.0;
int i=3;
int j=0;

j=x+i;   /* (temporary copy of)i will be converted to double type
            before '+' operation.
            the value of i in memory is unchanged */
y=x+i;

printf("j=%d , y=%f\n",j,y);
```

Automatic Type Conversion

- "lower" types are promoted to "higher" types. The expression itself will have the type of its highest operand. The **type hierarchy is as** follows
  - **`long double`**
  - **`double`**
  - **`float`**
  - **`int`**
  - **`short , char`**

- If either operand is `long double`, convert the other to `long double`
- Otherwise, if either operand is `double`, convert the other to `double`
- Otherwise, if either operand is `float`, convert the other to `float`
- Otherwise, convert `char` and `short` to `int`

Automatic Type Conversion with assignment operator

- Example

```
double x=5.5;
int y=3;

y=x;            /* x will be converted to int type */

x=y;            /* y will be converted to double type */
```

Type casting

- Programmers can enforce type conversion to a variable

Ex1)

```
double x=3.5;
double y=2.7;
double below_point;

below_point = x*y - (int)(x*y) ;
```

Ex2)

```
double x=3.5;
printf("integer number of x = %d\n",(int)x);
```

CSE-103

Structured Programing Language

As a new comer You can skip pages(In 1st CT only) :

Page NO: 10-13, 16-19, 25, 27 ☺

A Brief History of C

C is a general-purpose language which has been closely associated with the **UNIX** operating system for which it was developed - since the system and most of the programs that run it are written in C.

Many of the important ideas of C stem from the language **BCPL**, developed by Martin Richards. The influence of BCPL on C proceeded indirectly through the language **B**, which was written by Ken Thompson in 1970 at Bell Labs, for the first UNIX system on a **DEC** PDP-7. **BCPL** and **B** are "type less" languages whereas C provides a variety of data types.

In 1972 Dennis Ritchie at Bell Labs writes C and in 1978 the publication of The C Programming Language by Kernighan & Ritchie caused a revolution in the computing world.

In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or "ANSI C", was completed late 1988.

YOU CAN ANSWER THESE QUESTIONS FROM PREVIOUS PAGE:

- ✓ What is C ?
- ✓ What is UNIX?
- ✓ Why C was Developed?
- ✓ What is the difference between B and C ?
- ✓ When and where C was written?
- ✓ What is ANSI ?
- ✓ When ANSI C was completed?

- You will create and learn questions from every pages like this by yourself. ☺

Beginning with C programming:

Finding a Compiler:

Before we start C programming, we need to have a compiler to compile and run our programs.

Windows: There are many compilers available freely for compilation of C programs like Code Blocks and Dev-CPP. I strongly recommend Code Blocks.

Linux: For Linux, gcc comes bundled with the linux, Code Blocks can also be used with Linux.

Why use C?

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems - and efficient **compilers** are available for machines ranging in power from the **Apple** Macintosh to the **Cray** supercomputers.

The largest measure of C's success seems to be based on purely practical considerations:

- the portability of the compiler;
- the standard library concept;
- a powerful and varied repertoire of operators;
- an elegant syntax;
- ready access to the hardware when needed;
- and the ease with which applications can be optimized by hand-coding isolated procedures

C is often called a "Middle Level" programming language.

This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low level functions. Most high-level languages (e.g. Fortran) provides everything the programmer might want to do already built into the language. A low level language (e.g. **assembler**) provides nothing other than access to the machines basic instruction set. A middle level language, such as C, probably doesn't supply all the constructs found in high-languages - but it provides you with all the building blocks that you will need to produce the results you want!

Uses of C

C was initially used for system development work, in particular the programs that make-up the operating system. Why use C? Mainly because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

In recent years C has been used as a general-purpose language because of its popularity with programmers. It is not the world's easiest language to learn and you will certainly benefit if you are not learning C as your first programming language! ☺ C is trendy . many well established programmers are switching to C for all sorts of reasons, but mainly because of the portability that writing standard C programs can offer.

Running C Programs

Objectives

Having read this section you should be able to:

- Edit, link and run your C programs

This section is primarily aimed at the beginner who as no or little experience of using compiled languages. We cover the various stages of program development. The basic principles of this section will apply to what ever C compiler you choose to use, the stages
are *nearly* always the same

The Edit-Compile-Link-Execute Process

Developing a program in a compiled language such as C requires at least four steps:

- **editing** (or writing) the program
- **compiling** it
- **linking** it
- **executing** it

We will now cover each step separately.

Editing

You write a computer program with words and symbols that are understandable to human beings. This is the *editing* part of the development cycle. You type the program directly into a window on the screen and save the resulting text as a separate file. This is often referred to as the *source file* (you can read it with the **TYPE** command in **DOS** or the **cat** command in **unix**). The custom is that the text of a C program is stored in a file with the extension .c for C programming language

Compiling

You cannot directly execute the source file. To run on any computer system, the source file must be translated into binary numbers understandable to the computer's Central Processing Unit (for example, the 80*87 microprocessor). This process produces an intermediate object file - with the extension .obj, the .obj stands for Object.

Linking

The first question that comes to most peoples minds is *Why is linking necessary?* The main reason is that many compiled languages come with library routines which can be added to your program. Theses routines are written by the manufacturer of the compiler to perform a variety of tasks, from input/output to complicated mathematical functions. In the case of C the standard input and output functions are contained in a library (stdio.h) so even the most basic program will require a library function. After linking the file extension is .exe which are executable files.

Executable files

Thus the text editor produces .c source files, which go to the compiler, which produces .obj object files, which go to the linker, which produces .exe executable file. You can then run .exe files as you can other applications, simply by typing their names at the **DOS** prompt or run using windows menu.

Structure of C Programs

Objectives

Having completed this section you should know about:

- C's character set
- C's keywords
- the general structure of a C program
- that all C statement must end in a ;
- that C is a free format language
- all C programs us header files that contain standard library functions.

C's Character Set

C does not use, nor requires the use of, every character found on a modern computer keyboard.

The only characters required by the C Programming Language are as follows:

- **A - Z**
- **a -z**
- **0 - 9**
- **space . , : ; ' $ "**
- **# % & ! _ {} [] () $$$$ &&&& |**
- **+ - / * =**

The use of most of this set of characters will be discussed throughout the course.

The form of a C Program

All C programs will consist of at least one function, but it is usual (when your experience grows) to write a C program that comprises several functions. The only function that has to be present is the function called **main**. For more advanced programs the **main** function will act as a controlling function calling other functions in their turn to do the dirty work! The **main** function is the first function that is called when your program executes.

C makes use of only 32 **keywords** which combine with the formal syntax to the form the C programming language. Note that all keywords are written in lower case - C, like UNIX, uses upper and lowercase text to mean different things. If you are not sure what to use then always use lowercase text in writing your C programs. A keyword may not be used for any other purposes. For example, you cannot have a variable called **auto**.

The layout of C Programs

The general form of a C program is as follows (don't worry about what everything means at the moment - things will be explained later):

```
pre-processor directives
global declarations
main()
{
local variables to function main ;
statements associated with function main ;
}
f1()
{
local variables to function 1 ;
statements associated with function 1 ;
}
f2()
{
local variables to function f2 ;
```

statements associated with function 2 ;
}
.
.
.
etc
Note the use of the bracket set () and {}. () are used in conjunction with function names whereas {} are used as to delimit the C statements that are associated with that function. Also note the semicolon - yes it is there, but you might have missed it! a semicolon (;) is used to terminate C statements. C is a free format language and long statements can be continued, without truncation, onto the next line. The semicolon informs the C compiler that the end of the statement has been reached. Free format also means that you can add as many spaces as you like to improve the look of your programs.
A very common mistake made by everyone, who is new to the C programming language, is to miss off the semicolon. The C compiler will concatenate the various lines of the program together and then tries to understand them - which it will not be able to do. The error message produced by the compiler will relate to a line of you program which could be some distance from the initial mistake.

Pre-processor Directives

C is a small language but provides the programmer with all the tools to be able to write powerful programs. Some people don't like C because it is too primitive! Look again at the set of **keywords** that comprises the C language and see if you can find a command that allows you to print to the computer's screen the result of, say, a simple calculation. Don't look too hard because it doesn't exist.

It would be very tedious, for all of us, if every time we wanted to communicate with the computer we all had to write our own output functions. Fortunately, we do not have to. C uses libraries of standard functions which are included when we build our programs. For the novice C programmer one of the many questions always asked *is does a function already exist for what I want to do?* Only experience will help here but we do include a function listing as part of this course.

All programs you will write will need to communicate to the outside world - I don't think I can think of a program that doesn't need to tell someone an answer. So all our C programs will need at least one of C's standard libraries which deals with standard inputting and outputting of data. This library is called **stdio.h** and it is declared in our programs before the **main** function. The .h extension indicates that this is a header file.

MY First C Program

```
#include<stdio.h>
int main()
{
   printf(“ My name is ABdur Razzak ”);
    return 0;
}
```

Output:

My name is ABdur Razzak

The program is a short one, to say the least. Here it is:

```
#include <stdio.h>
int main()
{
printf("Hello World\n");
}
```

The first line is the standard start for all C programs - **main()**. After this comes the program's only instruction enclosed in curly brackets **{}**. The curly brackets mark the start and end of the list of instructions that make up the program - in this case just one instruction.

Notice the semicolon marking the end of the instruction. You might as well get into the habit of ending every C instruction with a semicolon - it will save you a lot of trouble! Also notice that the semicolon marks the end of an instruction - it isn't a separator as is the custom in other languages.

If you're puzzled about why the curly brackets are on separate lines I'd better tell you that it's just a layout convention to help you spot matching brackets

C is very unfussy about the way you lay it out. For example, you could enter the **Hello World** program as:

main(){printf("Hello World\n");}

but this is unusual.

The **printf** function does what its name suggest it does: it prints, on the screen, whatever you tell it to. The **"\n"** is a special symbols that forces a new line on the screen.

OK, that's enough explanation of our first program! Type it in and save it as **Hello.c**. Then use the compiler to compile it, then the linker to link it and finally run it. The output is as follows:

Hello World

Add Comments to a Program

A **comment** is a note to yourself (or others) that you put into your source code. All comments are ignored by the compiler. They exist solely for your benefit. Comments are used primarily to document the meaning and purpose of your source code, so that you can remember later how it functions and how to use it. You can also use a comment to temporarily remove a line of code. Simply surround the line(s) with the comment symbols.

In C, the start of a comment is signaled by the **/*** character pair. A comment is ended by ***/**. For example, this is a syntactically correct C comment:

```
/* This is a comment. */
```

Comments can extend over several lines and can go anywhere except in the middle of any C keyword, function name or variable name. In C you can't have one comment within another comment. That is comments may not be nested. Lets now look at our first program one last time but this time with comments:

```
main() /* main function heading */
{
printf("\n Hello, World! \n"); /* Display message on */
} /* the screen */
```

This program is not large enough to warrant comment statements but the principle is still the same.

Data Types

Objectives

Having read this section you should be able to:

- declare (name) a local variable as being one of C's five data types
- initialize local variables
- perform simple arithmetic using local variables

Now we have to start looking into the details of the C language. How easy you find the rest of this section will depend on whether you have ever programmed before - no matter what the language was. There are a great many ideas common to programming in any language and C is no exception to this rule.

So if you haven't programmed before, you need to take the rest of this section slowly and keep going over it until it makes sense. If, on the other hand, you have programmed before you'll be wondering what all the fuss is about It's a lot like being able to ride a bike!

The first thing you need to know is that you can create *variables* to store *values* in. A variable is just a named area of storage that can hold a single value (numeric or character). C is very fussy about how you create variables and what you store in them. It demands that you declare the name of each variable that you are going to use and its *type*, or *class*, before you actually try to do anything with it.

Five Data Types:

- Integer
- Floating
- Double
- Character
- Void

- These data types are used to declare the variable names

For Example :

int S;

- Is a C programming statement to declare a variable named “S”. The type of S is integer .
- Some integer numbers are- 0, 1,5,6,2,7,2,89,……. .
- They are free from decimal point.

float S;

- Is a C programming statement to declare a variable named “S”. The type of S is floating .
- Some floating numbers are- 0.0, 1.5, 6.2, 2.89,……. .

Double:

double S;

- Is a C programming statement to declare a variable named “S”. The type of S double.
- Double is actually the double(twice) precision of floating.
- Some double numbers are – 0.000000, 1.200000, ….. Etc.

Character:

char S;

- Is a C programming statement to declare a variable named “S”. The type of S is character .
- Any Unique symbols are called a character. For example – a, 1, #, +, $ all of these are character.
- There are 256 unique characters.

Rules Of variable name:

You can not give the name of your variable as you wish in C. There are some rules for it.

- Name must begin with character or under score. EX- a, ab, _ab are unique variables.
- Can not begin with digit. EX- 1a, 3b is not valid.
- Variable name can not be any of C's 32 keywords. Ex: int, while, are some of C's keyword.
- Every variable must have unique name or unique combination of character. Ex: abc, acb, bcb, are unique variables.
- Variable name cannot be any of defined function in C. EX: scanf, printf cannot be a variable name.

Write a C program to find the SUM of two numbers:

```
#include<stdio.h>
Int main()
{
   int a, b, c;
   a=5, b= 10;
   c= a+b;
   printf(“%d”, c);
   return 0;
}
```

OUTPUT:

15

We can write the program in another way:

```
#include<stdio.h>
Int main()
{
   int a, b, c;
   scanf(“%d %d”, &a, &b);
   c= a+b;
   printf(“%d”, c);
   return 0;
}
```

OUTPUT:

“Depends on your Input in console”

Scanf() Is input function where Printf() is Output function

Structure of scanf:

- scanf(: scanf with brackets starts. (
- Scanf(“” : scanf goes with double quoations. “ ”
- Scanf(“%d”: scanf goes with format specifier. %d
- Scanf (“ %d ” , : scanf goes with comma aftar double quoation. ,
- Scanf (“%d” , & : scanf goes with reference operator. &.
- Scanf (“ %d”, &a : scanf goes with reference to a variable. a.
- Scanf(“%d”, &a) : scanf ends with the bracket close.

Format Specifier:

%d = integer (d= Decimal)

%f= Floating

%lf = Double

%c= Character.

Operators in C:

Five types of operators are in C:

Arithmatic operator: +, -, *, /, % etc.

Relational operator: ==, !=, <, > etc.

Logical Operator: &&, ||, ! Etc.

Bitwise Operator : &, |, ~ etc.

Assignment operator: =, +=, -= etc.

Try yourself:

- Write a c program to calculate sum of last two digit of your ID.
- Write a c program to print your father name as output.
- Write a c program to print the sum of last 3 digit of your mobile no.
- Write a c program multiply the last two digit of your mobile no.
- Write a c program to divide the last two digit of your mobile no.
- Write a c program to add two number.
- Write a c program to subtract two number.
- Write a c program to divide a number by another.
- Write a c program to print the distance of UGV from your home.
- Write a c program to print your birth date . (only date not month or year)
- Write a c program to print your expected CGPA.
- Which format specifier do you prefer why?
- Write an invalid variable name using last 3 digit of your ID.
- What is the functionality of %, +=, *=, &, ! , =, == operators in C?
- And So on …….. Practice ……. Practice ……. Practice. ☺

Any query feel free to contact. ☺

Prepared By
MD. Abdur Razzak
Lecturer, Department of CSE
UGV, Barisal.