1
Basic C Programming

This Lecture

- At the end of this lecture, you will be able to understand the following:
  - Why Learning C Programming Language?
  - Development of a C Program
  - Data Types, Constants, Variables and Operators
  - Simple Input/Output

Basic C Programming

- **Why Learning C Programming Language?**
- Development of a C Program
- Data Types, Constants, Variables and Operators
- Simple Input/Output

Why Learning C Programming Language?

- **Advantages** on using C
  - Powerful, flexible, efficient, portable
  - Enable the creation of well-structured programs
- Any **disadvantages**?
  - Free style and **not strongly-typed**
  - The use of ***pointers***, which may confuse many students
- **Why** doing data structures in C
  - Efficient
  - Provide ***pointers*** for building data structures which are powerful
  - Bridge to C++ (OO Programming)

Basic C Programming

- Why Learning C Programming Language?
- **Development of a C Program**
- Data Types, Constants, Variables and Operators
- Simple Input/Output

Structure of a C Program

A simple C program has the following structure:

```
/* comment line 1 */
//  or comment line 2
preprocessor instructions
int main()
{
   statements;
   return 0;
}
```

e.g. # include <stdio.h>

global declaration

if no return, use void

Structure of a C Program

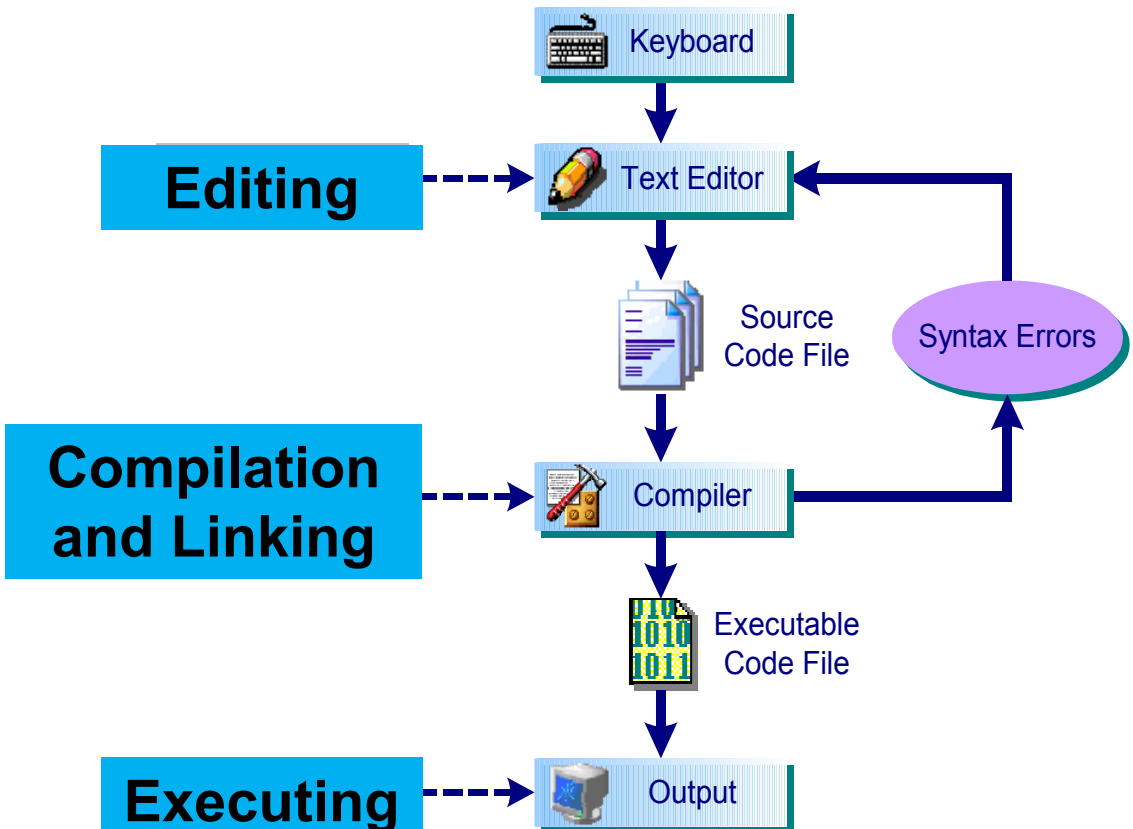
A simple C program has the following structure:

```
/* comment line 1 */
//  or comment line 2
preprocessor instructions
int main()
{
   statements;
   return 0;
}
```

An Example Program

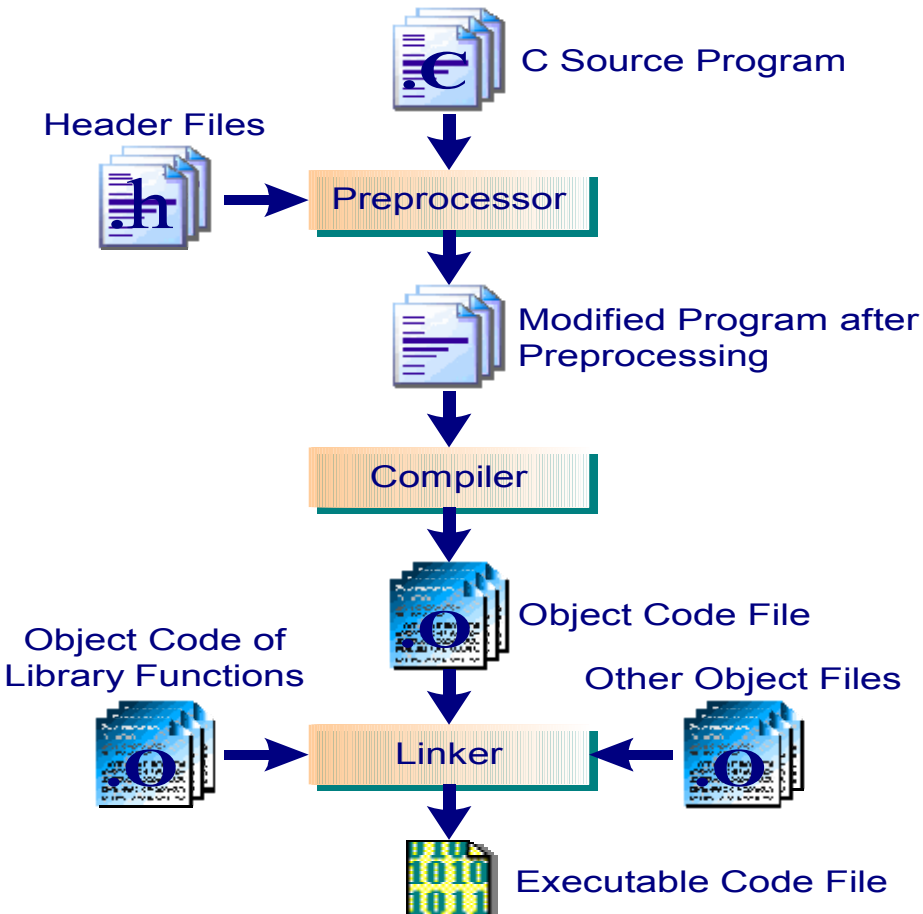
```
 /* Purpose: a program to
print Hello World! */
#include <stdio.h>
int main()
{   // begin body
    printf("Hello World! \n");
    return 0;
}   // end body
```

Development of a C Program



Compilation and Linking

After writing the C program and typing it into the editor, the program needs to be processed by ***(1)preprocessor*** ***(2)compiler*** ***(3)linker*** before you can execute the program.



Integrated Development Environment (IDE)

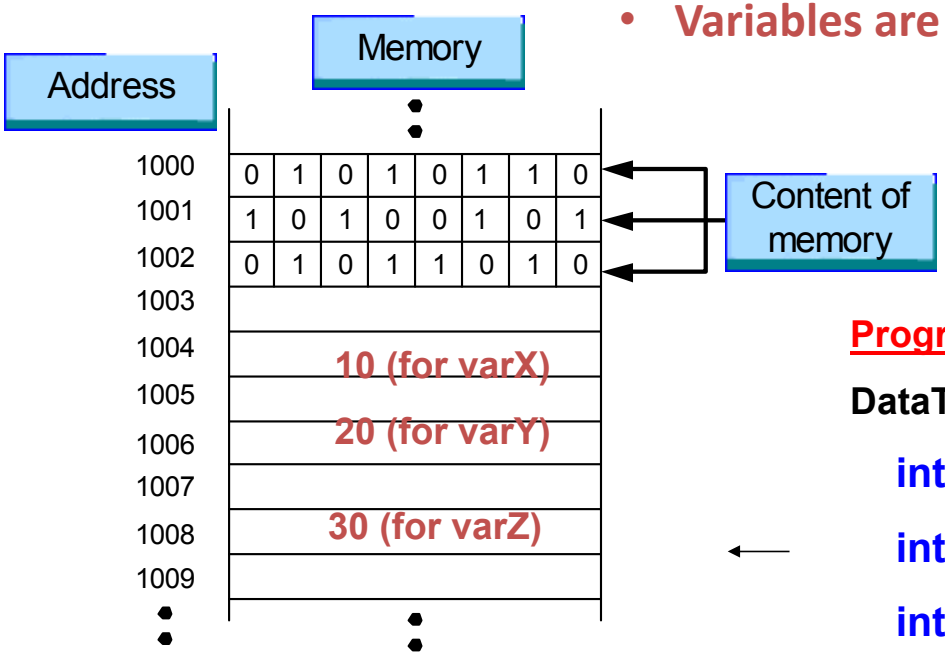
- **Code::Blocks**
- **Microsoft Visual Studio C/C++** (Dreamspark for Student, https://www.dreamspark.com/)
- **JGRASP** (http://www.jgrasp.org/)
- **Bloodshed Dev-C++** (http://www.bloodshed.net/devcpp.html/)

Basic C Programming

- Why Learning C Programming Language?
- Development of a C Program
- **Data Types, Constants, Variables and Operators**
- Simple Input/Output

Computer Memory and Variables

- Computer memory is used for storing data objects (variables or constants).
- Variables are used for storing data.



Program logic

DataType	Variable	Value
int	varX	= 10;
int	varY	= 20;
int	varZ;	

```
varZ = varX + varY;
```

Data and Types

- It determines the **kind of data** that a **variable** can hold, how many **memory cells** (bytes) are reserved for it and the operations that can be performed on it. (Note – the size in memory depends on machines.)

- **Integers**
  - short (2 bytes – 16 bits)
  - **int** (4 bytes or 2 bytes in some older systems)
  - long (4 bytes)
  - unsigned (4 bytes)
  - unsigned short (2 bytes)
  - unsigned long (4 bytes)

- **Floating Points**
  - **float** (4 byte – 32 bits)
  - double (8 bytes – 64 bits)
  - long double

- **Characters**
  - 128 distinct characters in the ASCII character set.
  - Two C character types: **char** and unsigned char.
    - char (1 byte – 8 bits, range: -128 – 127)
    - unsigned char (1 byte – 8 bits, range: 0 – 255)

Note: Operations involving the **int** data type are always *exact*, while the **float** and **double** data types can be *inexact*.

Character - ASCII Set (1 byte)

	0	1	2	3	4	5	6	7	8	9
0	NUL							BEL	BS	TAB
1	LF		FF	CR						
2								ESC		
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Examples of Escape Sequence

- Some useful **non-printable control characters** are referred to by the ***escape sequence*** which is a better alternative, in terms of memorization, than numbers. e.g. **'\n'** the newline (or linefeed) character instead of the number **10**.

'\a'	**alarm bell**	'\f'	**form feed**	**'\n'**	**newline**
'\t'	**horizontal tab**	'\"'	**double quote**	'\v'	**vertical tab**
'\b'	**back space**	'\\'	**backslash**	'\r'	**carriage return**
'\''	**single quote**				

Constants

- A constant is an object whose value is **unchanged** throughout the life of the program. There are four types of constants: integer constants, floating point constants, character constants and string constants.
- Four types of constant values:
  - **Integer**: e.g. 100, –256; **Floating-point**: e.g. 2.4, –3.0;
  - **Character**: e.g. 'a', '+' ; **String**: e.g. "Hello Students "

or 3124e3

- **Defining Constants – by using the preprocessor directive #define**

Format: **#define** CONSTANTNAME **value**

E.g. **#define TAX_RATE** 0.12

/* define a constant TAX_RATE with 0.12 */

- **Defining Constants - By defining a constant variable**

Format: **const type varName = value;**

E.g. **const** float **pi = 3.14159**;

/* declare a float constant variable pi with value 3.14159 */

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

name with lowercase

Variables

- A variable declaration always contains 2 components:
  - its **data_type** (eg. short, int, long, etc.)
  - its **var_name** (e.g. count, numOfSeats, etc.)

  The syntax for variable declaration: **data_type var_name**[, **var_name**];
- Declare your variables at the **beginning** of your program. Examples of variable initializations:

  int count = 20;

  float temperature, **result**;
- The following C **keywords** are **reserved** and **cannot** be used as variable names:

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
struct	switch	typedef	union	sizeof	static
volatile	while	unsigned	void		

Operators

- Fundamental Arithmetic operators: +, -, *, /, %
  - E.g. 7/3 = 2; 7%3 = 1; 6.6/2.0=3.3;
- Assignment operators:
  - E.g. float amount = 25.50;
- Arithmetic assignment operators: +=, -=, *=, /=,%=
  - E.g. a += 5;
- Chained assignment:
  - E.g. a = b = c = 3;
- Increment/decrement operators: ++, --
- Relational operators: ==, !=, <, <=, >, >=
  - E.g. 7 >= 5

[to be discussed in the next lecture]

uppercase → lowercase
+32

Increment Operators

- In **prefix mode**: **++**var_name.

  (1) var_name is incremented by 1 and
  (2) the value of the expression is the updated value of var_name.

- In **postfix mode**: var_name**++**.

  (1) The value of the expression is the current value of var_name
  (2) then var_name is incremented by 1.

```
#include <stdio.h>
int main()
{     int    num = 4;
      printf("value of num is %d\n", num);
      num++;      // ++num; i.e., num = num+1;
      printf("value of num is %d\n", num);
      num = 4;
      printf("value of num++ is %d\n", num++);
      printf("value of num is %d\n",num);
      printf("value of ++num is %d\n", ++num);
      printf("value of num is %d\n\n",num);
      return 0;
}
```

Output

```
value of num is 4
value of num is 5

value of num++ is 4
value of num is 5
value of ++num is 6
value of num is 6
```

Decrement Operators

- The way the **decrement operator** '--' works in the same way as the ++ operator, except that the variable is decremented by 1.

```
#include <stdio.h>
int main()
{
      int    num = 4;
      printf("value of num is %d\n", num);
      num--;   // same as --num;
      printf("value of num is %d\n", num);
      num = 4;
      printf("value of num-- is %d\n", num--);
      printf("value of num is %d\n", num);
      printf("value of --num is %d\n", --num);
      printf("value of num is %d\n", num);
      return 0;
}
```

Output

```
value of num is 4
value of num is 3

value of num-- is 4
value of num is 3
value of --num is 2
value of num is 2
```

Data Type Conversion

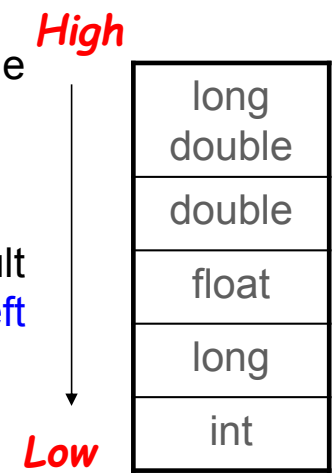
Arithmetic operations require **two numbers** in an expression/assignment are of the **same type**.

If not, three kinds of conversion are available:

1. ***Explicit conversion*** - uses type casting operators, i.e. (int), (float), ..., etc.
   - e.g. (int)2.7 + (int)3.5

2. ***Arithmetic conversion*** - in mix operation, it converts the operands to the type of the **higher ranking** of the two.
   - e.g. double a; a = **2**+ 3.5; // 2 to 2.0 then add

3. ***Assignment conversion*** - converts the type of the result of computing the expression to that of the type of the left hand side if they are different.
   - e.g. int b; b = **2.7 + 3.5**; // 6.2 to 6 then to b



Note: Possible *pit-falls* about type conversion -
Loss of precision: e.g. from **float** to **int**, the fractional part will be lost.

Mathematical Libraries

#include <math.h>

Function	Argument Type	Description	Result Type
`ceil(x)`	`double`	Return the smallest `double` larger than or equal to `x` that can be represented as an `int`.	`double`
`floor(x)`	`double`	Return the largest `double` smaller than or equal to `x` that can be represented as an `int`.	`double`
`abs(x)`	`int`	**Return the absolute value of `x`, where `x` is an `int`.**	`int`
`fabs(x)`	`double`	Return the absolute value of `x`, where `x` is a floating point number.	`double`
`sqrt(x)`	`double`	**Return the square root of `x`, where `x` >= 0.**	`double`
`pow(x,y)`	`double x, double y`	**Return x to the y power, x^y.**	`double`
`cos(x)`	`double`	Return the cosine of `x`, where `x` is in radians.	`double`
`sin(x)`	`double`	Return the sine of `x`, where `x` is in radians.	`double`
`tan(x)`	`double`	Return the tangent of `x`, where `x` is in radians.	`double`
`exp(x)`	`double`	Return the exponential of `x` with the base e, where e is 2.718282.	`double`
`log(x)`	`double`	Return the natural logarithm of `x`.	`double`
`log10(x)`	`double`	Return the base 10 logarithm of `x`.	`double`

Basic C Programming

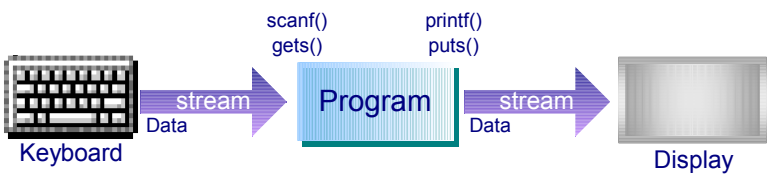
- Why Learning C Programming Language?
- Development of a C Program
- Data Types, Constants, Variables and Operators
- **Simple Input/Output**

Simple Input/Output

- The following two Input/Output functions are most frequently used:

  - **printf()** : output function
  - **scanf()** : input function

putchar ( )
getchar ( )



- The I/O functions are in the C library **<stdio>**, to use the I/O functions, we need to include the header file:

  #include <stdio.h>

  as the **preprocessor instruction** in a program.

printf(): Control-string

The printf() statement has the form:

printf (control-string, argument-list);

```
#include <stdio.h>
int main()
{
    int num1 = 1, num2 = 2;
    printf("%d + %d = %d\n",                    );
    return 0;
}
```

Output

1 + 2 = 3

- The **control-string** is a string constant. It is printed on the screen.
  - %d is a **conversion specification**. An item will be substituted for it in the printed output.

printf(): Argument-list

The printf() statement has the form:

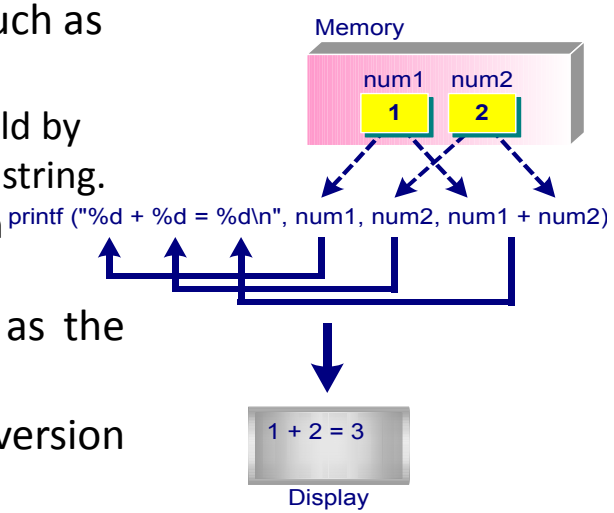
printf (control-string, argument-list);

```
#include <stdio.h>
int main()
{
    int num1 = 1, num2 = 2;
    printf("%d + %d = %d\n", num1, num2, num1+num2);
    return 0;
}
```

Output

1 + 2 = 3

- The **argument-list** contains a list of items such as item1, item2, …, etc.
  - Values are to be substituted into places held by the conversion specification in the control string.
  - An item can be a constant, a variable or an expression like num1 + num2.
- The **number** of items must be the same as the number of conversion specifiers.
- The **type** of items must also match the conversion specifiers.



Control-String
Conversion Specification

- A **conversion specification** is of the form

% [flag] [minimumFieldWidth] [.precision]conversionSpecifier

  - ***%*** and ***conversionSpecifier*** are compulsory. The others are optional.

Note:

- We will focus on using the compulsory options **%** and **conversionSpecifier**.
- Please refer to your textbook for the other options such as *flag*, *minimumFieldWidth* and *precision*.

printf() – Conversion Specifier

Some common types of *Conversion Specifiers*:

d	signed decimal conversion of int
o	unsigned octal conversion of unsigned
x,X	unsigned hexadecimal conversion of unsigned
c	single character conversion
f	signed decimal floating point conversion
s	string conversion

printf(): Example

```
#include <stdio.h>
int main( )
{
    int             num = 10;
    float           i = 10.3;
    double          j = 100.3456;

    printf(“int num = %d\n”, num);
    printf("float i = %f\n", i);
    printf("double j = %f\n", j);
      /* by default, 6 digits are printed
            after the decimal point */
    printf("double j = %.2f\n", j);
    printf("double j = %10.2f\n", j);
      /* formatted output */
    return 0;
}
```

Output

```
int num = 10
float i = 10.300000
double j = 100.345600


double j = 100.35
double j =       100.35
```

Simple Input: scanf()

- A scanf() statement has the form

  scanf (control-string, argument-list);

- ***control-string*** is a string constant containing conversion specifications.
- The **argument-list** contains a list of items.
  - The **items** in scanf() may be any variable matching the type given by the conversion specification. It cannot be a constant. It cannot be an expression like n1 + n2.
  - The **variable name** has to be preceded by an **&**. This is to tell scanf() the **address** of the variable so that scanf() can read the input value and store it in the variable.
- scanf stops reading when it has read all the items as indicated by the control string or the **EOF** (end of file) is encountered.

scanf(): Example

- A scanf() statement has the form

  scanf (control-string, argument-list);

```
#include <stdio.h>
int main( )
{
      int    n1, n2;
      float f1;
      double f2;

      printf(“Please enter 2 integers:\n”);
      scanf("%d %d", &n1, &n2);
      printf(“The sum = %d\n”, n1+n2);
      printf(“Please enter 2 floats:\n”);
      scanf("%f %lf", &f1, &f2);
      // Note: use %lf for double data
      printf(“The sum = %f\n”, f1+f2);
      return 0;
}
```


Output

Please enter 2 integers:
5 10
The sum = 15
Please enter 2 floats:
5.3 10.5
The sum = 15.800000

Common Error 1: scanf()

```
#include <stdio.h>
 int main( )
{
    int   n1, n2;
    float f1;
    double f2;

    printf("Please enter 2 integers:\n");
    scanf("%d %d", n1, n2);
    printf("The sum = %d\n", n1+n2);
    printf("Please enter 2 floats:\n");
    scanf("%f %lf", f1, f2);
    // Note: use %lf for double data
    printf("The sum = %f\n", f1+f2);
    return 0;
}
```

Can you compile the program? ✓

Can you run the program? ✗

Common Error 2: scanf()

Intended Output:
Please enter a number: ***1234<Enter>***
The number read is 1234
Correct (y/n)? ***y***
your reply : y

```
#include <stdio.h>
int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("your reply : %c\n", reply);   // display the char
      return 0;
}
```

Can you compile the program?
Can you run the program as intended?

Common Error 2: Problem

```
#include <stdio.h>
int main( )
{
        int number;
        char reply;
        printf("Enter a number: ");
        scanf("%d", &number); //read in an integer
        printf("The number read is %d\n", number);

        printf("Correct (y/n)? ");
        scanf("%c", &reply);    //read in a char

        printf("your reply : %c\n", reply);  //display the char
        return 0;
}
```

When the program runs:

Output

Enter a number: ***1234<Enter>***

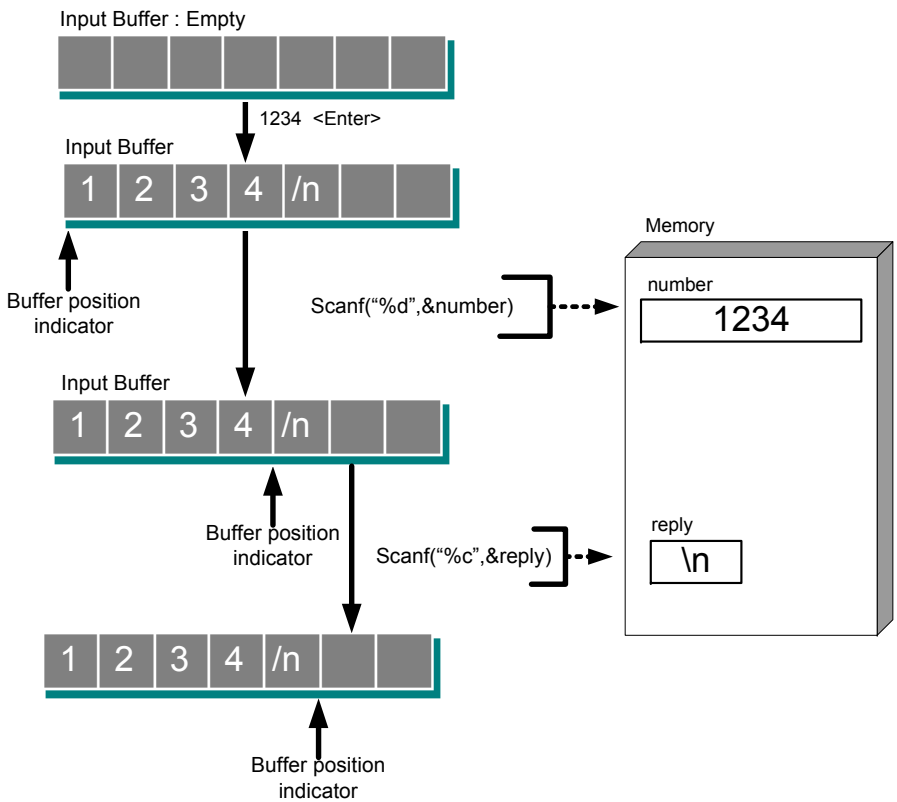
The number read is 1234

Correct (y/n)? your reply :

// an error here

// the reply is not read

Common Error 2: Reason



***Reason*:**

There is a **hidden character** '\n' entered when you type 1234 <Enter>

Common Error 2: Suggested Solutions

- **Solution 1: read in '\n'**

```
        . . .
printf("Correct (y/n)?");
scanf("\n%c", &reply);          // read the newline
printf("Your reply: %c\n", reply);
        . . .
```

OR
char dummy;
…
scanf("%c", &dummy);

- **Solution 2: using fflush()**

```
int number; char reply;
printf("Enter a number: ");
scanf("%d", &number);                  //read in an integer
printf("The number read is %d\n", number);
fflush(stdin);        // flush the input buffer with newline
printf("Correct (y/n)?");
scanf("%c", &reply);
printf("Your reply: %c\n", reply);
```

Common Error 2: Suggested Solutions

- **Solution 1: read in '\n'**

```
. . .
printf("Correct (y/n)?");
scanf("\n%c", &reply);          // read the newline
printf("Your reply: %c\n", reply);
. . .
```

OR

```
char dummy;
…
scanf("%c", &dummy);
```

- **Solution 2: using fflush()**

```
int number; char reply;
printf("Enter a number: ");
scanf("%d", &number);                  //read in an integer
printf("The number read is %d\n", number);
fflush(stdin);          // flush the input buffer with newline
printf("Correct (y/n)?");
scanf("%c", &reply);
printf("Your reply: %c\n", reply);
```

Character Input/Output

putchar( )

- The syntax of calling putchar is

```
putchar(characterConstantOrVariable);
```

It is equivalent to

```
printf("%c", characterConstantOrVariable);
```

- The difference is that putchar is **faster** because printf needs to process the control string for formatting. Also, it returns either the integer value of the written character or EOF if an error occurs.

getchar ( )

- The syntax of calling getchar is

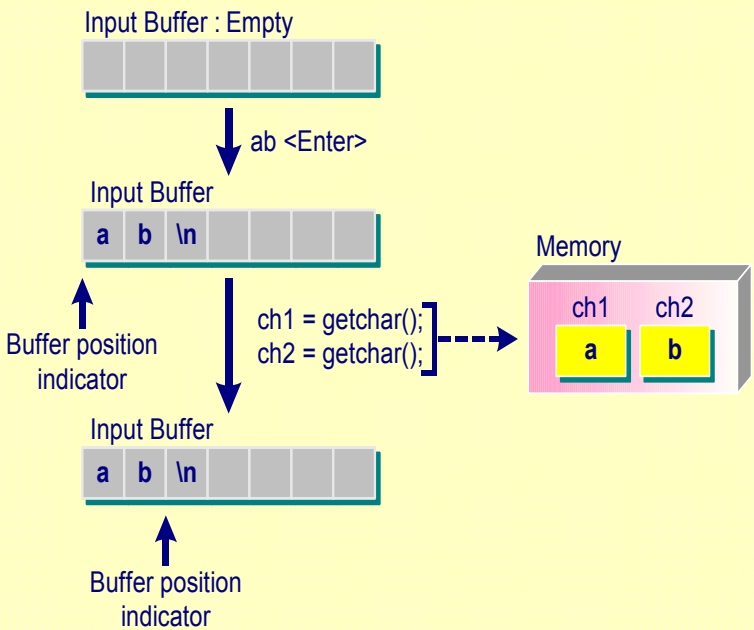
```
ch = getchar(); // ch is a character variable.
```

It is equivalent to

```
scanf("%c", &ch);
```

Character Input/Output

```
/* example to use getchar() and putchar()  */
 #include <stdio.h>
 int main(void)
{
      char ch, ch1, ch2;
      putchar('1');
      putchar(ch='a');
      putchar('\n');
      printf("%c%c\n", 49, ch);
      ch1 = getchar();
      ch2 = getchar();
      putchar(ch1);
      putchar(ch2);
      putchar('\n');
      return 0;
}
```



Output

1a

1a

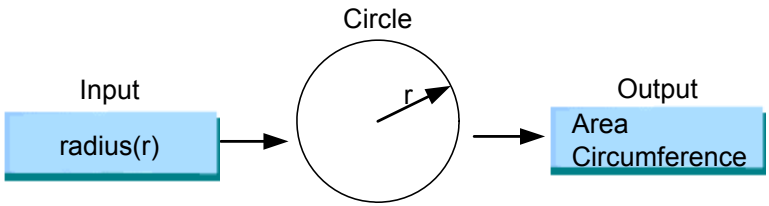
ab *(User Input)*

ab

Programming Problem

Problem: Writing a Simple C Program
(Sequential Structure)

```
/* Purpose: A sample program to calculate the area and circumference.
   Author: S.C. Hui */
```



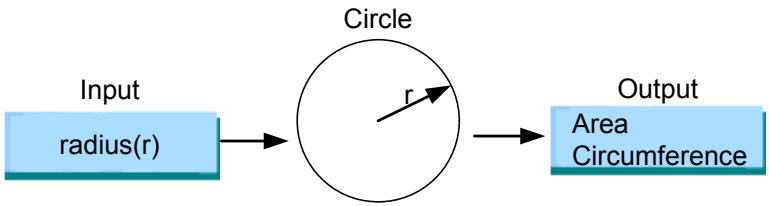
Area = π*r*r
Cirumference= 2*π*r

```
#include <stdio.h>
#define PI 3.14
int main( )
{   // declare variables
    float radius, area, circumference;
    // Read the radius of the circle
→     /* Write your code here */
    // Calculate the area
→     /* Write your code here */
    // Calculate the circumference
→     /* Write your code here */
    // Print the area and circumference of the circle
→     /* Write your code here */
    return 0;
}
```

Output
Enter the radius: 5.0
The area is 78.50
The circumference is 31.40

Writing a Simple C Program (Sequential Structure)

```
/* Purpose: A sample program to calculate the area and circumference.
   Author: S.C. Hui*/
#include <stdio.h>
#define PI 3.14
int main( )
{  // declare variables
   float radius, area, circumference;
   // Read the radius of the circle
   printf("Enter the radius: ");
   scanf("%f", &radius);
   // Calculate the area
   area = PI * radius * radius;
   // Calculate the circumference
   circumference = 2 * PI * radius;
   // Print the area and circumference of the circle
   printf("The area is %.2f\n", area);
   printf("The circumference is %.2f", circumference);
   return 0;
}
```



Area = π*r*r

Cirumference= 2*π*r

Output

Enter the radius: 5.0

The area is 78.50

The circumference is 31.40