Tutorial 1
Basic C and Control Flow - Q1

State the data type of each of the following:

a.	**'1'**	g.	**1870943465324L**
b.	**23**	h.	**1.234F**
c.	**0.0**	i.	**-564**
d.	**'\040'**	j.	**0177**
e.	**0x92**	k.	**0XfF4**
f.	**'\a'**	l.	**0xaaBB76L**

Q1: Suggested Answer

a.	**'1'**	: character
b.	**23**	: decimal integer
c.	**0.0**	: floating-point
d.	**'\040'**	: ASCII code in octal for the space, i.e. ' '
e.	**0x92**	: hexadecimal integer
f.	**'\a'**	: character (output: alarm escape sequence)
g.	**1870943465324L**	: decimal long integer
h.	**1.234F**	: floating-point
i.	**-564**	: negative decimal integer
j.	**0177**	: octal integer, starts with '0'
k.	**0XfF4**	: hexadecimal integer
l.	**0xaaBB76L**	: hexadecimal long integer

Q1: Suggested Answer

1. About 1(d) '\040': why and how should one use such a character constant?
2. Not all ASCII characters (see an ASCII table) can be keyed in from the keyboard like 'A', 'B', ..., 'Z', 'a', 'b', ..., 'z', '0', '1', '2', ..., '9', '!', '@', .., For example, the FF, Form Feed character (see an ASCII table) is an invisible control character.
3. Fortunately, the escape sequence of '\ooo', where o represents one octal digit, can be used to assign the FF character to a character variable like

```
char ch
ch = '\014';
```

Other ways in doing the same assignment are

```
ch = '\xC';      /* using hexadecimal character */
ch = 12;         /* using decimal integer */
ch = 014;        /* using octal integer */
ch = 0xC;        /* using hexadecimal integer */
```

ASCII Table

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	lf	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	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		

Basic C and Control Flow - Q2

(a) What will the following program output? (refer to an ASCII table)

(b) What will happen if the format specifier of the second printf is changed to **%d**?

(c) What will be the result if **0x** in the third printf is removed?

(d) What if the first **0** in the fourth printf is deleted?

```
#include <stdio.h>
int main()
{
    printf("%c", 'A');
    printf("%c", 65);
    printf("%c", 0x41);
    printf("%c", 0101);
    return 0;
}
```

Q2: Suggested Answer

(a) The output will be: **A A A A**

The constants in all the printf are the various forms of the alphabet 'A' (the character itself, the decimal form, the hexadecimal & the octal forms of the ASCII code for 'A' respectively.)

(b) If the format specifier of the second printf is changed to **%d**, the output will be: **A 65 A A**

(c) if **0x** in the third printf is removed, the '**)**' will be printed instead, because 41 (decimal) is the ASCII code for the left parenthesis.

(d) If the **0** of the fourth printf is deleted, the value of 101 will be interpreted as the decimal ASCII code and it is the ASCII code for **e**.

Basic C and Control Flow - Q3

Assume x and y are integer variables. What will happen if one of the following statements is executed?

(a) **`scanf("%d %d", &x, &y);`**

(b) **`scanf("%d %d", x, y);`**

(c) **`scanf("%d/%d", &x, &y);`**

Q3: Suggested Answer

(a) The program will wait for the input of two integers separated by one space (or one **white space**, i.e. the space, tab or newline character), e.g. 23 45.

(b) This is an errorous statement by omitting the '&', i.e. the address operator. The program will terminate abnormally.

(c) The program will wait for the input of two integers separated by one '/', e.g. 23/45.

Basic C and Control Flow - Q4

The output of the following code is not zero. Why?

```
{    .......
     double A = 373737.0;
     double B;

     B = A * A * A + 0.37/A - A * A * A - 0.37/A;
     printf(" The value of B is %f.\n", B);
}
```

Q4: Suggested Answer

- B is assigned a value which is mathematically zero. But on most machines, this value will not be zero, showing that even the double precision is not sufficient.
- When a very large number (A * A * A) is added to a very small number (0.37/A), the result is an approximation of the real sum. In this case the approximation is the very large number that we started with.
- Thus the subtraction gets the result down to zero, and the final value assigned to B is just –0.37/A. This effect is called **roundoff error.**
- Beware, after thousands of successive operations, the total roundoff error can be ridiculously high if care is not taken.
- This limitation is present in all programming languages, not just C.

Basic C and Control Flow - Q5

Given the following declarations and initial assignments:

```
int  i,  j,  m,  n;
float f,  g;
i  =  j  =  2;              m  =  n  =  5;
f  =  1.2;                  g  =  3.4;
```

evaluate the following expressions independently, i.e. all variables start with the same set of initial values. Show any conversions which take place and the type of result.

(a)	**m * j / j**	(b)	**m / j * j**
(c)	**(f + 10) * 20**	(d)	**(i++) * n**
(e)	**i++ * n**	(f)	**-12L * (g - f)**
(g)	**m = n = --j;**	(h)	**(int) g * 10**
(i)	**(int) (g * 10)**	(j)	**j = i + f**

Q5: Suggested Answer

(a) m * j / j = 5 * 2 / 2 = 5 (integer)

(b) m / j * j = 5 / 2 * 2 = (5 / 2) * 2 = 2 * 2 = 4 (integer)

(c) (f + 10) * 20 = (1.2 + 10.0) * 20 = 11.2 * 20.0 = 224.0 (float)

(d) (i++) * n = 2 * 5 = 10 (integer); after this i takes value of 3

(e) same as (d)

(f) -12L * (g - f) = -12L * (3.4 - 1.2) = -12. * 2.2 = -26.4 (float)

(g) m = n = --j ==> m = (n = (--j)) ==> m = (n = 1) ==> n = 1;
j is 1 too

(h) (int) g * 10 = 3 * 10 = 30 (integer)

(i) (int) (g * 10) = (int) (3.4 * 10.) = (int) 34.0 = 34 (integer)

(j) j = i + f : (float) i + f = 2.0 + 1.3 = 3.2 => j = (int) 3.2 = 3

Basic C and Control Flow - Q6

Which of the following are acceptable case constant expressions? Assume the convention that upper case is used for defining a constant, e.g.

```c
#define  SVALUE  10
```

and other identifiers are variables.

(a) case 76:

(b) case number*2:

(c) case SVALUE*2:

(d) case 80.1:

Q6: Suggested Answer

(a) valid because 76 is an integer constant.

(b) invalid because number is a variable.

(c) valid if SVALUE is an integer constant.

(d) invalid because it is not an integer.

Basic C and Control Flow - Q7

In some computer games it is necessary to introduce a delay to slow the computer down. Assume that you are running the following program on a computer which uses 16 bits to represent an integer. How can the delay be (a) shortened, (b) made a thousand times longer, (c) made variable after compilation?

```
#include  <stdio.h>
#define  DLENGTH 32000
int main() {
    int count;
    ......
    for (count = -DLENGTH; count <= DLENGTH; count++)
        ;   /* this is a NULL statement which does nothing */
    ......
}
```

Q7: Suggested Answer

(a) change the constant definition in #define to a lower value ;

(b) add an outer for loop :

```
for (t = 1; t <= 1000; t++)
    for (count = -DLENGTH; ....)
        ;
```

(c) instead of using the constant DLENGTH, use a **variable** which will hold a user specified value.

Basic C and Control Flow - Q8

Are the following code segments the same?

a) `if (x != 0 && 2/x != 1) { .....}`

b) `if (2/x != 1 &&  x != 0) { .....}`

Q8: Suggested Answer

- No. When x is zero, in (a) the condition evaluates to false (0) while in (b) it causes an illegal division by zero.
- This is called short circuit behavior of logical operators that skips evaluating parts of a (if/while/...) condition when able. In case of a logical operation on two operands, the first operand is evaluated (to true or false) and if there is a verdict (i.e. first operand is false when using &&, first operand is true when using ||), the second operand is not evaluated.

Basic C and Control Flow - Q9

Write a section of C program to interchange the values of two integer variables. Is there a way of solving this problem without using a third variable?

Q9: Suggested Answer

Assume the declaration:

```
int     first, second, temp;
```

The three assignment statements needed are:

```
temp = first;
first = second;
second = temp;
```

Alternatively, if we want to use only **two variables**:

```
first -= second;
second += first;
first = second - first;
```