CS 1037
Computer Science Fundamentals II

Part Five: Simple I/O

SIMPLE C PROGRAM

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
   char a;     /* 1 byte   */
   int b;      /* 4 bytes  */
   float c;    /* 4 bytes  */
   double d;   /* 8 bytes  */


   a = 'K';
   b = 37;
   c = 2.5;
   d = 75.3;

   printf( "1st value of a is : %c \n" , a );
   printf( "2nd value of b is : %d \n" , b );
   printf( "3rd value of c is : %f \n" , c );
   printf( "4rd value of d is : %lf \n" , d );

   return 0 ;
}
```

OUTPUT:

```
1st value of a is : K
2nd value of b is : 37
3rd value of c is : 2.50000000
4th value of d is : 75.50000000
```

PRINTING in C

In printing, C will view the **content** of a **variable** as a (**generic**) sequence of bits

C does not know (nor care) about the **data type** of the variable

You must tell (instruct) C on how to **interpret** the bit pattern !!!

The **printf()** function is used to print values of all built-in data types in C.

Syntax of the printf() function:

```
printf ( " format string " , value1, value2, ..... );
```

```
printf( "2nd value of b is : %d \n" , b );
```

The "format string" contains instructions on how to interpret each of the values in the parameter list

PRINTING in C

FORMAT STRING:
The format string in the printf() function contains **formatting characters** that instruct the C compiler to print a value in the given format

Formatting Character	Meaning
%d	Print the (next) value as a signed integer value
%u	Print the (next) value as a unsigned integer value
%ld	Print the (next) value as a long signed integer value
%lu	Print the (next) value as a long unsigned integer value
%f	Print the (next) value as a floating point value
%lf	Print the (next) value as a double precision floating point value
%c	Print the (next) value as a character (ASCII code)
%s	Print the (next) value as a string(to be explained later)

PRINTING in C

```
 int main( int argc, char* argv[] )
   {
      int i = 65, j = 'B';       /* ASCII code for 'B' = 66 */
      float x = 65.0;

      printf( "signed integer i: %d and signed integer j: %d\n", i, j );

      printf( "signed integers i: %c and j: %c as characters ", i, j );
      printf( "using ASCII code.\n");

      printf( "\n" );
      printf( "float x: %f\n", x );

      return (0);
}
```

```
signed integer i: 65 and signed integer j: 66
signed integers i: A and j: B as characters using ASCII code.

float x: 65.000000
```

PRINTING in C

```
int main( int argc, char* argv[] )
  {
     int i = 65, j = 'B';        /* A
     float x = 65.0;

     printf( "signed integer i: %d a

     printf( "signed integers i: %c
     printf( "using ASCII code.\n");

     printf( "\n" );
     printf( "float x: %f\n", x );

     return (0);
}
```

```
signed integer i: 65 and signed integer j: 6
signed integers i: A and j: B as characters u

float x: 65.000000
```

Label	Address	Value	Binary
	399		
i	400	65	0000 0000
	401		0000 0000
	402		0000 0000
	403		0100 0001
j	404	66	0000 0000
	405		0000 0000
	406		0000 0000
	407		0100 0010
x	408	65.0	0000 0010
	409		0000 0000
	410		0000 0000
	411		0000 0000
	412		0000 0000
	413		0000 0000
	414		0000 0000
	415		0000 0000
	416		
	417		
	418		
	…		

PRINTING in C

WARNING:
The **C compiler** do ***not*** perform any **type checks** in the **printf( )** function call
You must **make sure** that the **data type** of the **variables correspond** to **formatting character**

```
 int main( int argc, char* argv[] )
   {
      int i = 65, j = 'B';       /* ASCII code for 'B' = 66 */
      float x = 65.0;

      printf( "signed integer i: %f and signed integer j: %f\n", i, j );

      printf( "signed integers i: %lu and j: %u as characters ", i, j );
      printf( "using ASCII code.\n");

      printf( "\n" );
      printf( "float x: %d\n", x );

      return (0);
}
```

```
signed integer i: 0.000000 and signed integer j: 0.000000
process returned -1073741819 (0xC000000005)
```

PRINTING in C

WARNING:
The **C compiler** do ***not*** perform any **type che**
You must **make sure** that the **data type** of the **cter**

```
 int main( int argc, char* argv[] )
   {
      int i = 65, j = 'B';        /* ASC
      float x = 65.0;

      printf( "signed integer i: %f an

      printf( "signed integers i: %lu a
      printf( "using ASCII code.\n");

      printf( "\n" );
      printf( "float x: %d\n", x );

      return (0);
}
```

```
signed integer i: 0.000000 and signed intege
process returned -1073741819 (0xC000000
```

Label	Address	Value	Binary
	399		
i	400	65	0000 0000
	401		0000 0000
	402		0000 0000
	403		0100 0001
j	404	66	0000 0000
	405		0000 0000
	406		0000 0000
	407		0100 0010
x	408	65.0	0000 0010
	409		0000 0000
	410		0000 0000
	411		0000 0000
	412		0000 0000
	413		0000 0000
	414		0000 0000
	415		0000 0000
	416		
	417		
	418		
	…		

PRINTING in C

printf() special characters:
The following character sequences have a special meaning when used as printf format specifiers

Formatting Character	Meaning
`\a`	audible alert
`\b`	backspace
`\f`	form feed
`\n`	newline, or linefeed
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\\`	backslash

PRINTING in C

printf() special characters:
examples:

Description	Code	Result
Insert a tab character in a string	printf("Hello\tworld");	Hello world
Insert a newline character in a string	printf("Hello\nworld");	Hello world
Typical use of the newline character	printf("Hello world\n");	Hello world
A DOS/Windows path with backslash characters	printf("C:\\Windows\\System32\\");	C:\Windows\System32\

PRINTING in C

Controlling integer width with printf
The %3d specifier is used with integers, and means a minimum width of three spaces, which, by default, will be right-justified:

printf("%3d", 0);	0
printf("%3d", 123456789);	123456789
printf("%3d", -10);	-10
printf("%3d", -123456789);	-123456789

PRINTING in C

Left-justifying printf integer output

To left-justify integer output with printf, just add a minus sign (-) after the % symbol, like this:

printf("%-3d", 0);	0
printf("%-3d", 123456789);	123456789
printf("%-3d", -10);	-10
printf("%-3d", -123456789);	-123456789

PRINTING in C

The printf integer zero-fill option

To zero-fill your printf integer output, just add a zero (0) after the % symbol, like this:

printf("%03d", 0);	000
printf("%03d", 1);	001
printf("%03d", 123456789);	123456789
printf("%03d", -10);	-10
printf("%03d", -123456789);	-123456789

PRINTING in C

printf integer formatting

As a summary of printf integer formatting, here's a little collection of integer formatting examples. Several different options are shown, including a minimum width specification, left-justified, zero-filled, and also a plus sign for positive numbers.

Description	Code	Result
At least five wide	printf("'%5d'", 10);	' 10'
At least five-wide, left-justified	printf("'%-5d'", 10);	'10 '
At least five-wide, zero-filled	printf("'%05d'", 10);	'00010'
At least five-wide, with a plus sign	printf("'%+5d'", 10);	' +10'
Five-wide, plus sign, left-justified	printf("'%-+5d'", 10);	'+10 '

PRINTING in C

formatting floating point numbers with printf

Here are several examples showing how to format floating-point numbers with printf:

Description	Code	Result
Print one position after the decimal	printf("'%.1f'", 10.3456);	'10.3'
Two positions after the decimal	printf("'%.2f'", 10.3456);	'10.35'
Eight-wide, two positions after the decimal	printf("'%8.2f'", 10.3456);	' 10.35'
Eight-wide, four positions after the decimal	printf("'%8.4f'", 10.3456);	' 10.3456'
Eight-wide, two positions after the decimal, zero-filled	printf("'%08.2f'", 10.3456);	'00010.35'
Eight-wide, two positions after the decimal, left-justified	printf("'%-8.2f'", 10.3456);	'10.35 '
Printing a much larger number with that same format	printf("'%-8.2f'", 101234567.3456);	'101234567.35'

PRINTING in C

printf string formatting
Here are several examples that show how to format string output with printf:

Description	Code	Result
A simple string	printf("'%s'", "Hello");	'Hello'
A string with a minimum length	printf("'%10s'", "Hello");	' Hello'
Minimum length, left-justified	printf("'%-10s'", "Hello");	'Hello '

USER INPUT in C

Reading in value of the built-in data types

The **scanf()** function is used to read in **values** of ***all*** **built-in data types** in **C**.

Syntax of the scanf() function:

```
scanf ( " format string " , &var1, &var2, ..... );
```

```
scanf( "%d" , &x );
```

The **format string** in the **scanf()** function contains **formatting characters** that instruct the **C compiler** to ***read in*** a **value** and **store** it in the given **representation (encoding memory)**

FORMAT STRING:

The format string in the scanf() function contains the exact same **formatting characters** that are used by the printf() function to print a value in the given format

USER INPUT in C

```
int main( int argc, char* argv[] )
  {
     int a;
     float y;

     printf( "Enter an integer value:");
     scanf( "%d", &a );
     printf( "a = %d\n", a);

     printf( "Enter a floating point value:");
     scanf( "%f", &y );
     printf( "y = %f\n", y);
  }
```

```
Enter an integer value: 37
a = 37
Enter a floating point value: 3.14159
y = 3.141590
```

USER INPUT in C

```
int main( int argc, char* argv[] )
  {
     int a;
     float y;

     printf( "Enter an integer value
     scanf( "%d", &a );
     printf( "a = %d\n", a);

     printf( "Enter a floating point
     scanf( "%f", &y );
     printf( "y = %f\n", y);
  }
```

```
Enter an integer value: 37
a = 37
Enter a floating point value: 3.14159
y = 3.141590
```

Label	Address	Value	Binary
	399		
a	400	37	0000 0000
	401		0000 0000
	402		0000 0000
	403		001 0001
y	404	3.14159	0000 0100
	405		1100 1011
	406		0010 1111
	407		000 00000
	408		0000 0000
	409		0000 0000
	410		0000 0000
	411		0000 0000
	412		
	413		
	414		
	415		
	416		
	417		
	418		
	…		

USER INPUT in C

Reading in value of the built-in data types
The **scanf()** function is used to read in **values** of ***all*** **built-in data types** in **C**.

Syntax of the scanf() function:

```
    scanf ( &var1, &var2, ..... );
```

The **&** character is the **"reference" operator** of the **C** programming language

The expression **&x** means: the **address** of the variable **x**

You **must** pass the **address** of a variable to the **scanf()** function for reading operations.