Department of Computer and Information Sciences
Towson University
Laboratory for Operating Systems Course

Lab 2: Intro to C

Learning objective of this lab:

- To become familiar with C language

This lab allows students who have not programmed in C to gain basic familiarity with this language.

After completing this lab, students will be able to write simple programs in C and better understand Linux system calls which are written in C.

Introduction to C

Every full C program begins inside a function called "main". The main function is always called when the program first executes.

To access the standard functions that comes with your compiler, you need to include a header files with the #include directive.

C uses the following standard variable types:

```
int    -> integer variable
short  -> short integer
long   -> long integer
float  -> single precision real (floating point) variable
double -> double precision real (floating point) variable
char   -> character variable (single byte)
```

The **printf** function is the standard C way of displaying output on the screen. The general syntax is printf("format", variables); The printf function uses its first argument, "format", to determine how many arguments will follow and of what types they are. If you don't use enough arguments or if they are of the wrong type than printf will get confuses, with as a result wrong answers.

- %d (print as a decimal integer)
- %6d (print as a decimal integer with a width of at least 6 wide)
- %e (the scientific notation format)
- %f (print as a floating point)
- %4f (print as a floating point with a width of at least 4 wide)

- %.4f (print as a floating point with a precision of four characters after the decimal point)
- %3.2f (print as a floating point at least 3 wide and a precision of 2)
- %c (print as a character)
- %s (print as a string)
- %u (print as an unsigned integer)
- %o (the unsigned octal format)
- %x (the unsigned hexadecimal format)
- \n (escape sequence -- new line)
- \t (tab)

Example:

```
#include <stdio.h>
int main ()
{
  /* variable definition: */
  int a, b;
  int c;
  float f;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;
  printf("value of c : %d \n", c);

  f = 70.0/3.0;
  printf("value of f : %f \n", f);

  return 0;
}
```

Question

1. Write a program which displays an integer value in different formats (e.g. integer, float, octal, hexadecimal, etc.).

Pointers

Every memory location has:

- a value
- a unique address

Declaring a variable - sets up memory location

- int alpha; //allocates space in memory for sizeof int, at an address
- alpha = 6; //assigns a value to memory location

Pointers point to locations in memory. The pointer declaration looks like this:

```
<variable_type> *<name>;
```

For example, you could declare a pointer that stores the address of an integer with the following syntax:

```
int   *int_ptr;
float *ftPtr;
```

Address-of operator

To get the memory address of a variable (its location in memory), we use the address of operator "&" – it returns the memory address.

```
int    i = 0;
int*  iptr;
iptr = NULL;
iptr = &i;          //pointer contains address
```

Dereferencing

The content of the memory location referenced by a pointer is obtained using the ``*" operator, this is called *dereferencing* the pointer. Thus, *iptr refers to the value of x. The following two assignment statements produce the same result.

```
i = 40;
*iptr = 40;
```

Example 1

```
#include <stdio.h>

void main()
{
   int x;                    /* A normal integer*/
   int *xp;                  /* A pointer to an integer

   x=100;
   xp = &x;                  /*  assigns the address of x to xp" */
   printf( " value of x  = %d\n", x );
   printf( " value of x  = %d\n", *xp );  /* Note the use of the * to get the value */
   printf(" address of x =%8x \n", xp);
}
```

Example 2

```
void main()
{
   float x, y;               /* x and y are of float type        */
   float *fp, *fp2;          /* fp and fp2 are pointers to float  */

   x = 6.5;                  /* x now contains the value 6.5              */

/* print contents and address of x   */
   printf("Value of x is %f, address of x %8x\n", x, &x);

   fp = &x;                                    /* fp now points to location of x    */

  /* print the contents of fp       */
   printf("Value in memory location fp(%8x) is %f\n", *fp);

                                               /* change content of memory location */
   *fp = 9.2;
   printf("New value of x is %f = %f \n", *fp, x);

                                               /* perform arithmetic                */
   *fp = *fp + 1.5;
   printf("Final value of x is %f = %f \n", *fp, x);

                                               /* transfer values                   */
   y = *fp;
   fp2 = fp;
   printf("Transfered value into y = %f and fp2 = %f \n", y, *fp2);
}
```

Array name is a constant pointer and points to the location of the first array element.

```
#include <stdio.h>

void main()
{
   int arr[10];
   int i;

   for (i=0; I <10; i++)
            arr[i] = 10*i;


   printf(" address of arr[0] = %8x \n", &arr[0]);
   printf(" address of arr[0] = %8x \n", arr);

  int *temp = arr;

  for (i=0; I <10; i++)
            printf("%d \t", *temp++);

 printf("\n");

}
```

Pointer arithmetic

the pointer is incremented/decremented according to the size of the object it points to

```
int i,*ptr = &i;          // ptr contains address 100

ptr + 1                   // 100 + 1 * sizeof(int) = 104

ptr + 3                   // 100 + 3 * sizeof(int) = 112

ptr - 2                   // 100 - 2 * sizeof(int) = 92
```

Questions

2. Which of the following is the proper declaration of a pointer?

   A. int x;
   B. int &x;
   C. ptr x;
   D. int *x;

3. Which of the following gives the memory address of integer variable a?\

   A. *a;
   B. a;

C. &a;
D. address(a);

4. Which of the following gives the value stored at the address pointed to by pointer a?
   A. a;
   B. val(a);
   C. *a;
   D. &a;

Reading Input

scanf() function is used to read character, string, numeric data from stdin.

int scanf(format string, argument);

It reads data and stores them according to the parameter *format* into the locations **pointed** by the additional arguments. The additional arguments should point to already allocated objects of the type specified by their corresponding format specifier within the *format* string.

```
int main()

{
  int ID;
  char name[20];
  printf("Enter your ID: ");
  scanf("%d", &ID);
  printf(“Enter you name: “);
  scanf(“%s”, name);   //remember that array name is a pointer.
  printf("Entered ID:%d\n", ID);
  printf(“Entered Name: %s\n”, name);
  return(0);
}
```

Structures

Structures provide a way of storing many different values in variables of potentially different types under the same name.

```
struct  struct_name{
  Members
};
```

Where struct_name is the name of the entire type of structure and members are the variables within the struct. The syntax for creating a single structure is:

```
struct struct_name     name_of_single_structure;
```

To access a variable of the structure, we can use the dot '.' Operator.

```
struct EmployeeType
{
  int id_number;
  char name[50];
  int age;
  float salary;
};

int main()
{
  struct EmployeeType  emp1, emp2;

  emp1.age = 22;
  emp1.id_number = 1;
  emp1.salary = 12000.0;
  emp1.name ="John Doe";
  emp2.age = 45;
  …..
}
```

We can have array of structures and well as nested structures. If you wish to have a pointer to a structure, to actually access the information stored inside the structure that is pointed to, you use the -> operator in place of the . operator. All points about pointers still apply.

```
int main()
{
   struct EmployeeType  emp1;
   struct EmployeeType  *emp_ptr;

   emp_ptr = &emp1;
   emp1.name = “Jane Doe”;
   emp_ptr-> age = 25;
   printf( "%d\n",  );
   return 0;
}
```

Questions:

5. Which of the following accesses a variable in structure b?

A. b->var;
B. b.var;
C. b-var;
D. b>var;

6. Which of the following accesses a variable in a pointer to a structure, *b?

A. b->var;
B. b.var;
C. b-var;
D. b>var;

7. Which of the following is a properly defined struct?

A. struct {int a;}
B. struct a_struct {int a;}
C. struct a_struct int a;
D. struct a_struct {int a;};

8. Which properly declares a variable of struct foo?

A. struct foo;
B. struct foo var;
C. foo;
D. int foo;

References:

http://www.tutorialspoint.com/cprogramming/c_quick_guide.htm
http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/