CSE321 - Operating Systems Lab

Lab 03: Introducing C Programming (02)

Summary

- Flow Statement in C
- Loops in C
- Function in C
- FILE I/O
- Command Line Arguments

if statement

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 )
   {
       /* if condition is true then print the following */
       printf("a is less than 20\n" );
   }
   else
   {
       /* if condition is false then print the following */
       printf("a is not less than 20\n" );
   }
   printf("value of a is : %d\n", a);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
a is not less than 20;

value of a is : 100
```

While Loop:

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

For Loop:

```c
#include <stdio.h>
int main ()
{
    /* for loop execution */
    for( int a = 10; a < 20; a = a + 1 )
    {
        printf("value of a: %d\n", a);
    }
    return 0;
}
```

Nested Loop:

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int i, j;

   for(i=2; i<100; i++) {
      for(j=2; j <= (i/j); j++)
        if(!(i%j)) break; // if factor found, not prime
        if(j > (i/j)) printf("%d is prime\n", i);
   }

   return 0;
}
```

Function in C

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Function call by value

passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

```
/* function definition to swap the values */
void swap(int x, int y)
{
   int temp;

   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put x into y */

   return;
}
```

```
#include <stdio.h>

/* function declaration */
void swap(int x, int y);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   swap(a, b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

Let us put above code in a single C file, compile and execute it, it will produce the following result:

```
Before swap, value of a :100

Before swap, value of b :200
```

Function call by reference

passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

```
/* function definition to swap the values */
void swap(int *x, int *y)
{
   int temp;
   temp = *x;    /* save the value at address x */
   *x = *y;      /* put y into x */
   *y = temp;    /* put x into y */

   return;
}
```

```
#include <stdio.h>

/* function declaration */
void swap(int *x, int *y);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values.
    * &a indicates pointer to a ie. address of variable a and
    * &b indicates pointer to b ie. address of variable b.
   */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );
```

```
Before swap, value of a :100

Before swap, value of b :200

After swap, value of a :100

After swap, value of b :200
```

FILE I/O

```c
#include <stdio.h>

main()
{
   FILE *fp;

   fp = fopen("/tmp/test.txt", "w+");
   fprintf(fp, "This is testing for fprintf...\n");
   fputs("This is testing for fputs...\n", fp);
   fclose(fp);
}
```

```c
#include <stdio.h>

main()
{
   FILE *fp;
   char buff[100];

   fp = fopen("/tmp/test.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1 : %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);

}
```

Command Line Arguments

pass some values from the command line to your C programs when they are executed. These values are called command line arguments.

The command line arguments are handled using main() function arguments where argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program.

It should be noted that argv[0] holds the name of the program itself and argv[1] is a pointer to the first command line argument supplied, and *argv[n] is the last argument.

```c
#include <stdio.h>

int main( int argc, char *argv[] )
{
   if( argc == 2 )
   {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 )
   {
      printf("Too many arguments supplied.\n");
   }
   else
   {
      printf("One argument expected.\n");
   }
}
```

Some examples of executing this program -

```
$./a.out testing

The argument supplied is testing
```

```
$./a.out testing1 testing2

Too many arguments supplied.
```

```
$./a.out

One argument expected
```

Lab Tasks:

1. Write a C program to check if the entered character is vowel or consonant.
2. Write a C program to check whether a triangle can be formed by the given value for the angles.
3. Write a C program to find the sum of even numbers between 1 to n.
4. Write a C program to reverse the numbers in an array.
5. Write a C program to find diameter, circumference and area of circle using function.
6. Write a C program to swap two numbers in an array using function call by reference.
7. Write a C program to open a text file and print the characters [a-z, A-Z] only from that file.
8. Write a C program to sort given numbers using command line arguments.