CS11001/CS11002
Programming and Data Structures (PDS)
(Theory: 3-1-0)

What does this program do?

```
#include <stdio.h>

  main ()
  {
    unsigned int n, i, j, s;

    printf("Enter a positive integer : ");
    scanf("%d",&n);
    s = 0x00000041 ^ (unsigned int)'A';
     printf("s = %d\n", s);

    while (i = --n) while (j = --i) while (--j) ++s;
    printf("s = %d\n", s);
  }
```

What is the output for n= 3, 4, 5, 6?

Functions without Recursions

- A function is expected to carry out a specific job depending on the argument values passed to it.
- After the job is accomplished, the function returns some value to the caller.
- The basic syntax of writing a function goes like this:
- *return_type* function_name (*list_of_arguments*) { function body }

Components of a function

- The argument list should be a comma-separated list of type-name pairs, where type is any valid data type and name is any legal formal name of a variable. Argument values can be accessed inside the function body using these names.
- The return type of a function should again be a valid data type (like int, float, char *). A function may even choose to return no explicit values in which case its return type is to be mentioned as void.
- The function body starts with a declaration of **local variables**. These variables together with the function arguments are accessible only in the function body and not outside it.
- After the declarations one writes the C statements that compute the specific task that the function is meant for. The function returns a value using the statement:
  - **return (*return_value*);**

Calling a function

- The parentheses around *return_value* are optional. In case a function is expected to return nothing (i.e., void), the return statement looks like:
- return; The return statement not only returns a value (possibly void) to the caller, but also returns the control back to the place where it is called.
- In case no explicit return statements are present in the function body, control goes back to the caller after the entire body of the function is executed.
- Calling a function uses the following syntax:
  - function_name (*argument_values*)

Example

```
int gcd ( int a , int b )
{ int r;
/* Check for errors : gcd(0,0) is undefined */
if ((a==0) && (b==0)) return (0);
/* Make the arguments non-negative */
if (a < 0) a = -a; if (b < 0) b = -b;
/* Special case : gcd(a,0) = a */
if (b == 0) return (a);
/* The Euclidean gcd loop */
while (1)
{ r = a % b; if (r == 0) return (b); a = b; b = r; } }
```

Calling the function

```
int main ()
{ int i, j, s;
   s = 0;
   for (i=1; i<=20; ++i)
    { for (j=i; j<=20; ++j)
          { s += gcd(j,i); }
    }
 printf("The desired sum = %d\n", s);
}
```

Printing a message through a function

```
printmesg()
{
  printf("This is a print function\n");
}

void main()
{
 printmesg();
}
```

Another example

```
int cube(int i)
{
  int retvalue;
  retvalue = i * i * i;
  return(retvalue);
}
```

The main function

```
#include<stdio.h>
void main()
{
  int n;
  printf(“Enter a number: “);
  scanf(“%d”,&n);
  printf(“\n The cube is %d\n”,cube(n));
}
```

The main function with an additional check

```
#include<stdio.h>
main()
{
 int n;
  printf(“Enter a number: “);
  if(scanf(“%d”,&n)<1)
        printf(“You have to enter an integer\n”);
  else
       printf(“\n The cube is %d\n”,cube(n));
}
```