CS1100 – Introduction to Programming

Trimester 3, April – June 2021

Instructor:

Shweta Agrawal (shweta.a@cse.iitm.ac.in)

Lecture 20

`functions` in C

`functions` in C-language helps us to :

`functions` in C

`functions` in C-language helps us to :

- Define our own subtasks which we want to use in bigger tasks and program them to reuse them whenever needed. This is called **modular approach** to program design. Very effective and less error-prone.

`functions` in C

`functions` in C-language helps us to :

- Define our own subtasks which we want to use in bigger tasks and program them to reuse them whenever needed. This is called **modular approach** to program design. Very effective and less error-prone.
- Define our own functions, and use them.

`functions` in C

`functions` in C-language helps us to :

- Define our own subtasks which we want to use in bigger tasks and program them to reuse them whenever needed. This is called **modular approach** to program design. Very effective and less error-prone.
- Define our own functions, and use them.
- Re-use lots of code, tested code.

`functions` in C

`functions` in C-language helps us to :

- Define our own subtasks which we want to use in bigger tasks and program them to reuse them whenever needed. This is called **modular approach** to program design. Very effective and less error-prone.
- Define our own functions, and use them.
- Re-use lots of code, tested code.
- Giving a job to `functions` $\equiv$ outsourcing.

Example : Find Sum

```
#include "stdio.h"
int FindSum(int, int);

int main()
{
    int var1 = 10;
    int var2 = 20;
    int var3,var4;
    var3 = FindSum(var1,var2);
    var4 = FindSum(var3,var2);
    printf("%d\n", var3);
    printf("%d\n", var4);
    return 0;
}

int FindSum(int a, int b)
{
     int c=a+b;
     return c;
     }
```

Example : implementing `fact`

```
#include<stdio.h>
int fact(int);

int main() {
    int x, y;
    printf("Enter a number:");
    scanf("%d", &x);
    y = fact(x);
    printf("%d\n",y);
}

int fact(int n) {
    int i = 1;
    while(n>0) {
        i = i * n;
        n--;
    }
    return i;
}
```

Example : Checking co-primeness

```
#include "stdio.h"
int GCD (int m, int n) {
  int rem;
  do {
    rem = m % n;
    m = n;
    n = rem;
  } while (rem != 0);
  return m; }
int main () {
  int x, y, gcd;
  printf ("input two nonzero positive integers:");
  scanf ("%d %d", &x, &y);
  gcd = GCD (x, y);
  if (gcd == 1)
    printf ("%d and %d are coprime\n", x, y);
  else
    printf ("%d and %d are not coprime\n", x, y); }
```

Example : Matrix Multiplication by Repeated Addition

```
#include "stdio.h"
int mult(int a, int b) {
  int i;
  int sum = 0;
  for (i = 1; i <= a; i++)
    sum = sum + b;
  return sum;
}
int main () {
  int x, y;
  printf ("input two integers (positive)");
  scanf ("%d %d", &x, &y);
  printf ("Product of %d and %d is %d\n",x, y,  mult(x,y));
  return 0;
}
```

Example : Finding Prime Numbers in an Interval

```
#include <stdio.h>
int checkPrimeNumber(int n);
int main() {
    int n1, n2, i, flag;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    printf("Prime numbers between %d and %d are: ", n1, n2);
    for (i = n1 + 1; i < n2; ++i) {
      flag = checkPrimeNumber(i);
        if (flag == 1)   printf("%d ", i);     }
    return 0; }
int checkPrimeNumber(int n) {
    int j, flag = 1;
    for (j = 2; j <= n / 2; ++j) {
        if (n % j == 0) {
            flag = 0;
            break;
        }  }
    return flag; }
```

Example : Swapping Two Numbers

```
#include<stdio.h>
void swap(int *a, int *b);

int main()
{
    int m = 22, n = 44;
    //  calling swap function by reference
    printf("values before swap m = %d \n and n = %d",m,n);
    swap(&m, &n);
}

void swap(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    printf("\n values after swap a = %d \nand b = %d", *a, *b);
}
```

Example : Binary to Decimal Conversion

```
#include <math.h>
#include <stdio.h>
int convert(long long n);
int main() {
    long long n;
    printf("Enter a binary number: ");
    scanf("%lld", &n);
    printf("%lld in binary = %d in decimal", n, convert(n));
    return 0;
}
int convert(long long n) {
    int dec = 0, i = 0, rem;
    while (n != 0) {
        rem = n % 10;
        n /= 10;
        dec += rem * pow(2, i);
        ++i; }
    return dec; }
```

Yesterday's Example: Character Grid

How do we write `SearchPattern(string, pattern)`?

Yesterday's Example: Character Grid

How do we write `SearchPattern(string, pattern)`?

Simpler Task : SearchPatternFrom(string,pattern,from)
Is the pattern the substring of string starting at the `from`-th position of string? Answer **Yes/No**

Yesterday's Example: Character Grid

How do we write `SearchPattern(string, pattern)`?

Simpler Task : SearchPatternFrom(string,pattern,from)
Is the pattern the substring of string starting at the `from`-th position of string? Answer **Yes/No**

```
int search_pattern_from()
{
        int j;
        for(j=0;j<pattern_length;j++)
        {
            if(string[from+j]=='\0') return 0;;
            if(pattern[j]!=string[from+j]) return 0;
        }
        return 1;
}
```

Yesterday's Example: Character Grid

```
char string[1024],pattern[1024];
int string_length, pattern_length,from;

int search_pattern(void);
int search_pattern_from(void);

int search_pattern()
{
    int i,yesno;

    i = 0;
    for (i=0; i<string_length; i++)
    {
     from = i;
     yesno = search_pattern_from();
     if (yesno != 0) return (from+1);
    }
    return(-1);
}
```

De-mystifying the `main()` function

- When we type `./a.out` the control is set to be transferred to the starting point of the main. (This is set to be so by the C-compiler when it produced the `a.out` file.)

De-mystifying the `main()` function

- When we type `./a.out` the control is set to be transferred to the starting point of the main. (This is set to be so by the C-compiler when it produced the `a.out` file.)
- Who “calls” the main()?

De-mystifying the `main()` function

- When we type `./a.out` the control is set to be transferred to the starting point of the main. (This is set to be so by the C-compiler when it produced the `a.out` file.)
- Who “calls” the main()? The command-line program, which is a part of the operating system on which the entire program is running - calls the `main()`.

De-mystifying the `main()` function

- When we type `./a.out` the control is set to be transferred to the starting point of the main. (This is set to be so by the C-compiler when it produced the `a.out` file.)
- Who “calls” the main()? The command-line program, which is a part of the operating system on which the entire program is running - calls the `main()`.
- Can main have arguments?

De-mystifying the `main()` function

- When we type `./a.out` the control is set to be transferred to the starting point of the main. (This is set to be so by the C-compiler when it produced the `a.out` file.)
- Who “calls” the main()? The command-line program, which is a part of the operating system on which the entire program is running - calls the `main()`.
- Can main have arguments? Yes, if we want to pass on a value to the program while executing a.out, it can be passed as an argument.

Blocks and Scope: Recap

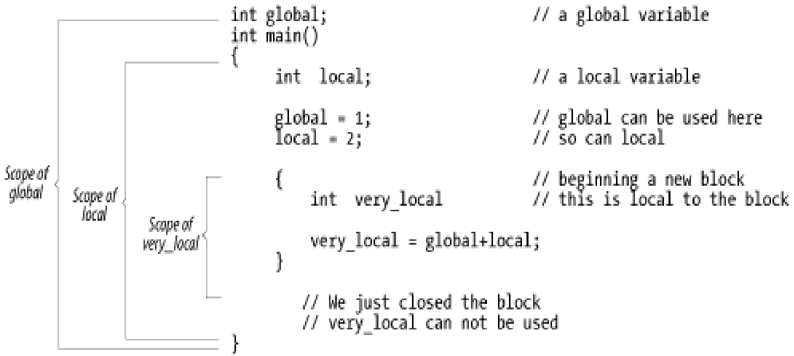
Block : A program segment written within curley brackets.

Scope : The program segment where a particular declaration of a variable is applicable.

Blocks and Scope: Recap

Block : A program segment written within curley brackets.

Scope : The program segment where a particular declaration of a variable is applicable.



```
int global;                           // a global variable
int main()
{
    int  local;                       // a local variable

    global = 1;                       // global can be used here
    local = 2;                        // so can local

    {                                 // beginning a new block
        int  very_local               // this is local to the block

        very_local = global+local;
    }

       // We just closed the block
       // very_local can not be used
}
```

Practicing the Concept : Blocks and Scope

```c
#include <stdio.h>
int FindSum(int, int);
int var1 = 10;

int main()
{
    int var2 = 20;
    {
      int var3;
      var3 = FindSum(var1,var2);
      printf("%d\n", var3);
    }
    float var3=100;
    printf("%f\n", var3);
    return 0;
}
```

```c
int FindSum(int a, int b)
{
     int c=a+b;
     printf("%d\n",var1);
     return c;
}
```

Practicing the Concept : Blocks and Scope

```
#include <stdio.h>
int FindSum(int, int);
int var1 = 10;

int main()
{
    int var2 = 20;
    {
      int var3;
      var3 = FindSum(var1,var2);
      printf("%d\n", var3);
    }
    float var3=100;
    printf("%f\n", var3);
    return 0;
}
```

```
int FindSum(int a, int b)
{
     int c=a+b;
     printf("%d\n",var1);
     return c;
}
```

- Scope of `var2` is the whole of `main`

Practicing the Concept : Blocks and Scope

```
#include <stdio.h>
int FindSum(int, int);
int var1 = 10;

int main()
{
    int var2 = 20;
    {
      int var3;
      var3 = FindSum(var1,var2);
      printf("%d\n", var3);
    }
    float var3=100;
    printf("%f\n", var3);
    return 0;
}
```

```
int FindSum(int a, int b)
{
     int c=a+b;
     printf("%d\n",var1);
     return c;
}
```

- Scope of `var2` is the whole of `main`
- Scope of `int var3` is only the inner block.

Practicing the Concept : Blocks and Scope

```
#include <stdio.h>
int FindSum(int, int);
int var1 = 10;

int main()
{
    int var2 = 20;
    {
      int var3;
      var3 = FindSum(var1,var2);
      printf("%d\n", var3);
    }
    float var3=100;
    printf("%f\n", var3);
    return 0;
}
```

```
int FindSum(int a, int b)
{
     int c=a+b;
     printf("%d\n",var1);
     return c;
}
```

- Scope of `var2` is the whole of `main`
- Scope of `int var3` is only the inner block.
- Scope of `float var3` is only the outer block.

Practicing the Concept : Blocks and Scope

```
#include <stdio.h>
int FindSum(int, int);
int var1 = 10;

int main()
{
    int var2 = 20;
    {
      int var3;
      var3 = FindSum(var1,var2);
      printf("%d\n", var3);
    }
    float var3=100;
    printf("%f\n", var3);
    return 0;
}
```

```
int FindSum(int a, int b)
{
     int c=a+b;
     printf("%d\n",var1);
     return c;
}
```

- Scope of `var2` is the whole of `main`
- Scope of `int var3` is only the inner block.
- Scope of `float var3` is only the outer block.
- Scope of `int var1` is the whole program.

Practicing the Concept : Blocks and Scope

```
#include <stdio.h>
int FindSum(int, int);
int var1 = 10;

int main()
{
    int var2 = 20;
    {
      int var3;
      var3 = FindSum(var1,var2);
      printf("%d\n", var3);
    }
    float var3=100;
    printf("%f\n", var3);
    return 0;
}
```

```
int FindSum(int a, int b)
{
     int c=a+b;
     printf("%d\n",var1);
     return c;
}
```

- Scope of `var2` is the whole of `main`
- Scope of `int var3` is only the inner block.
- Scope of `float var3` is only the outer block.
- Scope of `int var1` is the whole program.

Local vs **Global** variables : `var1` is global but `var2` is local for `main` function.

Use of static

```
#include "stdio.h"
void DoSomething() {
  static int x=5;
  {
    static int y=6;
    x++;
    y++;
    printf ("x = %d y = %d\n", x, y);
  }
}
int main () {
  int i;
  for (i = 1; i < 10; i++)
    DoSomething();
}
```

Take Aways

- Functions : Modular Programming. Build programs brick by brick. Reusing built and tested part.
- Declaration, Definition and Invocation of functions.
- Block and Scope. Local and Global Variables.
- prefixing variables with `static`

Coming up :

- We will do hands-on examples of using functions.
- Is `main` program a function?
  Why are we ending with `"return 0;"` Who is it returning to?

Take Aways

- Functions : Modular Programming. Build programs brick by brick. Reusing built and tested part.
- Declaration, Definition and Invocation of functions.
- Block and Scope. Local and Global Variables.
- prefixing variables with `static`

Coming up :

- We will do hands-on examples of using functions.
- Is `main` program a function?
  Why are we ending with "`return 0;`" Who is it returning to?
- Can a function invoke other functions?

Take Aways

- Functions : Modular Programming. Build programs brick by brick. Reusing built and tested part.
- Declaration, Definition and Invocation of functions.
- Block and Scope. Local and Global Variables.
- prefixing variables with `static`

Coming up :

- We will do hands-on examples of using functions.
- Is `main` program a function?
  Why are we ending with `"return 0;"` Who is it returning to?
- Can a function invoke other functions? Yes !

Take Aways

- Functions : Modular Programming. Build programs brick by brick. Reusing built and tested part.
- Declaration, Definition and Invocation of functions.
- Block and Scope. Local and Global Variables.
- prefixing variables with `static`

Coming up :

- We will do hands-on examples of using functions.
- Is `main` program a function?
  Why are we ending with `"return 0;"` Who is it returning to?
- Can a function invoke other functions? Yes !
- Can a function invoke itself?

Take Aways

- Functions : Modular Programming. Build programs brick by brick. Reusing built and tested part.
- Declaration, Definition and Invocation of functions.
- Block and Scope. Local and Global Variables.
- prefixing variables with `static`

Coming up :

- We will do hands-on examples of using functions.
- Is `main` program a function?
  Why are we ending with `"return 0;"` Who is it returning to?
- Can a function invoke other functions? Yes !
- Can a function invoke itself? Yes ! Recurison !.

Hands-on Example : Referee of Tic-Tac-Toe

X	X	O
		O
O	X	

- Two Player Game (X-player & O-player).

Hands-on Example : Referee of Tic-Tac-Toe

X	X	O
		O
O	X	

- Two Player Game (X-player & O-player).
- The game proceeds when each player places 'X' or 'O' in a blank space in the matrix in alterante turns.

Hands-on Example : Referee of Tic-Tac-Toe

$$\begin{array}{c|c|c} X & X & O \\ \hline & & O \\ \hline O & X & \end{array}$$

- Two Player Game (X-player & O-player).
- The game proceeds when each player places 'X' or 'O' in a blank space in the matrix in alterante turns.

- Initial configuration : the board is empty.

Hands-on Example : Referee of Tic-Tac-Toe

X	X	O
		O
O	X	

- Two Player Game (X-player & O-player).
- The game proceeds when each player places 'X' or 'O' in a blank space in the matrix in alterante turns.

- Initial configuration : the board is empty.
- Winning : if there is a sequence of three consecutive cells (vertical, horizontal, forward diagonal or reverse diagonal) where the player's symbol appears.

X	X	O
	X	O
O	X	

Hands-on Example : Referee of Tic-Tac-Toe

X	X	O
		O
O	X	

- Two Player Game (X-player & O-player).
- The game proceeds when each player places 'X' or 'O' in a blank space in the matrix in alterante turns.

- Initial configuration : the board is empty.
- Winning : if there is a sequence of three consecutive cells (vertical, horizontal, forward diagonal or reverse diagonal) where the player's symbol appears.
- Draw : if the board is full, but neither of the players has reached a winning configuration yet.

X	X	O
	X	O
O	X	

X	X	O
O	O	X
X	X	O

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Think modular : Tasks involved for a referee - <u>the board keeper</u>.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Think modular : Tasks involved for a referee - the board keeper.

- Show the board to both players.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Think modular : Tasks involved for a referee - <u>the board keeper</u>.

- Show the board to both players.
- Check if any of them won, if so, declare won.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Think modular : Tasks involved for a referee - the board keeper.

- Show the board to both players.
- Check if any of them won, if so, declare won.
- If not, ask for a move from the correct player.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Think modular : Tasks involved for a referee - the board keeper.

- Show the board to both players.
- Check if any of them won, if so, declare won.
- If not, ask for a move from the correct player.
- Check if the move is legal, if so, update the board.

Programming the Referee: functions

- Representing the board: A 3×3 character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order 3×3.

Think modular : Tasks involved for a referee - <u>the board keeper</u>.

- Show the board to both players.
- Check if any of them won, if so, declare won.
- If not, ask for a move from the correct player.
- Check if the move is legal, if so, update the board.
- Keep doing this until board is full or somebody wins.

Four Functions:

We will do this using four functions:

- `showconfig()` : to print the current configuration of the board.

Four Functions:

We will do this using four functions:

- `showconfig()` : to print the current configuration of the board.
- `checkwin()` : to check if the current configuration of the board (available in the global array `board`) is a winning configuration for any of the players, if yes, print the appropriate message. If it is a draw, then also it can print an appropriate message.

Four Functions:

We will do this using four functions:

- `showconfig()` : to print the current configuration of the board.
- `checkwin()` : to check if the current configuration of the board (available in the global array `board`) is a winning configuration for any of the players, if yes, print the appropriate message. If it is a draw, then also it can print an appropriate message.
- `checklegal(i,j)` : to check if putting a symbol in the `i,j` the location of the board is legal or not. That is, is a symbol already there? Then the move is illegal.

Four Functions:

We will do this using four functions:

- `showconfig()` : to print the current configuration of the board.
- `checkwin()` : to check if the current configuration of the board (available in the global array `board`) is a winning configuration for any of the players, if yes, print the appropriate message. If it is a draw, then also it can print an appropriate message.
- `checklegal(i,j)` : to check if putting a symbol in the `i,j` the location of the board is legal or not. That is, is a symbol already there? Then the move is illegal.
- `putsymbol(i,j,c)` : Assuming we checked the legality of the move by the player, put down the symbol *c* (which is either 'X' or 'O') at the entry `board[i][j]`.