# CS1100 – Introduction to Programming

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 : Checking `co-primeness`

# 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 : Finding Prime Numbers in an Interval

# 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; }
```

# Reversing an Array: Using Auxiliary Array

# Reversing an Array: Using Auxiliary Array

```
#include <stdio.h>
 void print(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
}
 void reverse(int arr[], int n)
{
    int aux[n];

    for (int i = 0; i < n; i++) {
        aux[n - 1 - i] = arr[i];
    }

    for (int i = 0; i < n; i++) {
        arr[i] = aux[i];
    }
}

int main(void)
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(arr)/sizeof(arr[0]);

    reverse(arr, n);
    print(arr, n);

    return 0;
}
```

# Reversing an Array: In Place

# Reversing an Array: In Place

```
#include <stdio.h>

void print(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
}
 void reverse(int arr[], int n)
{
    for (int low = 0, high = n - 1; low < high; low++, high--)
    {
        int temp = arr[low];
        arr[low] = arr[high];
        arr[high] = temp;
    }
}

int main(void)
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(arr)/sizeof(arr[0]);

    reverse(arr, n);
    print(arr, n);

    return 0;
}
```

# 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; }
```

# 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.

# 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();
}
```

# 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 alternate turns.

# 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 alternate 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 alternate 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 alternate 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 $3 \times 3$ character array. Stores, 'X' or 'O' or Blank in each cell.

# Programming the Referee: functions

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

# Programming the Referee: functions

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

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

# Programming the Referee: functions

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

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

- Show the board to both players.

# Programming the Referee: functions

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

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.

# Programming the Referee: functions

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

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 $3 \times 3$ character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order $3 \times 3$.

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 $3 \times 3$ character array. Stores, 'X' or 'O' or Blank in each cell.
- Define it as a global character array `board[3][3]` of order $3 \times 3$.

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.
- 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]`.

# Pseudo-code of the main program

Now the main prorgam is compact and intuitive.

```
// Assume 1 and 2 are used for X and O.
p = 0
while (checkwin() returns false)
{
  showconfig();
  read the next move (i,j) of player no:(p+1)
  // note that p+1 is either 1 or 2.

  if (checklegal(i,j) == false) continue;
  putsymbol(i,j,(p+1));
  p = (p+1) % 2.
}
Print "Game Over"
```

# The prototype declarations

```
#include <stdio.h>

char board[1000][1000]; int N=3;
char player[2] = {'X','O'};

void init();
void showconfig(void);
int checkwin(void);
int checklegal(int, int);
int putsymbol(int,int,char);

int main()
{
  init();
  ....
```

# Implementing `showconfig()`

# Implementing `showconfig()`

Exercise on printing a 2-dimensional array in matrix form.

# Implementing `showconfig()`

Exercise on printing a 2-dimensional array in matrix form.

```
void showconfig()
{
  printf("\n-------------\n");
  for (int i=0; i<N; i++)
  {
    for (int j=0; j<N; j++)
      printf("| %c ",board[i][j]);
    printf("|\n-------------\n");
  }
}
```

# Implementing `checkwin()` : The naive way

**Idea 1 :** `checkwin` : is a close cousin of the *character grid question*.

# Implementing `checkwin()` : The naive way

**Idea 1 :** `checkwin` : is a close cousin of the *character grid question*.

Recall character grid question : *Given a character grid, and a string s, check if the rows, columns or diagonals of the grid that contain s.*

# Implementing `checkwin()` : The naive way

**Idea 1 :** `checkwin` : is a close cousin of the *character grid question*.

Recall character grid question : *Given a character grid, and a string s, check if the rows, columns or diagonals of the grid that contain s.*

- Let the `board[2][2]` be the character grid.
- Do the character search with `s = XXX` to determine if X-player wins.
- Do the character search with `s = OOO` to determine if O-player wins.

So we can reuse that code.

# Implementing `checkwin()`

# A better "modular" design for `checkwin()`

**Idea 2** : Think Modular !

# A better "modular" design for `checkwin()`

**Idea 2** : Think Modular !

New function `checkwindir(int dir, char player)` : checks the winning configuration for `player` ('X'/'O') in the direction (1/2/3/4 - representing horiz/vert/diag/revdiag).

# A better "modular" design for `checkwin()`

**Idea 2** : Think Modular !

New function `checkwindir(int dir, char player)` : checks the winning configuration for `player` ('X'/'O') in the direction (1/2/3/4 - representing horiz/vert/diag/revdiag).

**Pseudocode** for `checkwindir(dir,player)`

- for i=1 to N
- for j=1 to N
  - If `dir = 1` all checks should be `board[i][j] != 'X'`.
  - If `dir = 2` all checks should be `board[j][i] != 'X'`.
  - If `dir = 3` all checks should be `board[j][j] != 'X'`.
  - If `dir = 4` all checks should be `board[j][N-j-1] != 'X'`.
- If any check fails, then try next $i$. If all succeeds for the full run of the j-loop, then declare WINNING.

# A better "modular" design for `checkwin()`

# Two more functions to define

- `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.

# Two more functions to define

- `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]`.