Basic Elements in C/C++

DCE

DEPT. OF COMPUTER ENGINEERING

Introduction

- A computer **program is a set of instructions** used to operate a computer to produce a specific result
- Writing computer programs is called computer programming
- The languages used to create computer programs are called **programming languages**

Programing Language Generation

- Machine languages are the lowest level of computer languages. **Programs written in machine language consist of 1s and 0s**

- Assembly languages perform the same tasks as machine languages, but use **symbolic names for opcodes and operands instead of 1s and 0s**
  - **ADD** A,B

- **High level programming languages** create computer programs using instructions that much easier to understand

Compiler

- Programs in a **high-level languages must be translated into a low level language** using a program called a **compiler**

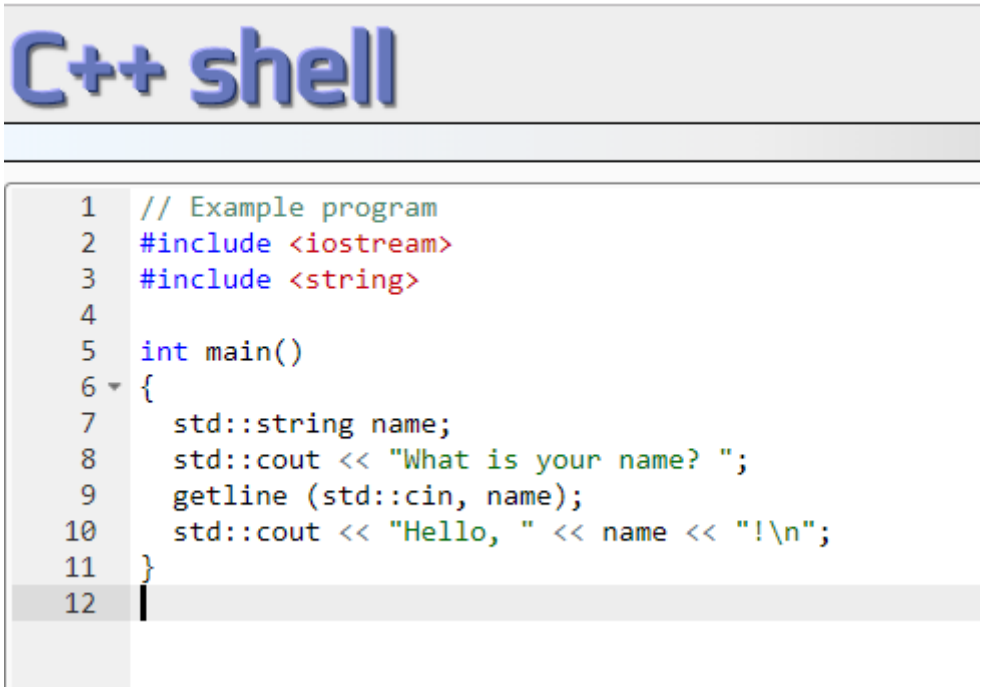
- **Compiler:** Translate all the instructions and then, execute them
- **Interpreter:** Translate and Execute one by one instruction

Programming IDE

- IDE = Integrated Development Environment
- **Programming Editor**: An environment for editing a program

- Online IDE: http://cpp.sh/

C++ Shell


C++ shell
1 // Example program
2 #include <iostream>
3 #include <string>
4
5 int main()
6 {
7 std::string name;
8 std::cout << "What is your name? ";
9 getline (std::cin, name);
10 std::cout << "Hello, " << name << "!\n";
11 }
12
Short URL: cpp.sh/
options
compilation
execution
What is your name? NHAN
Hello, NHAN!

The main() function

- The main() function is a special function that **runs automatically when a program first executes**

- All C++ programs must include one main() function. All other functions in a C++ program are executed from the main().

- The keyword before the main, e.g. **int** main() is called a function header line, or the **return of the main in integer number**.

Output: cout object

- The **cout object** is an output object that sends data given to it to the standard output display device

- To send a message to the **cout object**, you use the following pattern:
  - cout << "Hello World"**;**

- **printf** is also an instruction to print something to the screen:
  - printf("Hello World")**;**

- In C, only **printf** is supported

A Simple Program

C++ shell

```
1   // Example program
2   #include <iostream>
3
4   int main()
5   {
6
7     std::cout << "Hello World";
8
9   }
10
```

Header file

Namespace

- #include is not an instruction, **it is a directive**

Data Type and Variable

- int a;
- int a = 10;
- 2 or 4 bytes

- float b;
- float b = 2.5
- 4 bytes

Operators

- Arithmetic operators are used to perform mathematical calculations, such as addition, subtraction, multiplication, and division.

Operator	Description
+	Add two operands
-	Subtracts one operand from another operand
*	Multiplies one operand by another operand
/	Divides one operand by another operand
%	Divides two operands and returns the remainder

- A simple arithmetic expression consists of an arithmetic operator connecting two operands in the form:

  operand operator operand

Branch and Loop Statements in C

DCE

DEPT. OF COMPUTER ENGINEERING

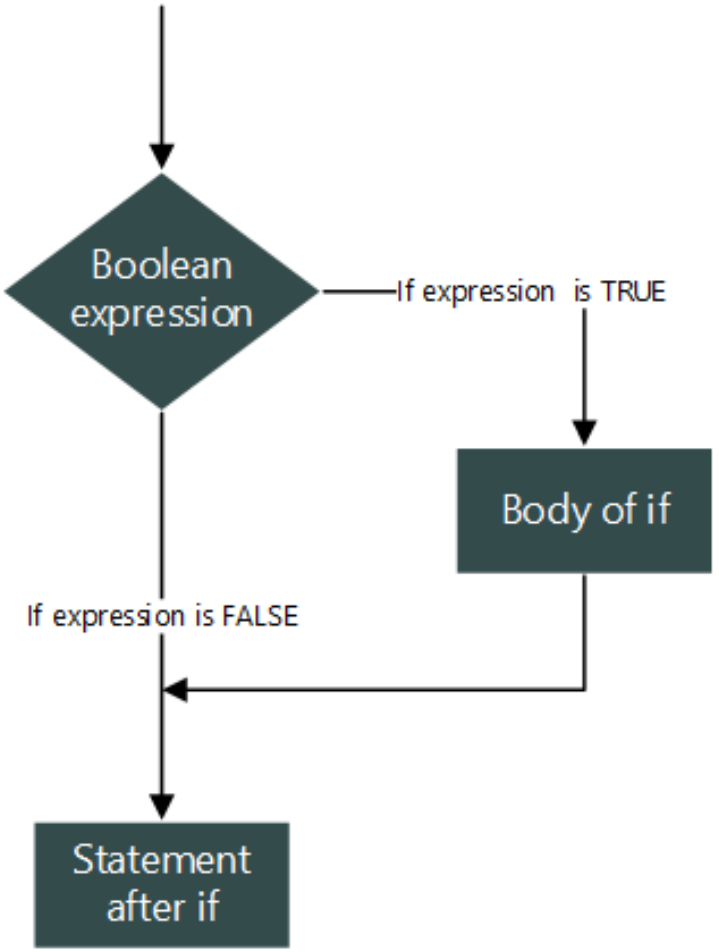
Branch Statement

- Branch statement allows us to select an action based on some conditions
- For example: if user inputs **valid account number and pin**, then allow money withdrawal, **otherwise**, alert a message
- **If statement** works like *"If condition is met, then execute the task"*.

Simple IF

```
if(boolean_expression) {
        // body of if
 }
```

- *Boolean_expression* returns true or false
- Value **zero is false**, others are true



Example

```c
#include <stdio.h>

int main()
{
    /* Variable declaration to store age */
    int age;

    /* Input age from user */
    printf("Enter your age: ");
    scanf("%d", &age);

    /* Use relational operator to check age */
    if(age >= 18)
    {
        /* If age is greater than or equal 18 years */
        printf("You are over 18!!!");
    }

    return 0;
}
```

Single and Multiple Statements

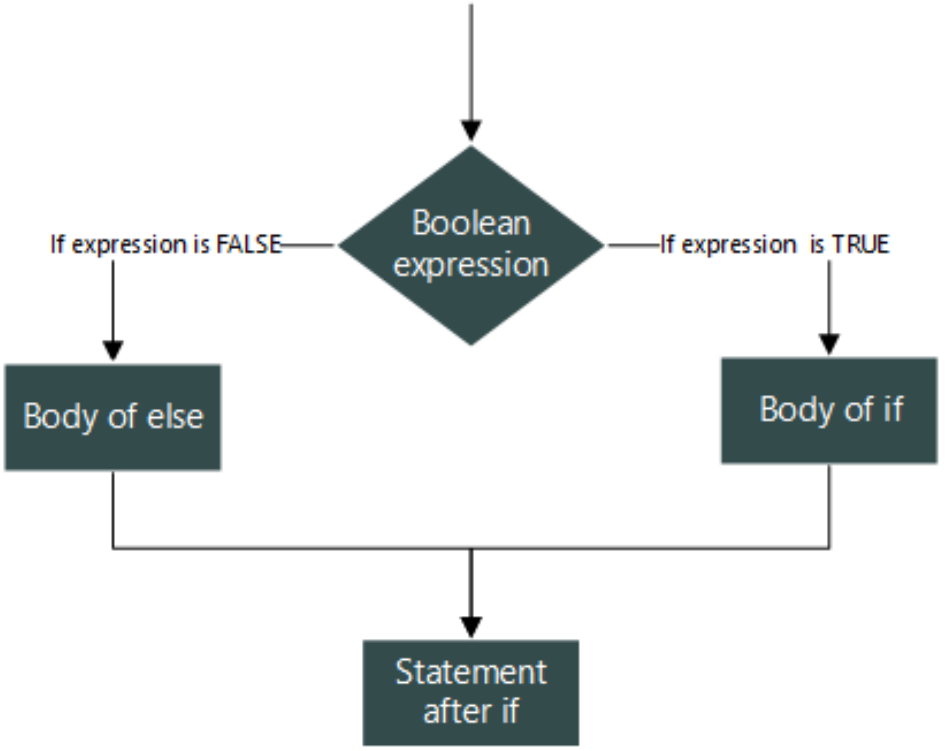
- Single statement : { and } are optional
- Multiple statements : { and } are mandatory

```
if(boolean_expression)
// Single statement inside if
```

```
if(boolean_expression){
    // Statement 1
    // Statement 2
    ...
    ...
    // Statement n
}
```

If Else Statement

```
if(boolean_expression){
    // Body of if

}
else{
    // Body of else

}
```



Exercise

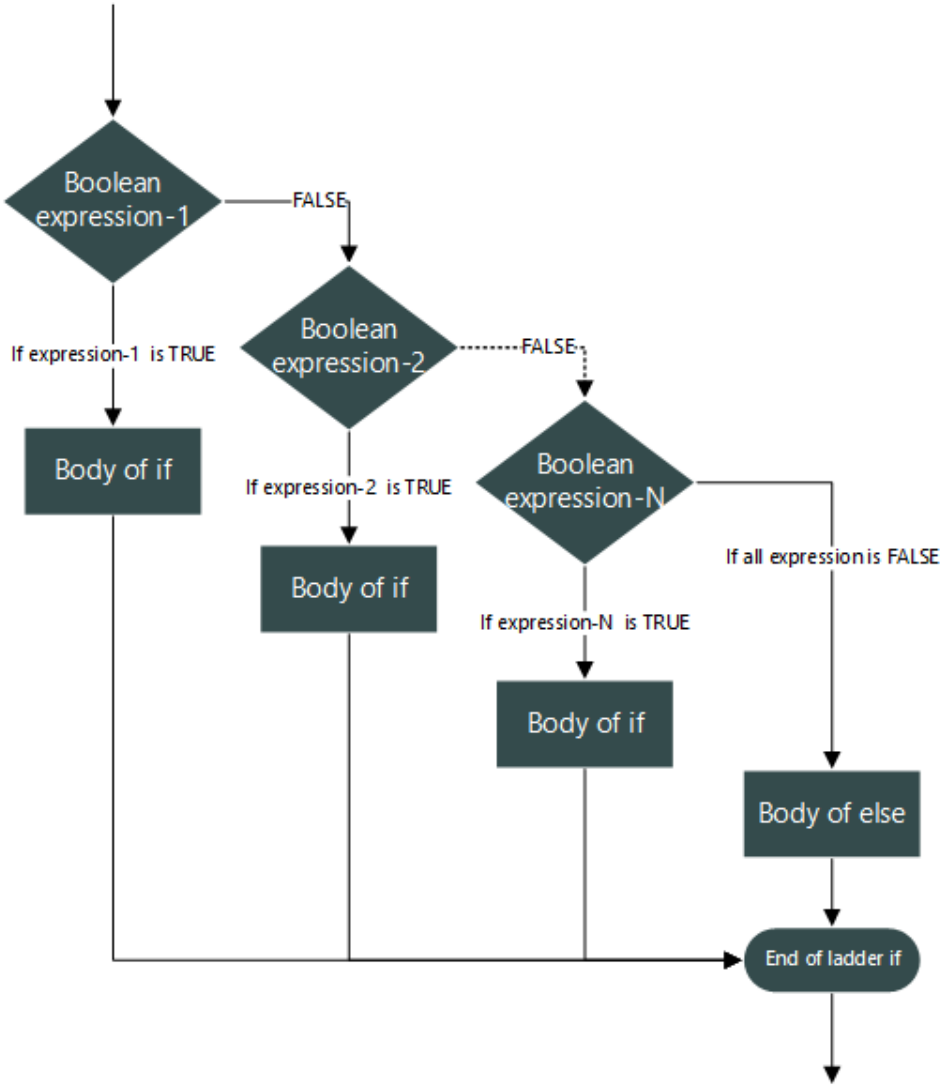
- For a given 3 Integer numbers, write a short program to find the maximum number

- For a given 4 Integer numbers, write a short program to find the minimum number

Multiple conditions

```
if (boolean_expression_1){

}
else if (boolean_expression_2){

}
else if (boolean_expression_n){

}
else{

}
```

Execution Flow



- Conditions are **checked one by one**
- **When one condition is true, the if is stopped**

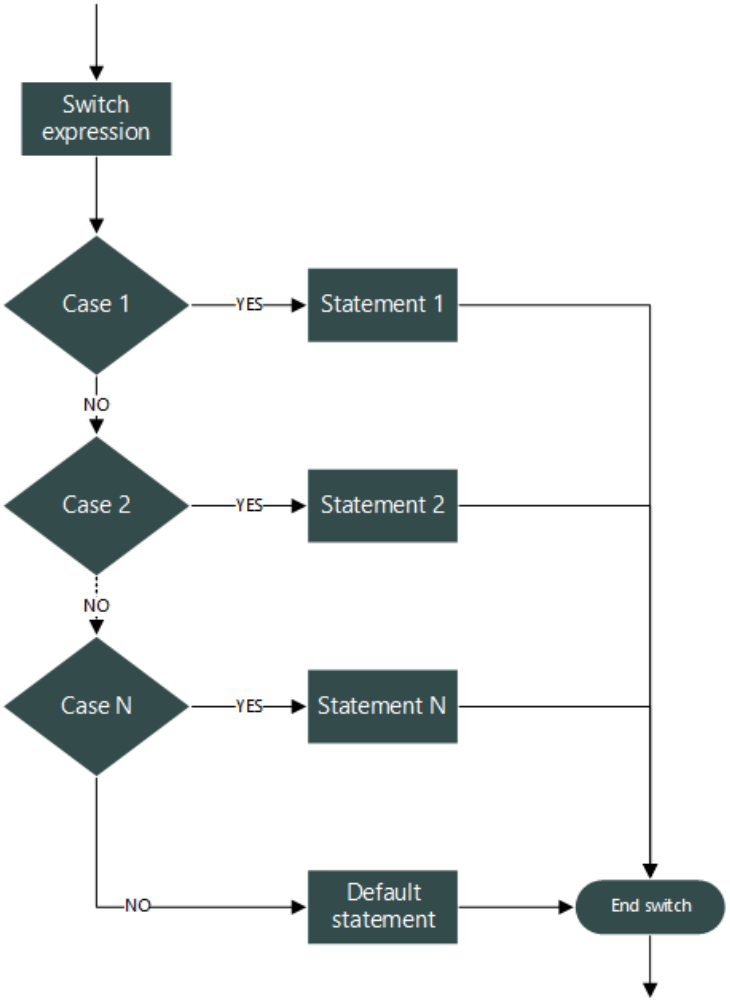
Nested IF

```
if (boolean_expression_1){
     if(nested_expression_1){

     }
     else{

     }

}
else{
     if(nested_expression_2){

     }
     else{

     }
}
```

Switch case

```
switch(expression){
    case 1:
        /* Statement/s */
        break;
    case 2:
        /* Statement/s */
        break;
    case n:
        /* Statement/s */
        break;
    default:
        /* Statement/s */
}
```

Flow chart of switch case



- Expression inside switch must evaluate to integer, character or enumeration constant
- switch…case only works with **integral, character** or *enumeration constant*
- The execution **is jumped directly to the right condition** instead of checking one by one as if else

Example

```
int num = 2;

switch(num)
{
    case 1: printf("I am One");
        break;
    case 2: printf("I am Two");
        break;
    case 3: printf("I am Three");
        break;
    default: printf("I am not 1, 2 and 3.");
}
```

IF ELSE and SWITCH CASE

- Use if...else...if statement when:
  - There are conditions instead of list of choices.
  - There are few number of conditions.

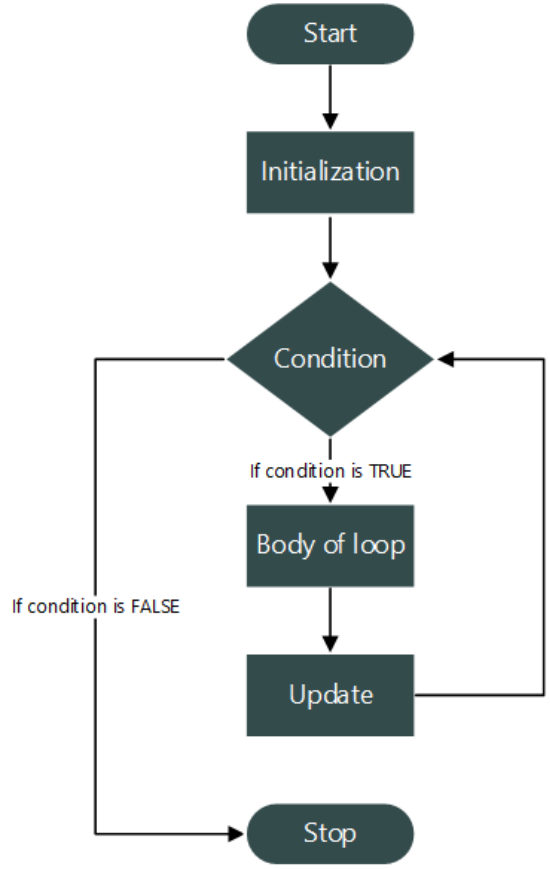
- Use switch...case when:
  - There is a list of choices, from which you need to take decision.
  - Choices are in the form of integer, character or enumeration constant

- Further reading: enumerable declaration, state machine, deterministic finite automata

Looping Statement: FOR

```
for(initialization ; condition ; update){
    // Body of for loop
}
```

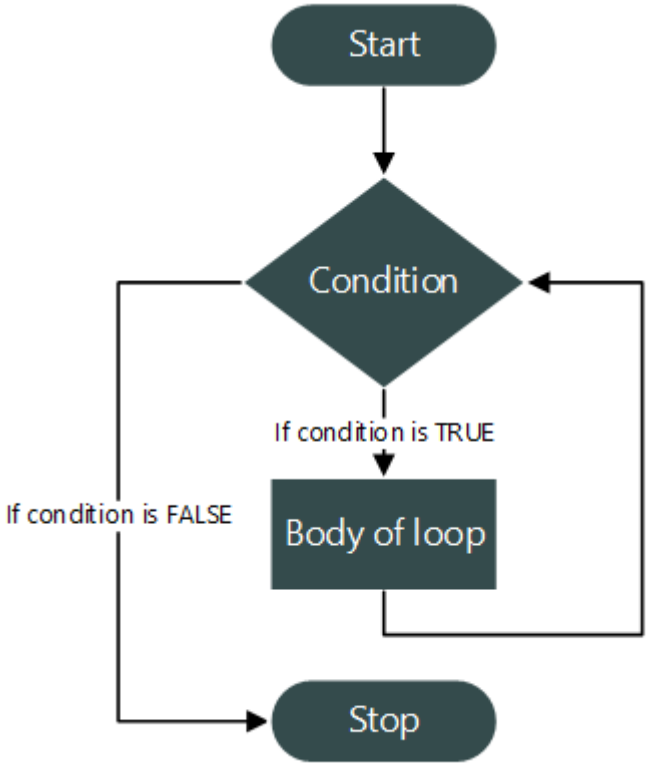
```
#include <stdio.h>

int main(){
    int count;

    for(count=1; count<=10; count++){
        /* Print current value of count */
        printf("%d ", count);
    }

    return 0;
}
```



Looping Statement: WHILE

```
while(condition) {
// Body of while loop
}
```

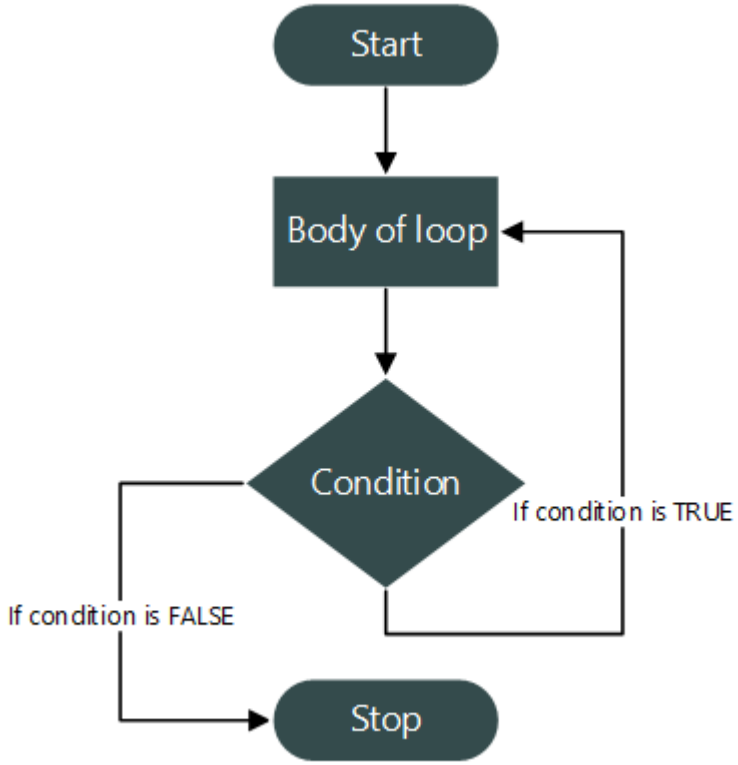
```c
#include <stdio.h>

int main() {
    int n = 1;

    while(n <= 10) {
        /* Body of loop */
        printf("%d ", n);

        n++; // n = n + 1
    }
    return 0;
}
```



Looping Statement: DO WHILE

```
do{
    // Body of do while loop
} while (condition);
```

```
#include <stdio.h>

int main(){
    int n=1;
    do{
        /* Body of loop */
        printf("%d ", n);

        /* Update loop counter variable */
        n++;

    } while(n <= 10); /* Loop condition */
    return 0;
}
```



Nested Loop

```c
#include <stdio.h>

int main(){
    /* Loop counter variable declaration */
    int i, j;

    for(i=1; i<=10; i++){

        for(j=1; j<=5; j++){
            printf("%d\t", (i*j));
        }

        /* Print a new line */
        printf("\n");
    }

    return 0;
}
```

Break statement in Loop

```
do
{
    // body of do while loop

    if(condition)
        break;
} while(condition);

// statements below do while loop
```

```
for(initialization; condition; update)
{
    // body of loop

    if(condition)
        break;
}

// statements below for loop
```

Example: Check a Prime Number

```c
#include <stdio.h>

int main()
{
    /* Variable declarations */
    int num, isPrime, i;

    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    /* Initially assume that the number is prime */
    isPrime = 1;
    for(i=2; i<num; i++){

        if(num % i == 0){
            isPrime = 0;
            break;
        }
    }

    return 0;
}
```

Question

- Why we need a break statement instead of putting it in the condition statement of the loop

Continue in Loop

```
for(initialization; condition; update)
{
    // body of loop

    if(condition)
        continue;
}

// statements below for loop
```

```
while(condition)
{
    // body of while loop

    if(condition)
        continue;
}

// statements below while loop
```

```
do
{
    // body of do while loop

    if(condition)
        continue;
} while(condition);

// statements below do while loop
```

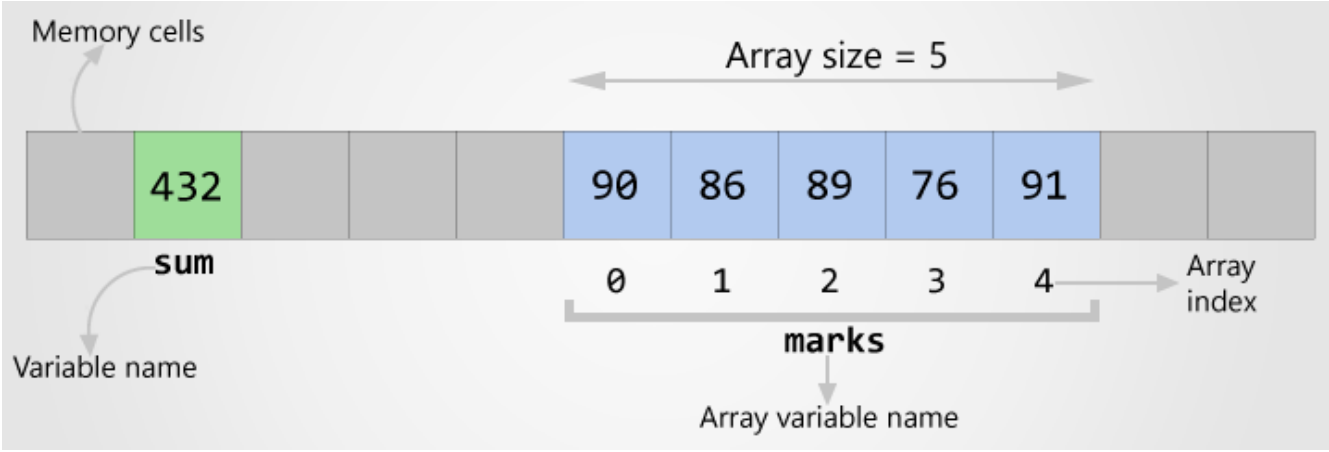
Example

```c
#include <stdio.h>

int main(){
    /* Variable declaration */
    int num;

    for(num=1; num<=100; num++){
        if(num % 2 == 1)
            continue;

        printf("%d ", num);
    }

    return 0;
}
```

- What are printed to the console?

Arrays in C – Declare, initialize and access

- **Array is a collection** - Array is a container that can hold a collection of data.
- **Array is finite** - The collection of data in array is always finite, which is determined prior to its use.
- **Array is sequential** - Array stores collection of data sequentially in memory.
- **Array contains homogeneous data** - The collection of data in array must share a same **data type**.



How to declare an array?

DATA_TYPE array_name[SIZE];

- DATA_TYPE is a valid C data type
- array_name is name given to array
- SIZE is a constant value

- int marks[5];

How to initialize an array

- int marks[5] = {90, 86, 89, 76, 91};
- int marks[] = {90, 86, 89, 76, 91};

- marks[0] = 90;
- marks[1] = 86;
- marks[4] = 91

```
int index;

sum = 0;
for(index=0; index<SIZE; index++)
{
    sum = sum + marks[index];
}
```

Exercise Online

- https://www.w3resource.com/c-programming-exercises/basic-algo/index.php
- https://codeforwin.org/2016/03/functions-programming-exercises-and-solutions-in-c.html

Exercise

- Write a program to **read and print elements** of array. It is assumed that the array has **5 elements**. The input is a text file, named **input_array.txt**. The split character in the text file is the space character.

Answer

```
int marks[5];
ifstream fin ("input_array.txt");
for(int i = 0; i<5; i++){
    fin >> marks[i];
}
```

Exercise

- Write a program to print all negative elements in an array.
- Write a C program to find sum of all array elements
- Write a C program to find maximum and minimum element in an array
- **Write a C program to find second largest element in an array.**
- Write a C program to count total number of even elements in an array.
- Write a C program to count frequency of each element in an array.
- Write a C program to print all unique elements in the array.

Exercise

- Write a C program to copy all elements from an array to another array.
- Write a C program to insert an element in an array.
- Write a C program to delete an element from an array at specified position.

Function and Recursive in C

DCE
DEPT. OF COMPUTER ENGINEERING

Introduction to Function

- A *function* is a collection of statements grouped together to do **some specific tasks**
- **printf(), scanf() and main() are functions**

- Why functions are used?
  - Divide code for all tasks into separate functions.
  - From main(), these functions are called to execute the tasks

Advantages of Functions

- **Reusability** of code. Functions once defined can be used any several times
- Function allows **modular design** of code. Modular programming leads to **better code readability, maintenance and reusability.**
- It is **easier to write programs** using functions. You can write code for separate task individually in separate function.
- Code **maintenance and debugging** is easier. In case of errors in a function, you only need to debug that particular function instead of debugging entire program.

Function Declaration

- **return_type** function_name(parameter_list);
  - **return type:** int, double, float **or void**
  - function_name: a valid C identifier
  - parameter_list: input data

Function Definition

```
return_type function_name(parameter list)
{
    // Function body
}
```

Example

```c
#include <stdio.h>

/* Addition function declaration */
int add(int num1, int num2);

int main()
{
    /* Variable declaration */
    int n1, n2, sum;

    n1 = 2; n2 = 3;
    sum = add(n1, n2);

    /* Print value of sum */
    printf("Sum = %d", sum);

    return 0;
}

int add(int num1, int num2)
{
    int s = num1 + num2;
    return s;
}
```

Arguments – Call by Value

```
                    // Formal parameters
void add(int num1, int num2) // Function definition
{
    // Function body
}


int main()
{
            // Actual parameters
    add(10, 20); // Function call

    return 0;
}
```

- Parameter value **is copied and passed** to formal parameter

Example

```c
#include <stdio.h>

void swap(int num1, int num2)
{
    int temp;

    temp = num1;
    num1 = num2;
    num2 = temp;
}

int main()
{
    int n1, n2;
    n1 = 2; n2 = 4;

    swap(n1, n2);

    return 0;
}
```

Argument – Call by Reference

```c
#include <stdio.h>


void swap(int * num1, int * num2)
{
    int temp;

    temp  = *num1;
    *num1 = *num2;
    *num2 = temp;
}


int main()
{
    int n1, n2;
    n1 = 1; n2 = 2;

    swap(&n1, &n2);

    return 0;
}
```

```
void swap(int *num1, int* num2){
        int temp = *num1;
        *num1 = *num2;
        *num2 = temp;
}
void main(){
        int n1 = 1, n2 = 2;
        swap(&n1, &n2);
        return;
}
```

```
void swap(int *num1, int* num2){
        int temp = *num1;
        *num1 = *num2;
        *num2 = temp;
}
void main(){
        int n1 = 1, n2 = 2;
        swap(&n1, &n2);
        return;
}
```

Recursion in C

```c
void recursive_function()
{
    // Some codes

    recursive_function();

    // Unreachable code
}

int main()
{
    recursive_function();
}
```

Example

```c
#include <stdio.h>

void print(int n){
    /* Print the current value of n */
    printf("%d ", n);

    if(n <= 1) {
        return;
    }

    /* Call print() function recursively with n-1 */
    print(n - 1);
}

int main(){
    print(5);

    return 0;
}
```

Exercise

- Write a C program to find cube of any number using function

- Input any number: 5
- Cube of 5 is: 125

Answer

```c
#include <stdio.h>

double cube(double num);

int main(){
    int num = 5;
    double c;

    c = cube(num);

    return 0;
}

double cube(double num){
    return (num * num * num);
}
```

- Function prototype decleration

Exercise

- Find maximum and minimum among 3 numbers using function
- Write a C program to check whether a number is prime
- Write a C program to find all prime numbers between given interval

Answer

```
int isPrime(int num) {
    int i;

    for(i=2; i<=num/2; i++){

        if(num%i == 0) {
            return 0;
        }
    }

    return 1;
}
```