Welcome to C-programming Class

do-while loop

Introduction to Do–While Loop

A do–while loop is an exit-controlled loop.

do-while loop

Introduction to Do–While Loop

A do–while loop is an exit-controlled loop.

The loop body executes at least once, even if the condition is false.

do-while loop

Introduction to Do–While Loop

A do–while loop is an exit-controlled loop.

The loop body executes at least once, even if the condition is false.

Because condition is checked after execution.

syntax of do-while loop

```
do
{
    // statements
}
while(condition);
```

syntax of do-while loop

```
do
{
    // statements
}
while(condition);
```

Important Points:

do block runs first

Condition checked after execution

Semicolon (;) is mandatory after while(condition)

Difference: While vs Do–While

While Loop	Do–While Loop
Entry-controlled	Exit-controlled
Condition checked first	Condition checked after execution
May execute 0 times	Executes at least once

Simple c example

Print numbers from 1 to 5 using do–while loop.

```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
     }
    while(i <= 5);
    return 0;
}
```

Simple c example

Print numbers from 1 to 5 using do–while loop.

```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
        i++;
    }
    while(i <= 5);
    return 0;
}
```

Algorithm:-

Algorithm to print numbers from 1 to 5 using do–while loop

1. Start
2. Declare integer variable i
3. Initialize i = 1
4. Execute:
   - Print i
   - Increment i
5. Check condition i <= 5
6. Stop

```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
        i++;
    }
    while(i <= 5);
    return 0;
}
```

Algorithm:-

Algorithm to print numbers from 1 to 5 using do–while loop

1. Start
2. Declare integer variable i
3. Initialize i = 1
4. Execute:
   - Print i
   - Increment i
5. Check condition i <= 5
6. If true, repeat Step 4
7. Stop

Print numbers from 1 to 5 using do–while loop.

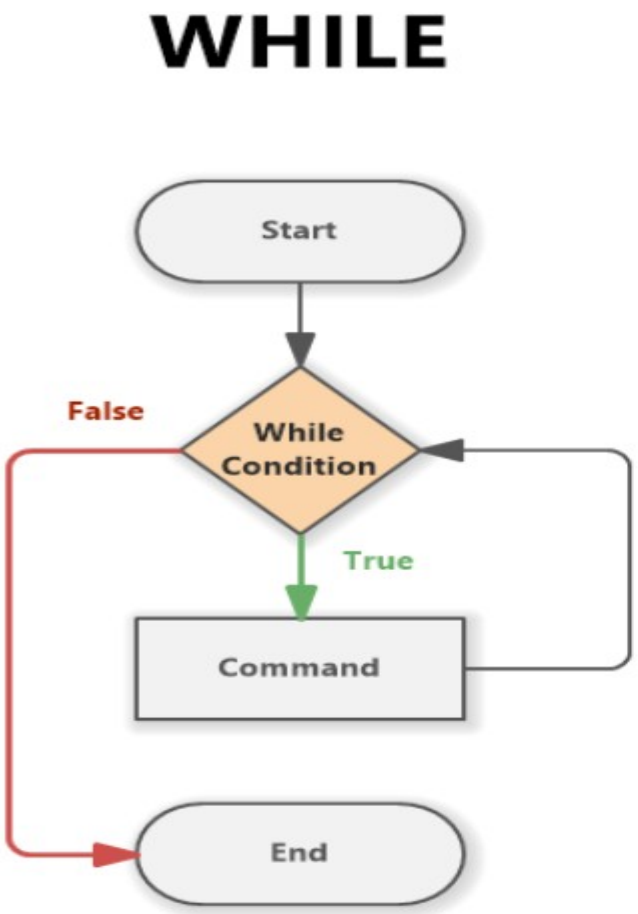
```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
        i++;
    }
    while(i <= 5);
    return 0;
}
```

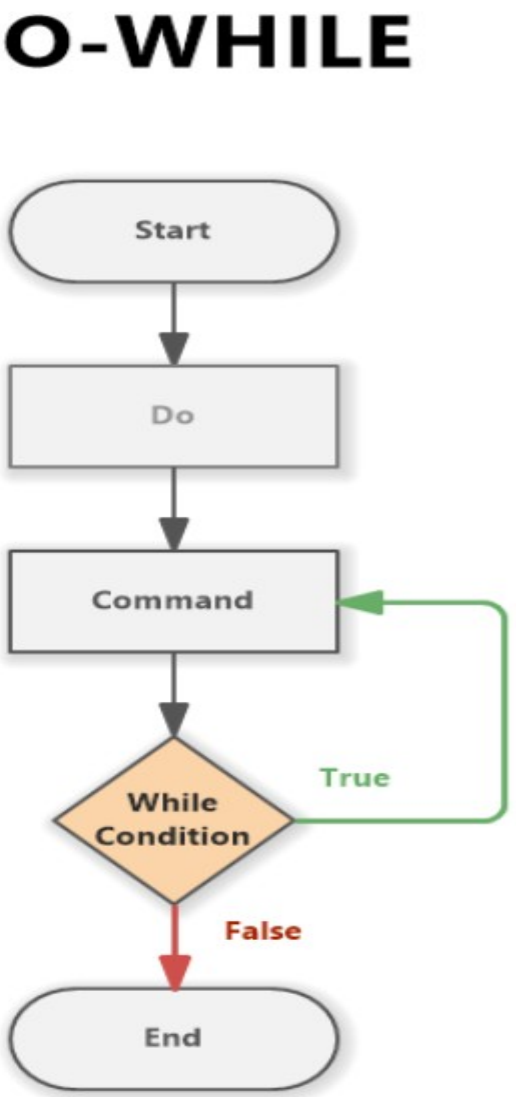
Pseudo-code

```
BEGIN
SET i ← 1
DO
PRINT i
i ← i + 1
WHILE (i ≤ 5)
END
```

```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
        i++;
    }
    while(i <= 5);
    return 0;
}
```

Flowchart

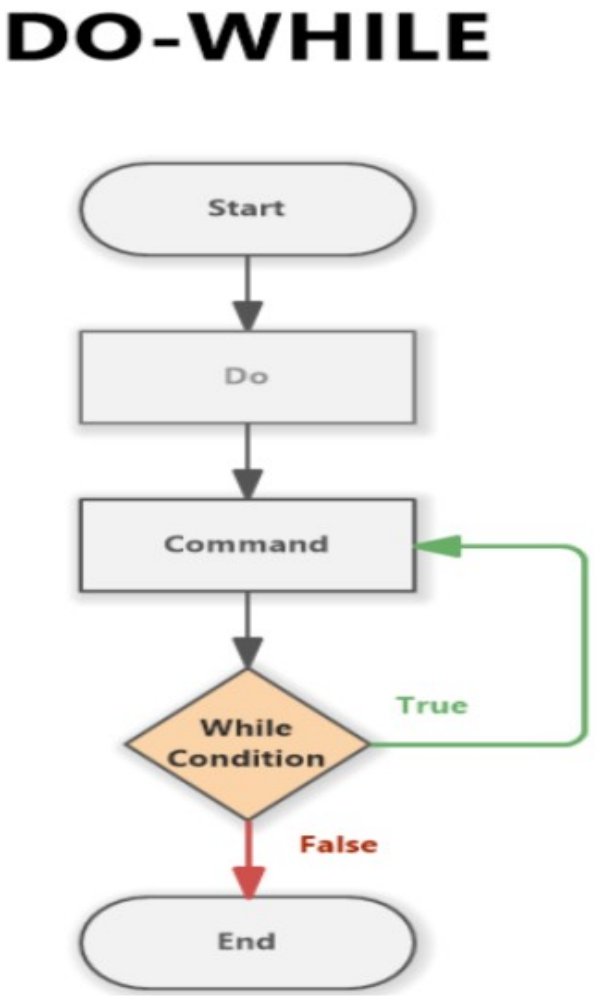

WHILE
Start
False
While Condition
True
Command
End


DO-WHILE
Start
Do
Command
While Condition
True
False
End

Flowchart

Flow Explanation:

- Start
- Initialize i = 1
- Execute loop body (Print + Increment)
- Check condition (i ≤ 5?)
- If Yes → Repeat
- If No → End



dry-run

Iteration	i Before Print	Output	i After Increment	Condition (i ≤ 5)
1	1	1	2	True
2	2	2	3	True
3	3	3	4	True
4	4	4	5	True
5	5	5	6	False (Stop)

output??

```
int i = 10;

do
{
    printf("%d", i);
}
while(i < 5);
```

FOR LOOP

```
for(initialization; condition;
increment)
{
   // statements
}
```

output??

i=1 → True → Print 1
i=2 → True → Print 2
i=3 → True → Print 3
i=4 → True → Print 4
i=5 → True → Print 5
i=6 → False → Stop

```
for(int i=1; i<=5; i++)
{
   printf("%d", i);
}
```

COMMON ERRORS

- Missing increment
- Wrong condition
- Infinite loops
- Missing semicolon in do–while

QUESTIONS

Define a loop in C.
What is an entry-controlled loop?
What is an exit-controlled loop?
Write syntax of for loop.
Write syntax of while loop.
Write syntax of do–while loop.
Which loop executes at least once?
What is an infinite loop?
What is the difference between while and do–while?
Why is a semicolon required in do–while loop?

BREAK STATEMENT

Definition

The break statement is used to terminate the loop or switch statement immediately.

Control moves to the next statement after the loop/switch.

SYNTAX: break;

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            break;
        printf("%d ", i);
    }
    return 0;
}
```

SYNTAX: break;

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            break;
        printf("%d ", i);
    }
    return 0;
}       output:- 1234
```

SYNTAX: break in switch

```
switch(choice)
{
    case 1:
        printf("Add");
        break;
    case 2:
        printf("Subtract");
        break;
}
```

Continue statement

The continue statement skips the current iteration and moves to the next iteration of the loop.It does NOT terminate the loop.

Continue statement

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 5; i++)
    {
        if(i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

Continue statement

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 5; i++)
    {
        if(i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
} output:- 1245
```

Difference between beak and continue

Feature	break	continue
Terminates loop	Yes	No
Skips iteration	No	Yes
Used in switch	Yes	No

GOTO STATEMENT

Definition

The goto statement transfers control to a labeled statement within the same function.

GOTO STATEMENT:-

```
goto label;
...
label:
   statement;
```

goto:-

```
#include <stdio.h>
int main()
{
    int i = 1;

start:
    printf("%d ", i);
    i++;

    if(i <= 5)
        goto start;

    return 0;
}
```

goto statement:-

```
#include <stdio.h>
int main()
{
    int i = 1;

start:
    printf("%d ", i);
    i++;

    if(i <= 5)
        goto start;

    return 0;
}                   output :- 1 2 3 4 5
```

Array:- collection of similar data types

An array is a collection of similar data types stored in continuous memory locations.

Array:- collection of similar data types

Instead of writing: int m1 = 80, m2 = 75, m3 = 90

We use:

int marks[3] = {80, 75, 90}

Array

Concept of Array:- An array is a collection of similar data types stored in continuous memory locations.

Instead of writing: int m1 = 80, m2 = 75, m3 = 90
We use: int marks[3] = {80, 75, 90}

Memory Concept

Index:	0	1	2
Value:	80	75	90

Index always starts from 0.

1 D Array

One-Dimensional Array (1D Array)

Declaration
```
int arr[5];
```

Initialization
```
int arr[5] = {1, 2, 3, 4, 5};
```

Accessing Elements
```
printf("%d", arr[0]);
```

1 D Array C Example:-

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int i;
    for(i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

1 D Array C Apllications in AIML and ECE

AIML relevance: Used to store dataset features

ECE relevance: Used to store signal samples

2 D Array

Used for matrices (rows & columns).

2 D Array

Used for matrices (rows & columns).

Declaration
int matrix[2][3];

2 D Array

Used for matrices (rows & columns).

Declaration

```
int matrix[2][3];
```

Initialization

```
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

2 D Array

Used for matrices (rows & columns).

Declaration

```
int matrix[2][3];
```

Initialization

```
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Accessing

```
printf("%d", matrix[1][2]);
```

2 D Array Example:- addition of an array

```
#include <stdio.h>
int main() {
    int a[2][2] = {{1,2},{3,4}};
    int b[2][2] = {{5,6},{7,8}};
    int c[2][2];
    int i,j;
    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    printf("Result Matrix:\n");
    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}
```

2 D Array Applications

AIML:
Used in neural network weight matrices

ECE:
Used in image processing

Multi-Dimensional Arrays

More than 2 dimensions

int arr[2][3][4];

Used in:

AI image datasets (RGB images)

3D signal processing

User-Defined Functions-definition

User Defined Functions:- Block of code written by programmer.

User-Defined Functions-Syntax

User Defined Functions:- Block of code written by programmer.

Syntax:-

```
return_type  function_name(parameters) {
    // body
}
```

User-Defined Functions- Example

Syntax

```
return_type  function_name(parameters) {
    // body
}
```

```
#include <stdio.h>
int add(int a, int b) {
    return a + b;
}
int main() {
    int result = add(5, 3);
    printf("Sum = %d", result);
    return 0;
}
```

Built-in Functions or user-defined functions

Provided by C library.

Examples:

```
sqrt(25);    // math.h
strlen("AI"); // string.h
```

Built-in Functions or user-defined functions

Provided by C library.

Examples:

```
sqrt(25);    // math.h
strlen("AI"); // string.h
```

Storage Classes

A storage class in C defines:

Scope → Where the variable can be accessed
Lifetime → How long the variable exists
Default initial value
Memory location

Storage Classes

Types of Storage Classes in C
auto
register
static
extern

Storage Classes

Storage Class	Scope	Lifetime
auto	Local	Inside block
static	Local	Entire program
extern	Global	Entire program
register	Local	CPU register

Storage Classes

AUTO Storage Class

- Default storage class for local variables
- Stored in main memory (RAM)

Scope:

Inside the block/function

Lifetime:

Until function execution ends

Default value: Garbage value

Storage Classes

Example

```
#include <stdio.h>
int main() {
    auto int x = 10;   // auto is optional
    printf("%d", x);
    return 0;
}
```

Storage Classes

Example

```
#include <stdio.h>
int main() {
    auto int x = 10;   // auto is optional
    printf("%d", x);
    return 0;
}
```

Note: Writing auto is optional because all local variables are auto by default.

Storage Classes

REGISTER Storage Class
Stored in CPU register (if available)
Used for fast access

Scope: Inside the block

Lifetime: Till function execution

Default value: Garbage value

Storage Classes

Example

```
#include <stdio.h>
int main() {
    register int i;
    for(i = 0; i < 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Storage Classes

Example

```
#include <stdio.h>
int main() {
    register int i;
    for(i = 0; i < 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Limitation:

You cannot use &i because register variables may not have memory address.

Storage Classes

Example: Without Static

```
#include <stdio.h>
void counter() {
    int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
    return 0;
}
```

Storage Classes

Example: Without Static

```
#include <stdio.h>
void counter() {
    int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
    return 0;
}
```

Output:

```
1
1
1
```

Storage Classes

STATIC Storage Class
Retains value between function calls
Stored in data segment
Scope:
Local to function (if declared inside function)

Lifetime: Entire program

Default value: 0

Storage Classes: C program as example

```
#include <stdio.h>
void counter() {
    static int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
}
```

Storage Classes: C program as example

```
#include <stdio.h>
void counter() {
    static int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
}
```

Output:

1

2

3

Call by Value

Copy of variable is passed.

```
void change(int x) {
    x = 100;
}

int main() {
    int a = 10;
    change(a);
    printf("%d", a);   // 10
}
```

Call by Reference (Using Pointers)

Original value changes.

```
void change(int *x) {
    *x = 100;
}
int main() {
    int a = 10;
    change(&a);
    printf("%d", a);   // 100
}
```

Passing Array to Function

```
void display(int arr[], int n) {
    int i;
    for(i=0;i<n;i++)
        printf("%d ", arr[i]);
}
```

Strings in C

Array of characters ending with \0

char name[] = "AIML";

Memory:

A I M L \0

Inputting Strings

```
char name[20];
scanf("%s", name);
```

Better method:

```
fgets(name, 20, stdin);
```

Character Library Functions

Include:

```
#include <ctype.h>
```

Examples:

```
toupper('a');  // 'A'
isdigit('5');  // 1
```

String Handling Functions

Include:

```
#include <string.h>
```

Function	Work
strlen()	Length
strcpy()	Copy
strcat()	Concatenate
strcmp()	Compare

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
    char a[20] = "AI";
    char b[] = "ML";

    strcat(a, b);
    printf("%s", a);   // AIML
}
```

Recursion:-

Recursion is a process in which a function calls itself until a stopping condition is reached. It consists of two parts: base condition and recursive call.

Factorial of a number.

```
int fact(int n)
{
    if(n==0)
        return 1;
    else
        return n * fact(n-1);
}
```

Recursion:-

```
#include <stdio.h>

int factorial(int n)
{
   if(n == 0)          // Base Case
      return 1;
   else
      return n * factorial(n - 1);   // Recursive Call
}

int main()
{
   int num = 5;
   int result;

   result = factorial(num);

   printf("Factorial = %d", result);

   return 0;
}
```

Recursion:-

Recursion is a process in which a function calls itself until a stopping condition is reached.

It consists of two parts: base condition and recursive call.

```
#include <stdio.h>
#include <string.h>

int main() {
    char a[20] = "AI";
    char b[] = "ML";
```

Questions

ARRAYS

Define an array. Why is it required in C?
What is the difference between 1D and 2D array?
What happens if array index goes out of bounds?
Write syntax of 2D array declaration.
What is a multidimensional array?

Questions

FUNCTIONS

What is a user-defined function?
Differentiate between call by value and call by reference.
Define recursion.
What is the purpose of static storage class?
What is the difference between local and global variables?

Questions

STRINGS

How are strings stored in C?
What is the role of '\0' in strings?
Write syntax of strlen() function.
Difference between gets() and fgets().
What is strcmp() used for?

Questions

FUNCTIONS

What is a user-defined function?
Differentiate between call by value and call by reference.
Define recursion.
What is the purpose of static storage class?
What is the difference between local and global variables?

God Bless you all

Thanks ...