OCS752 INTRODUCTION TO C PROGRAMMING

DECISION MAKING
&
BRANCHING

INTRODUCTION

- Instruction of a programs are executed either
  - Sequential manner
  - Branching
- C language supports the following decision making statements.
  - if statement
  - switch statement
  - conditional operator
  - goto statement

if statement

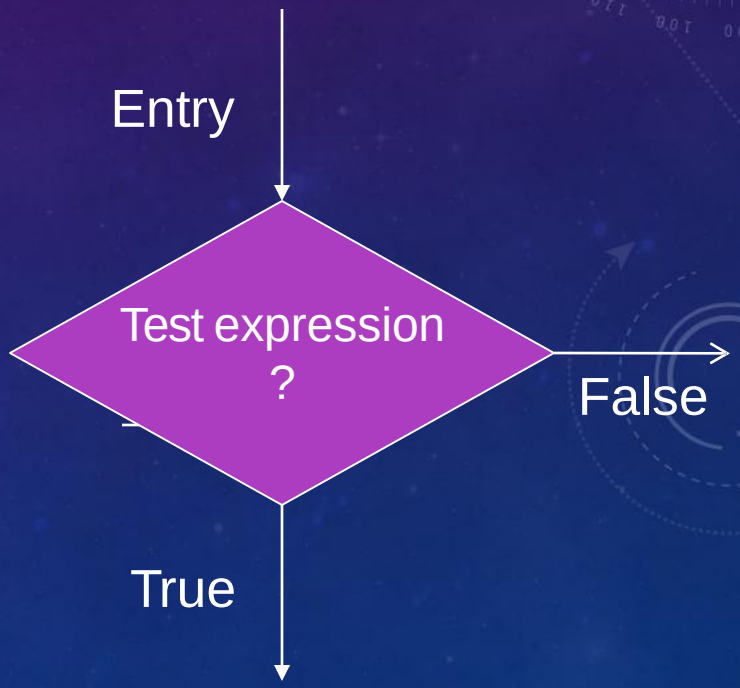
- It is used to control flow of execution of statement.
- It is two-way decision statement and is used in conjuction with an expression

Syntax- if (condition)

{ statement 1;

……..

}



Ex: if (age is more than 55)

Person is retired

DIFFERENT FORMS OF IF STATEMENT

simple if if-else nested if-else else if ladder

Simple if statement

Syntax- if (Condition)

{

statement block;

}

statement x ;

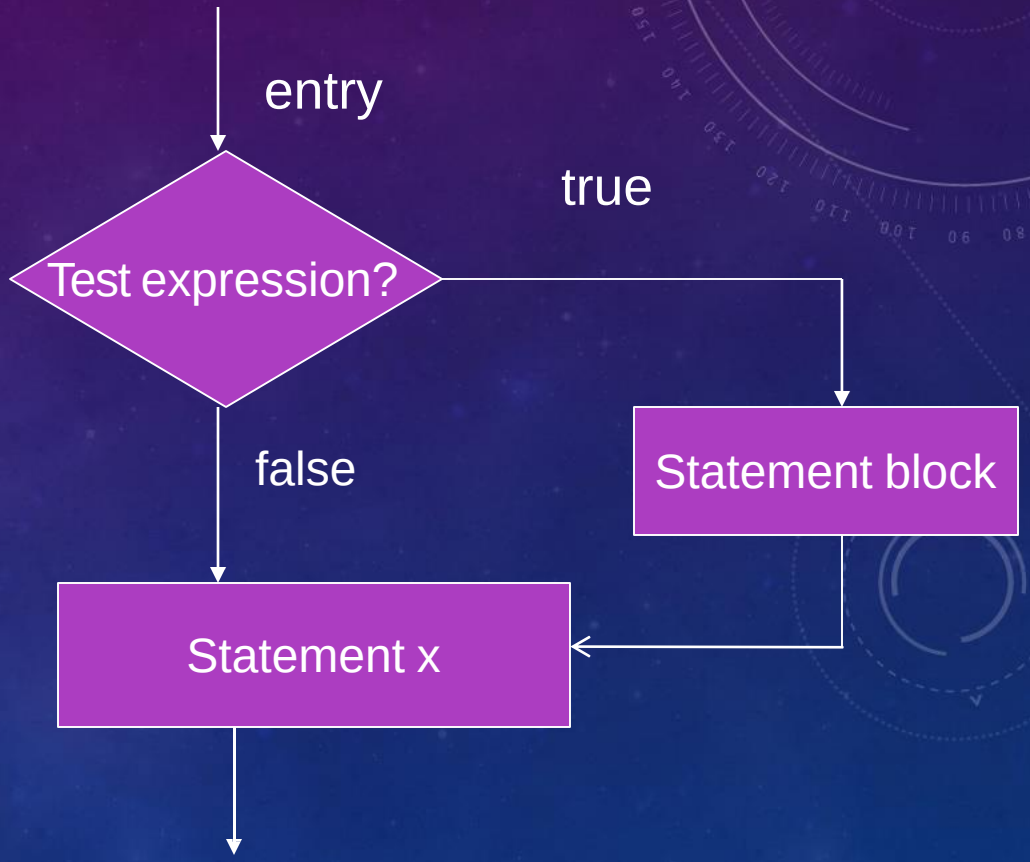
Ex: if (category == sports)

{

marks=marks+bonus;

}

printf (“%f”,marks);



Program of simple if statement

```
main()
{
int a,b,c,d;
float ratio;
printf("\n enter four integer value");
scanf("%d %d %d",&a,&b,&c);
 if(c-d !=0)
  {
  ratio= (float) (a+b)/(float)(c-d);
   printf("Ratio= %f", ratio);
  }
}
```

Output- enter four integer values

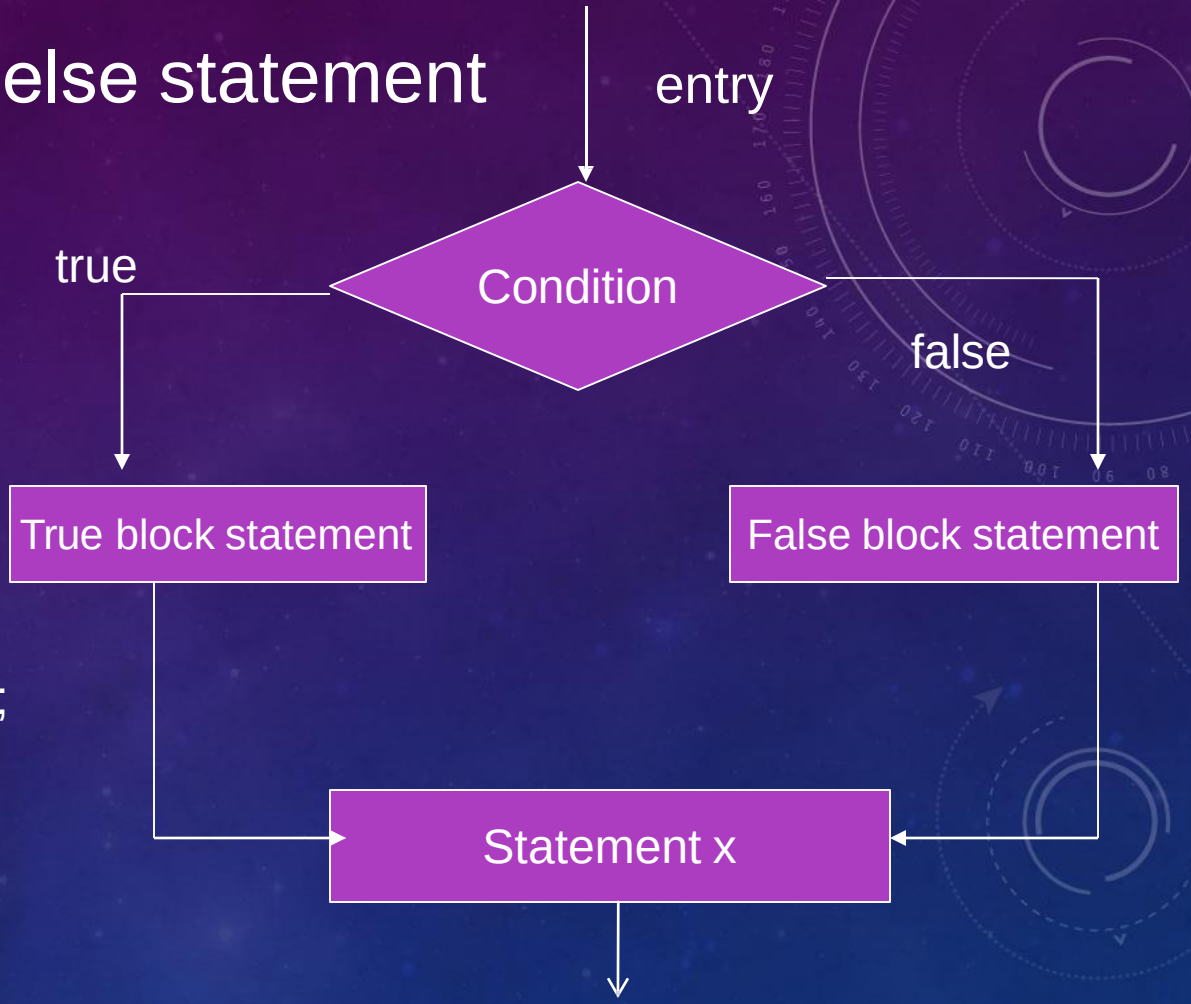
12 23 34 45

Ratio= -3.181818

The if…else statement

Syntax- if(Condition)

```
  {
     true block statement;
   }
 else
  {
     false block statement;
   }
 statement x;
```



Ex-: if (code== 1)

```
      boy=boy+  1;
     if ( code== 2)
     girl=girl+1;
```

```
if (code==1)
    boy=boy+1;
else
        girl=girl+1;
```

Program for if…else statement

```
main()
{
 int a;
 printf(“Enter an integer”);
 scanf(“%d”,&a);
 if(a%2==0)
  printf(“ %d is even number”,a);
    else
   printf(“%d is an odd number”,a);
}
```

output- Enter an integer

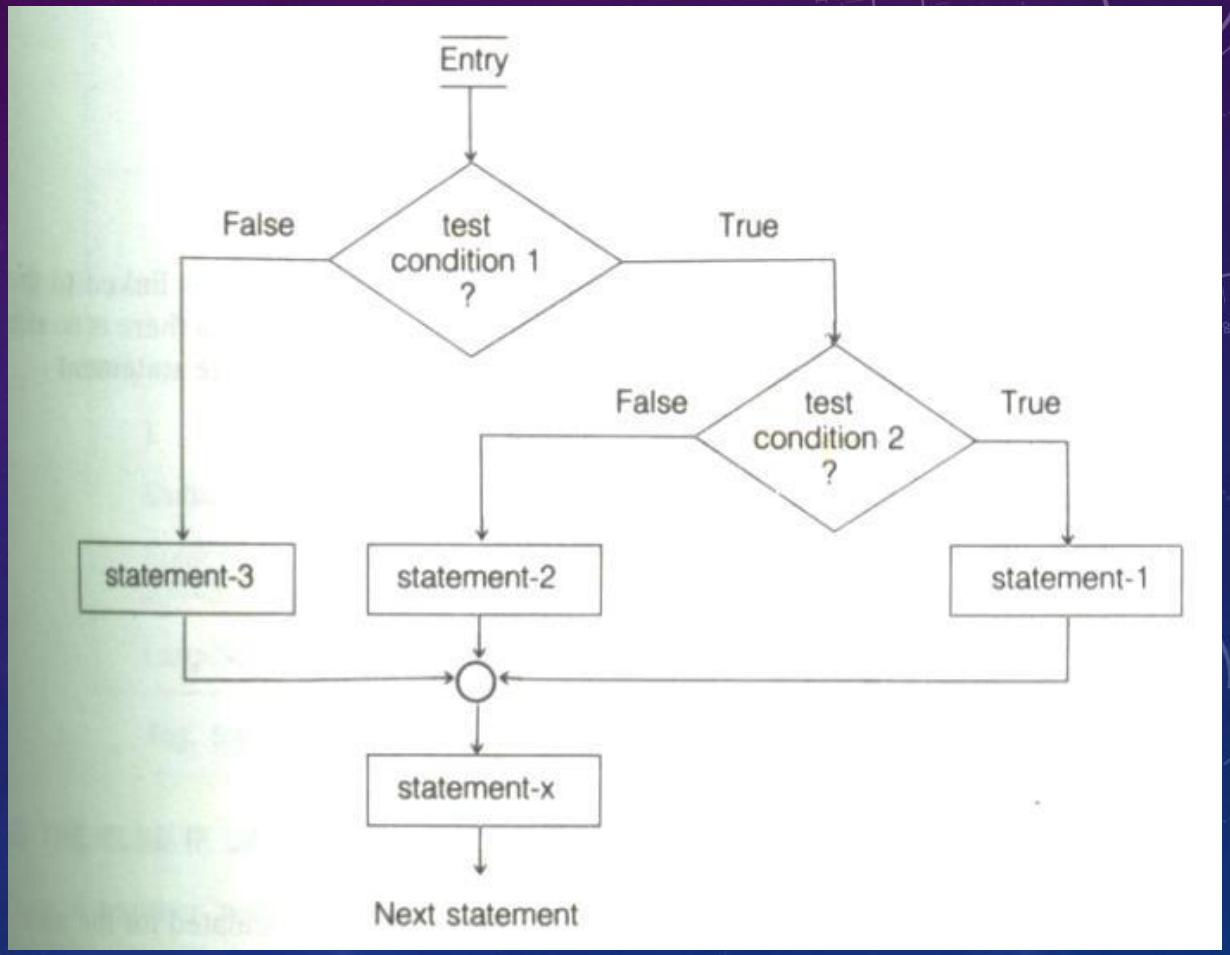
46

46 is an even number

NESTING OF IF…ELSE STATEMENT

Syntax-

```
if( test condition 1)
    {
    if (test condition 2)
       {
       statement 1;
       }
    else
       {
       statement 2;
       }
    }
else
   {
    statement 3;
   }
statement x  ;
```



PROGRAM FOR NESTED IF...ELSE STATEMENT

```
main( )
{
    float A, B, C;

    printf("Enter three values\n");
    scanf("%f %f %f", &A, &B, &C);

    printf("\nLargest value is   ");

    if (A>B)
      {
       if (A>C)
          printf("%f\n", A);
       else
          printf("%f\n", C);
      }
     else
      {
       if (C>B)
          printf("%f\n", C);
       else
          printf("%f\n", B);
      }
}
```

Output

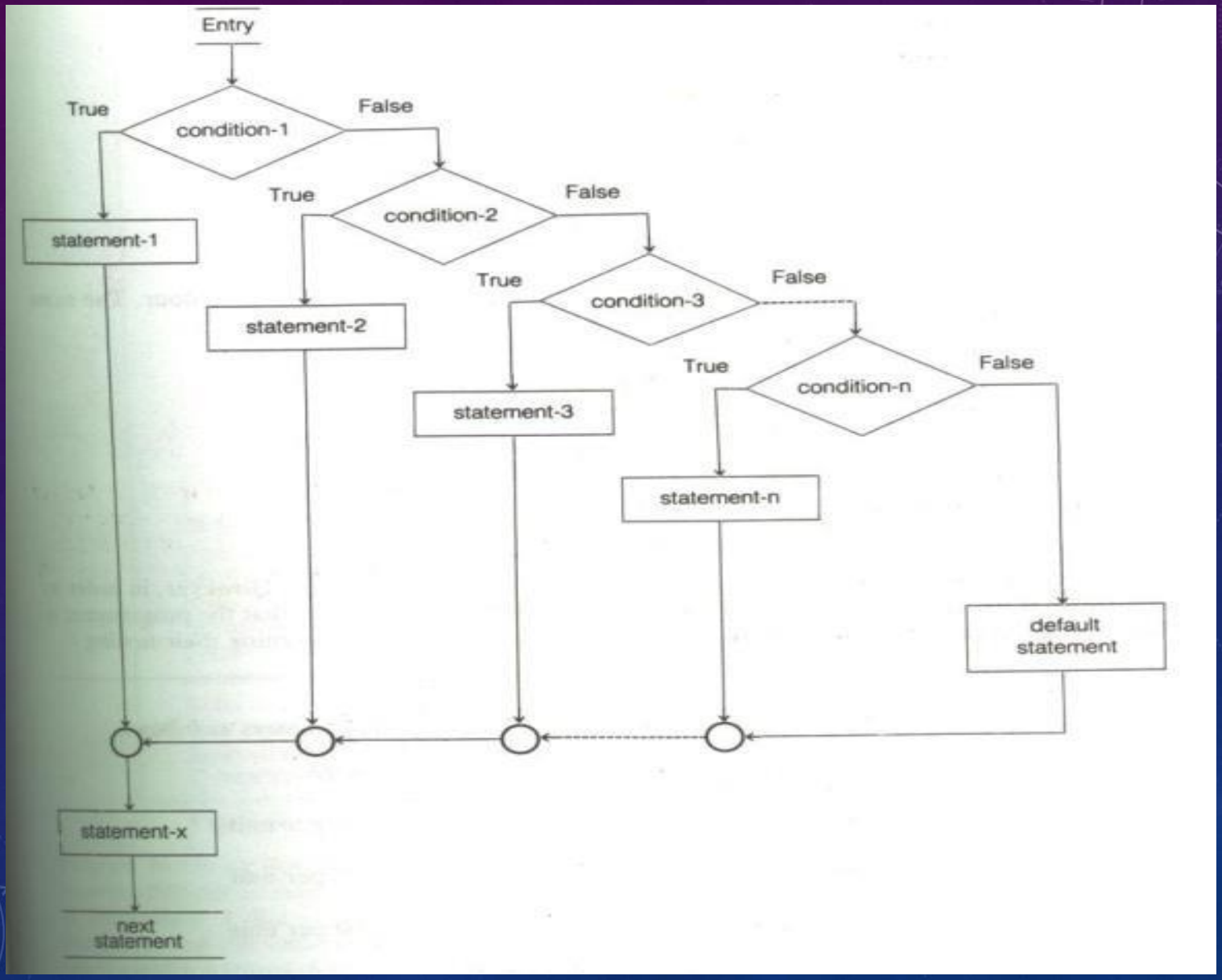
```
Enter three values
23445   67379   88843

Largest value is   88843.000000
```

The else if ladder

syntax-:

```
if  ( condition 1)
  statement 1 ;
     else if ( condition 2)
        statement 2 ;
           else if ( condition 3)
              statement 3 ;
                 else if( condition n)
                    statement n ;
                       else
                          default statement ;
        statement x;
```


Entry
True
condition-1
False
True
condition-2
False
statement-1
True
condition-3
False
statement-2
True
condition-n
False
statement-3
statement-n
default
statement
statement-x
next
statement

Program for else…if ladder

```
main( )
{
    int units, custnum;
    float charges;
    printf("Enter CUSTOMER NO. and UNITS consumed\n");
    scanf("%d %d", &custnum, &units);
    if (units <= 200)
      charges = 0.5 * units;
    else if (units <= 400)
          charges = 100 + 0.65* (units – 200);
        else if (units <= 600)
              charges = 230 + 0.8 * (units – 400);
            else
              charges = 390 + (units – 600);
    printf("\n\nCustomer No: %d: Charges = %.2f\n",
          custnum, charges);
}
```

Output

```
Enter CUSTOMER NO. and UNITS consumed 101   150
Customer No:101  Charges = 75.00
Enter CUSTOMER NO. and UNITS consumed 202   225
Customer No:202  Charges = 116.25
Enter CUSTOMER NO. and UNITS consumed 303   375
Customer No:303  Charges = 213.75
Enter CUSTOMER NO. and UNITS consumed 404   520
Customer No:404  Charges = 326.00
Enter CUSTOMER NO. and UNITS consumed 505   625
Customer No:505  Charges = 415.00
```

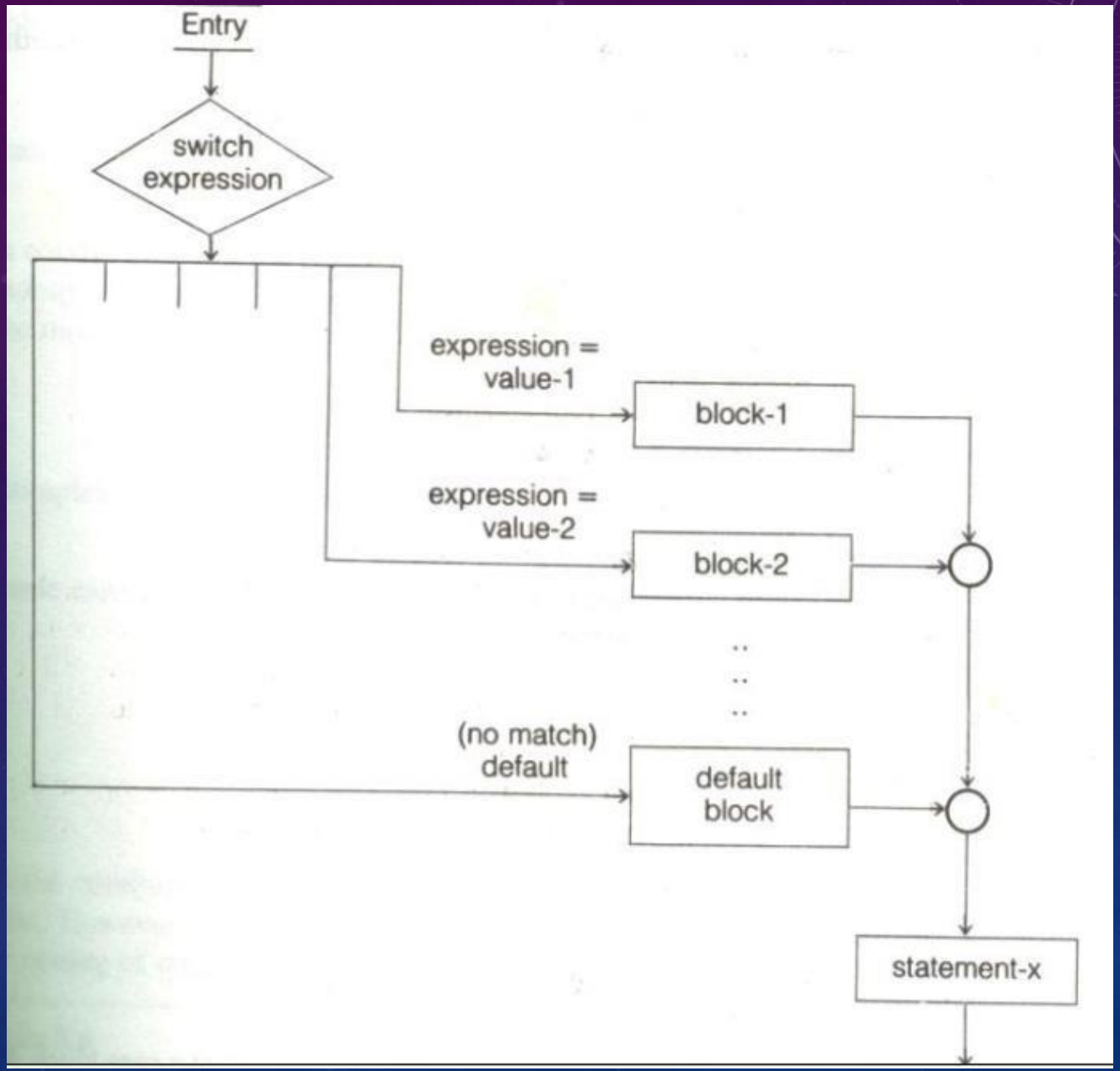
THE SWITCH STATEMENT

- The complexity of a program increases by increasing no. of if statement.
- To overcome this C has a built in multiway decision statement known as switch.

SYNTAX

```
switch (expression)
        {
    case value-1:                    case labels (257 case labels)
            block1;
             break;
    case value-2:
            block2;
             break;
          .
          .
          .
           default:
               default block;
                    break;
        }
statement x;
```

FLOWCHART FOR SWITCH STATEMENT



PROGRAM FOR SWITCH STATEMENT

```
main()
{
 int grade,mark,index;
 printf("Enter ur marks \n");
 scanf("%d",&mark);
 index=mark/10;
 switch(index)
 {
  case 10:
 case 9:
 case 8:
     grade="Honours";
     break;
 case 7:
case 6:
     grade="First Division";
     break;
case 5:
     grade="Second Division";
     break;
  case 4:
     grade="First Division";
     break;
  default:
      grade="Fail";
      break;
   }
  printf("%s",grade);
}
```

THE GOTO STATEMENT

- The goto statement is used for branching unconditionally.
- The goto statement breaks normal sequential execution of the program.
- The goto requires label to where it will transfer a control.
- Label is a valid variable name followed by a colon(:).

```
goto label:
  ----------
  - - - - - - -
 - - - - - - - -
  label:
statement;
```

FORWARD JUMP

```
  label:
  statement;
- - - - - - - - - - - -
- - - - - - - - - - - - -
----------------------
  goto label;;
```

BACKWARD JUMP

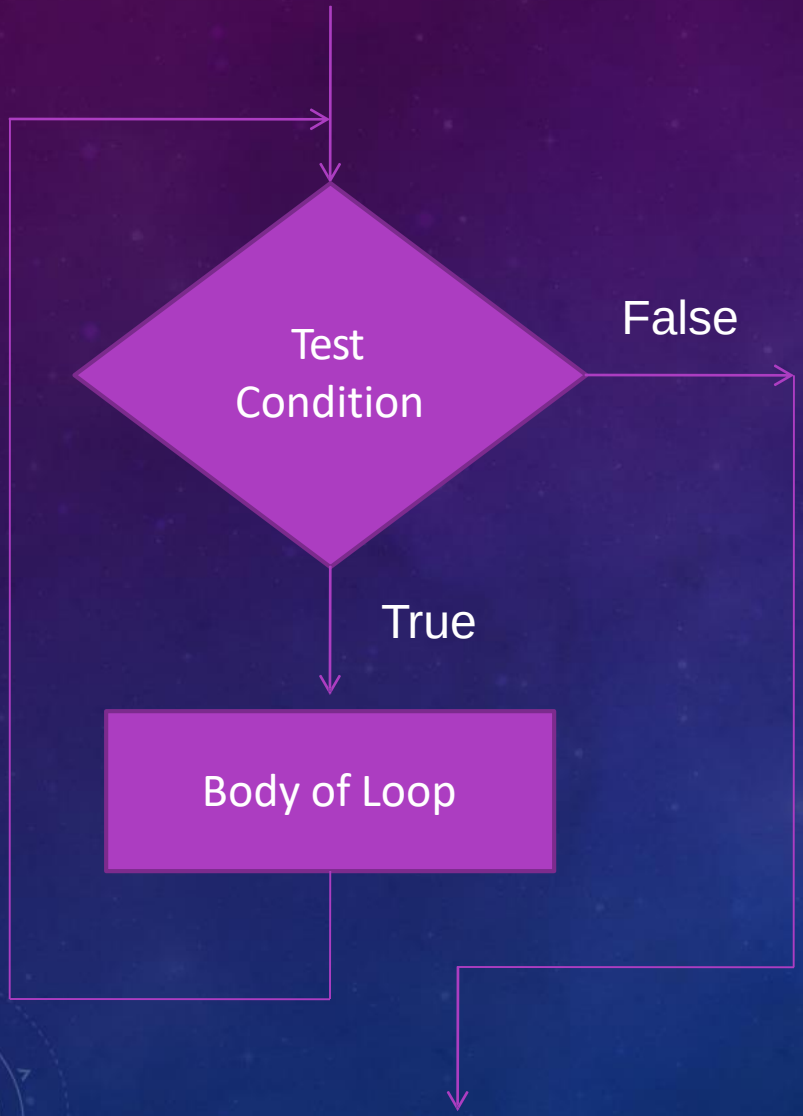
PROGRAM TO CALCULATE SUM OF SQUARES OF ALL INTEGERS

```
main()
{
 int sum=0,n=1;
loop:
 sum=sum+n*n;
 if( n==10)
 goto print;
else
 {
  n=n+1;
 goto loop;
 }
  print:
printf("\n Sum=%d",sum);
 }
```

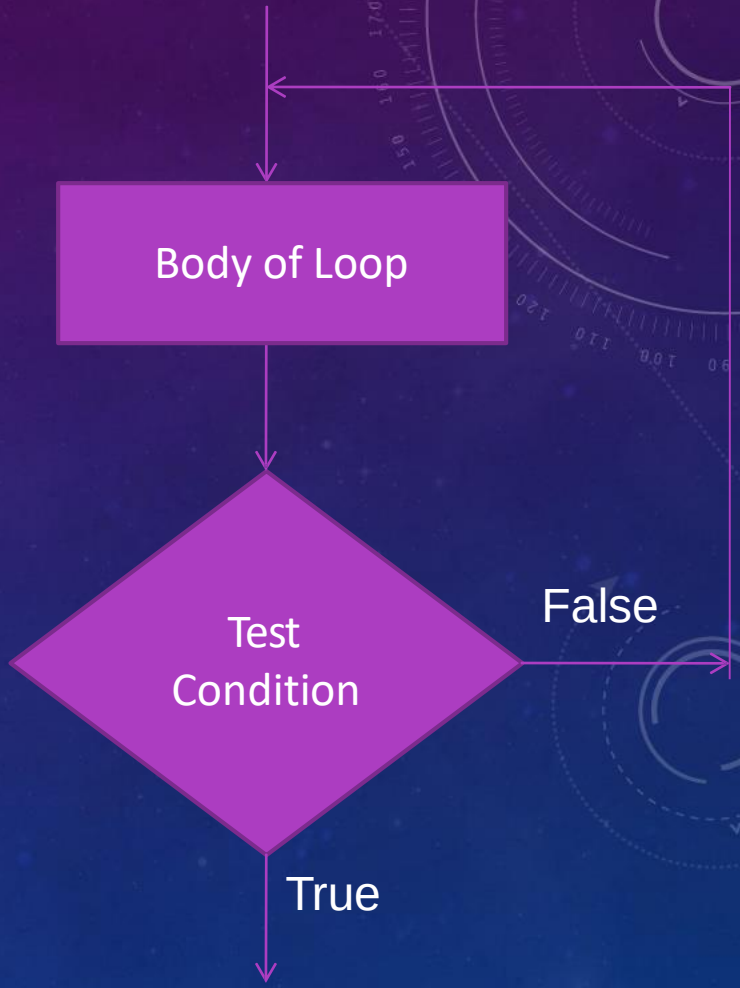
LOOPING

LOOPS

- A loop allows a program to repeat a group of statements, either any number of times or until some loop condition occurs
- Convenient if the exact number of repetitions are known
- Loop Consists of
  - Body of the loop
  - Control Statement



Entry Controlled/Pre-Test Loop

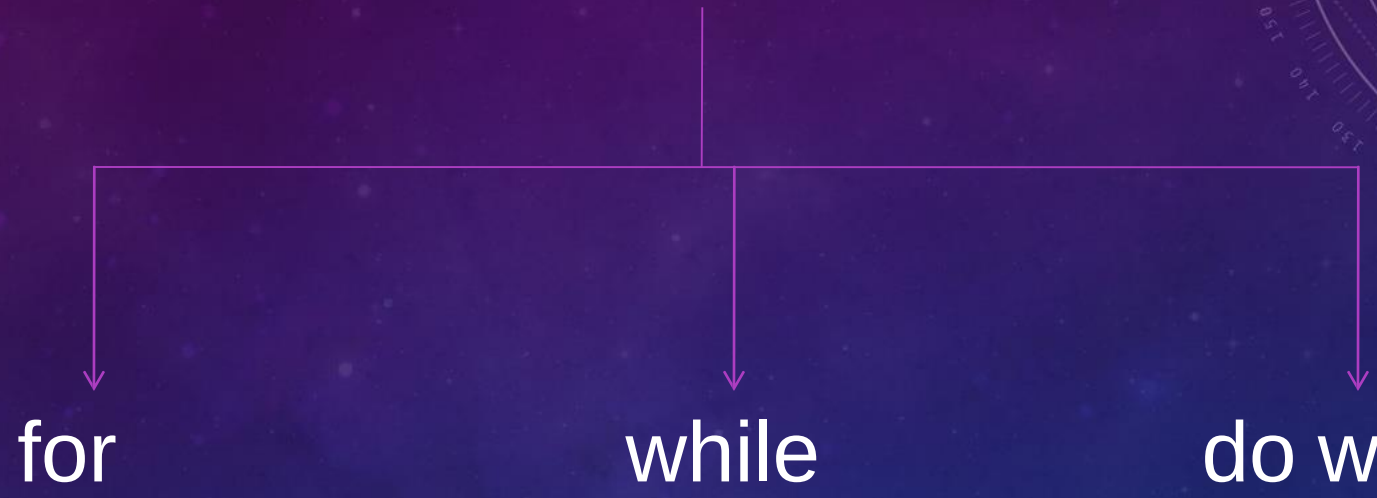


Exit Controlled/Post-Test Loop

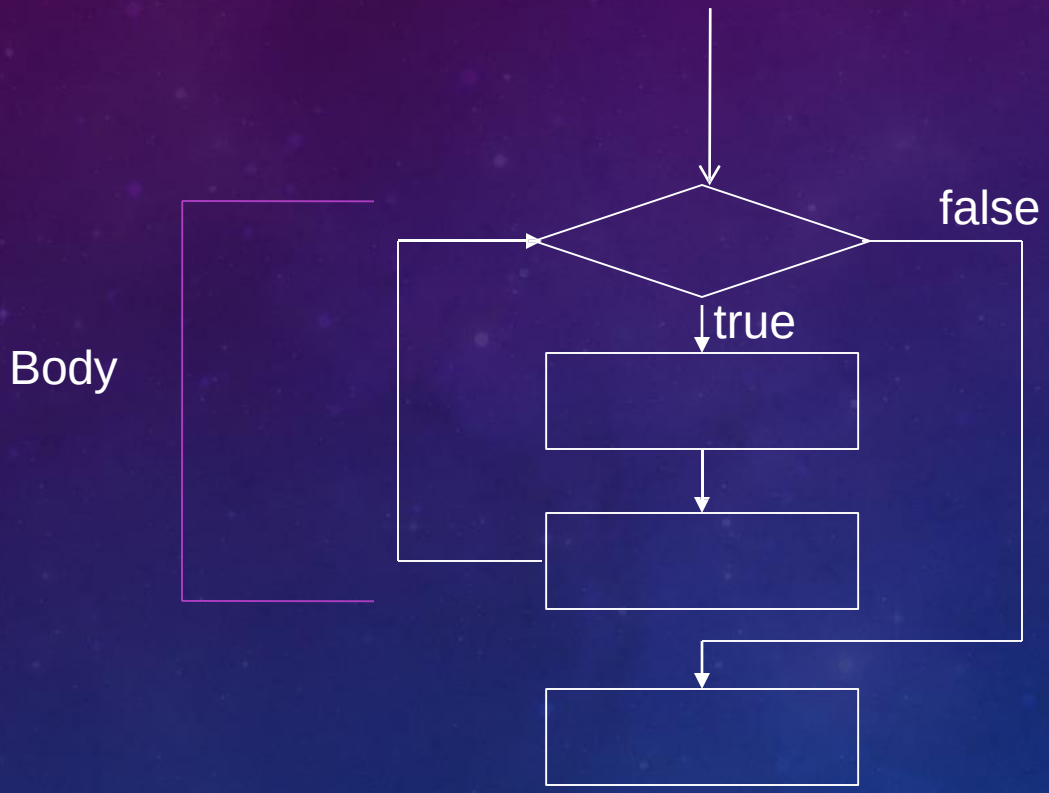
STEPS IN LOOPING

- Initialization of condition variable
- Execution of body of the loop
- Test the control statement
- Updating the condition variable

COMMON LOOPS



Repetition occurs as long as a condition is true

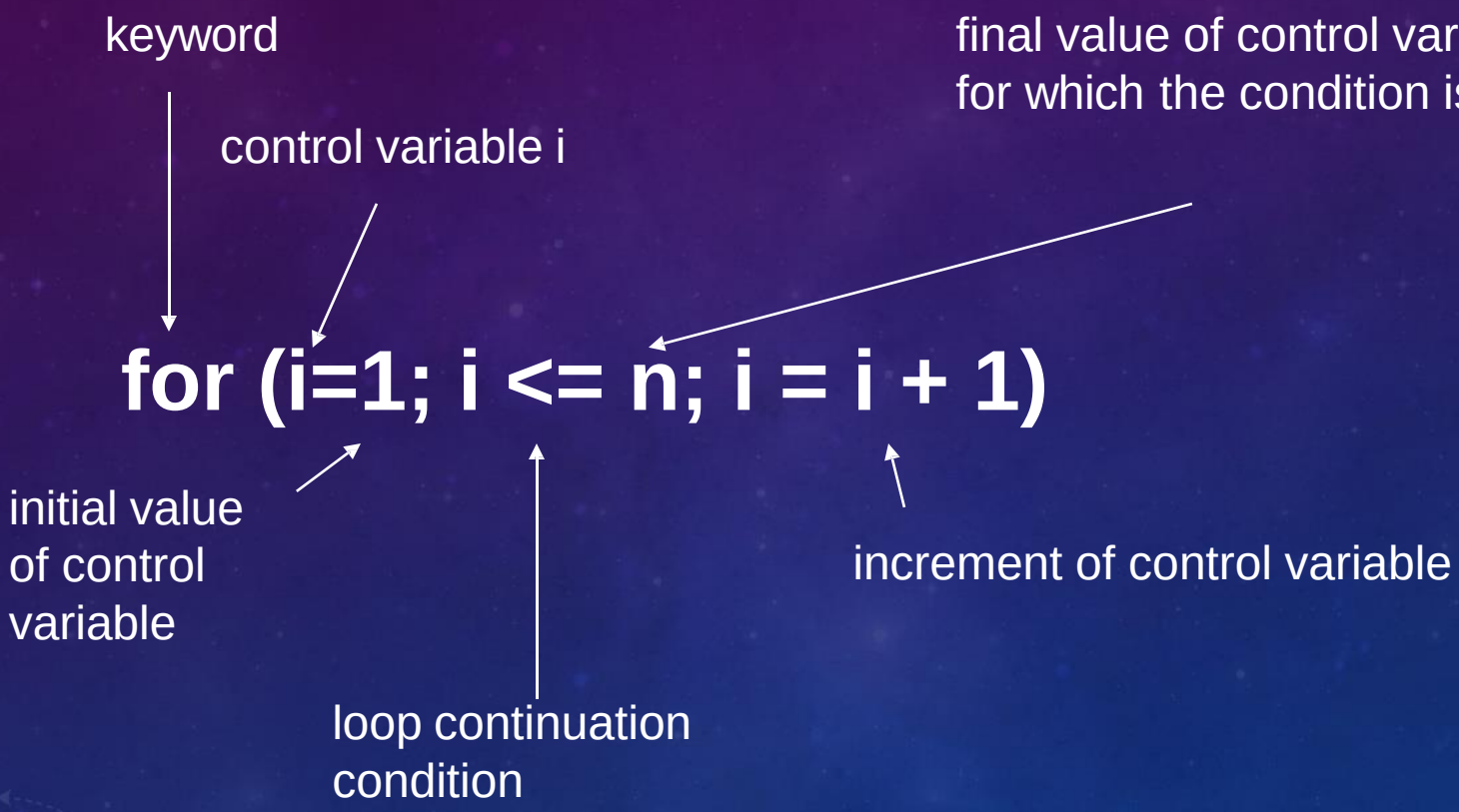


FOR

Syntax:

```
for (expr1; expr2; expr3)
    statement;
```

- expr1 controls the looping action,
- expr2 represents a condition that ensures loop continuation,
- expr3 modifies the value of the control variable initially assigned by expr1


keyword
control variable i
final value of control variable
for which the condition is true
for (i=1; i <= n; i = i + 1)
initial value
of control
variable
loop continuation
condition
increment of control variable

EXAMPLES

- Vary the control variable from 1 to 100 in increments of 1

```
for (i = 1; i <= 100; i++)
```

- Vary the control variable from 100 to 1 in increments of -1

```
for (i = 100; i >= 1; i--)
```

- Vary the control variable from 5 to 55 in increments of 5

```
for (i = 5; i <= 55; i+=5)
```

EXAMPLES 2

```
#include <stdio.h>

void main()
{
    /* a program to produce a Celsius to Fahrenheit conversion chart for the
    numbers 1 to 100 */
    int celsius;
    for (celsius = 0; celsius <= 100; celsius++)
        printf("Celsius: %d Fahrenheit: %d\n", celsius, (celsius * 9) / 5 + 32);
    }
```

NESTED FOR LOOPS

- Nested means there is a loop within a loop
- Executed from the inside out
  - Each loop is like a layer and has its own counter variable, its own loop expression and its own loop body
  - In a nested loop, for each value of the outermost counter variable, the complete inner loop will be executed once

General form

```
for (loop1_exprs) {
    loop_body_1a

    for (loop2_exprs) {
    loop_body_2
    }
    loop_body_1b
}
```

WHILE

```
expression 1;
while (expression 2)
{
statements;
expression 3;
}
```

initialization

condition checking

updation of condition

- The statement is executed repeatedly as long as the expression is true (non zero)

- The cycle continues until expression becomes zero, at which point execution resumes after statement

- If the test expression in a while loop is false initially, the while loop will never be executed

```
int i = 1, sum = 0;
while (i <= 10)
{
  sum = sum + i;
        i= i + 1;
}
printf("Sum = %d\n", sum);
```

FOR AND WHILE

```
for(expr1; expr2; expr3)
   statement;
```

```
  expr1;
 while(expr2)
{
    statement;
     expr3;
}
```

BREAK AND CONTINUE

- These interrupt normal flow of control
- **break** causes an exit from the innermost enclosing loop
- **continue** causes the current iteration of a loop to stop and the next iteration to begin immediately

```
while (expression)
{
    statements;
    if(condition)
    break;
    more_statements
}

while (expression)
{
    statements
    continue;
    more_statements
}
```

DO-WHILE

- When a loop is constructed using while, the test for continuation is carried out at the beginning of each pass
- With do-while the test for continuation takes place at the end of each pass

```
do
{
  statement
} while (expression);
```

EXAMPLE

```
int i = 1, sum = 0;
    do
    {
      sum = sum + i;
          i= i + 1;
    }while(i<=10);
    printf("Sum = %d\n", sum);
```

WHILE VS. DO-WHILE

- **while** -- the expression is tested first, if the result is false, the loop body is never executed

- **do-while** -- the loop body is always executed once. After that, the expression is tested, if the result is false, the loop body is not executed again

Thank you