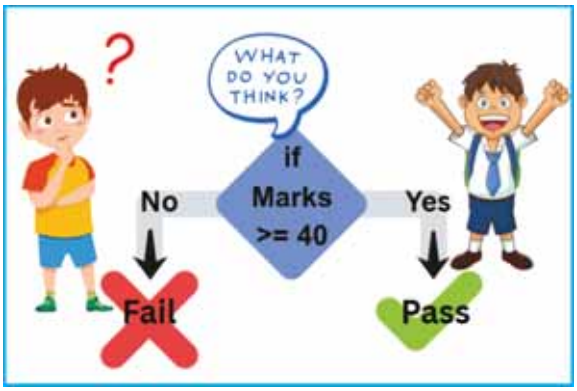
નિર્ણય લેવાની નિયંત્રણ સંરચનાઓ

પરિચય

અગાઉના પ્રકરણમાં, આપણે C પ્રોગ્રામિંગના ઈનપુટ/આઉટપુટ ફંકશનના ઉપયોગ વિશે ચર્ચા કરી છે. આ પ્રકરણ C પ્રોગ્રામિંગમાં નિર્ણય લેવાની નિયંત્રણ સંરચનાઓ (Decision Making Control Structures) પર ધ્યાન કેન્દ્રિત કરે છે. અત્યાર સુધી ચર્ચા કરાયેલા મોટાભાગના C પ્રોગ્રામ અમલ માટે ક્રમિક સંરચનાને અનુસરે છે. આનો અર્થ એ છે કે પ્રોગ્રામમાં જે ક્રમમાં વિધાનો (statements) દેખાય છે, તે જ ક્રમમાં તે તેનો અમલ (execute) કરવામાં આવે છે. પરંતુ, વાસ્તવિક જીવનના પ્રોગ્રામમાં, આપણે કદાચ અમુક શરતોના આધારે, આપણા પ્રોગ્રામના ફક્ત કેટલાક જ વિધાનોનો જ અમલ કરવા માગીએ છીએ. આ હેતુ માટે, સૂચનાઓના અમલના ક્રમના પ્રવાહને બદલવો જરૂરી છે. જેના માટે C ભાષા ખાસ વિધાનો પૂરા પાડે છે, જે પ્રોગ્રામની અંદરની સૂચનાઓના પ્રવાહમાં ફેરફાર કરવા સક્ષમ છે. આ વિધાનોને નિર્ણય અને નિયંત્રણ સંરચના વિધાનો તરીકે ઓળખવામાં આવે છે.

નિર્ણય લેવાની સંરચનાઓ (Decision Making Control Structures)

નિર્ણય લેવાની સંરચનાઓ કોઈપણ પ્રોગ્રામિંગ ભાષામાં પાયાના નિર્માણ ઘટકોમાં આવે છે. તે પ્રોગ્રામને અમુક શરતોના આધારે પસંદગીઓ કરવા, કાર્યોનું પુનરાવર્તન કરવા અને આપણે જે અમલ કરવા માંગીએ છીએ તેના મુજબ અમલના પ્રવાહને નિયંત્રિત કરવા દે છે.



આકૃતિ 10.1 : પ્રોગ્રામમાં નિર્ણય લેવાની સ્થિતિ

નિર્ણય સંરચનાઓ (Decision Structures)

પ્રોગ્રામિંગમાં નિર્ણય સંરચનાઓને પસંદગી સંરચનાઓ (Selection structures) તરીકે પણ ઓળખવામાં આવે છે. નિર્ણય સંરચનાઓનો મુખ્ય ઉદ્દેશ્ય પ્રોગ્રામના અમલ દરમિયાન નિર્ણય લેવામાં મદદ કરવાનો છે. જ્યારે કોઈ ચોક્કસ શરત સાચી કે ખોટી છે, તેના મૂલ્યાંકનના આધારે પ્રોગ્રામ જુદા જુદા વિધાનોનો અમલ કરે છે. જુઓ આકૃતિ 10.1.

ઉદાહરણ : નીચેના કોડમાં *printf()* ("Pass" અથવા "Fail" માંથી કોઈપણ એક) નો અમલ marks ચલમાં સંગ્રહિત મૂલ્ય પર આધારિત છે.

```
int marks = 50;
if (marks >= 40)
    printf("Pass");
else
    printf("Fail");
```

C પ્રોગ્રામિંગમાં સામાન્ય નિર્ણય સંરચનાઓ નીચે મુજબ છે:

- *if* વિધાન
  - if-else વિધાન
  - else-if ladder
- switch વિધાન

નિયંત્રણ સંરચનાઓ (Control Structures)

નિયંત્રણ સંરચનાઓ એ પ્રોગ્રામિંગના એવા ખ્યાલો છે જે પ્રોગ્રામમાં આપેલા વિવિધ વિધાનોના અમલ પ્રવાહને નિયંત્રિત કરે છે. બધી જ નિર્ણય સંરચનાઓ (જેવી કે *if, if-else, switch*) એ નિયંત્રણ સંરચનાઓના પ્રકાર જ છે. ટેબલ 10.1 વિવિધ નિયંત્રણ સંરચનાઓના પ્રકારો દર્શાવે છે.

પ્રકાર	વર્ણન
ક્રમિક (Sequential)	મૂળભૂત પ્રવાહ (Default flow); પ્રોગ્રામના બધા વિધાનોનો એક પછી એક એમ ક્રમશઃ અમલ થાય છે.
નિર્ણય લેનાર (Decision-making)	*if, if-else, switch* વગેરે વિધાનોનો ઉપયોગ કરીને શરતના આધારે કોડના ભાગનો અમલ કરે છે.
લૂપિંગ (Looping) (પુનરાવર્તન/ Itertaion)	*for, while, do-while* જેવા વિધાનોનો ઉપયોગ કરીને જ્યાં સુધી શરત સાચી હોય ત્યાં સુધી કોડનું પુનરાવર્તન કરે છે.
જમ્પિંગ (નિયંત્રણનું સ્થાનાંતરણ/transfer of control)	*break, continue, goto* અને *return* જેવા વિધાનોનો ઉપયોગ કરીને પ્રોગ્રામના અન્ય ભાગમાં નિયંત્રણનું સ્થાનાંતરણ કરે છે.

ટેબલ 10.1 : નિયંત્રણ સંરચનાઓના પ્રકારો

ચાલો આપણે દરેક પ્રકારની નિયંત્રણ સંરચનાઓ ઉદાહરણ સાથે સમજીએ.

ક્રમિક નિયંત્રણ સંરચનાઓ (Sequential Control Structures)

C પ્રોગ્રામિંગમાં, ક્રમિક નિયંત્રણ સંરચનાનો અર્થ એ છે કે પ્રોગ્રામ લખેલા વિધાનોનો સમાન ક્રમમાં, પહેલાં પછી બીજું, બીજા પછી ત્રીજું, એમ ક્રમમાં અમલ થાય છે. ક્રમિક નિયંત્રણ સંરચના સૌથી સરળ નિયંત્રણ સંરચના છે. તે પ્રોગ્રામના કોડનો ઉપરથી નીચે સુધી ક્રમશઃ અમલ કરે છે. પ્રોગ્રામમાં કોઈ નિર્ણય અથવા પુનરાવર્તિત ક્રિયાઓ જરૂરી ન હોય તેવા સરળ પ્રોગ્રામ લખવા તે ઉપયોગી છે.

ક્રમિક નિયંત્રણ સંરચનાના મુખ્ય લક્ષણો

- **ક્રમિક અમલ:** પહેલા વિધાન પછી બીજું વિધાન એમ ક્રમશઃ ચલાવવામાં આવે છે.
- **વાંચવા અને સમજવામાં સરળ:** આવા પ્રોગ્રામો વાંચવા અને સમજવા માટે સરળ હોય છે.
- **શરતો અથવા લૂપનો ઉપયોગ નહીં:** શરતી વિધાનો (જેમ કે *if, if-else*) અથવા લૂપીંગ (*for, while*) નો ઉપયોગ થતો નથી.
- **સરળ કાર્યો માટે ઉપયોગ:** ઇનપુટ, આઉટપુટ અથવા ગણતરીઓ જેવા સરળ કાર્યો માટે ઉપયોગ થાય છે.

ક્રમિક નિયંત્રણ સંરચનાનું ઉદાહરણ

```
/* Program to illustrate use of Sequential Control Structure */
#include <stdio.h>
int main() {
    int a, b, sum;

    printf("Enter two numbers: ");
```

```
    scanf("%d %d", &a, &b);

    sum = a + b;

    printf("Sum = %d\n", sum);
    return 0;
}
Result:

Enter two numbers: 5 6
Sum = 11
```

આ પ્રોગ્રામમાં મુખ્ય તર્ક (logic) એ ત્રણ *int* પ્રકારના ચલ (variables) જાહેર કરવાનો, પછી યુઝર પાસેથી ઇનપુટ તરીકે બે સંખ્યાઓ વાંચવાનો, તેનો સરવાળો કરવાનો અને પછી પરિણામ દર્શાવવાનો છે. ધ્યાન રાખો કે આ પ્રોગ્રામમાં, તમામ વિધાનો એક સ્પષ્ટ ક્રમને અનુસરીને એક પછી એક અમલમાં આવે છે. ચલોની ઘોષણા (declaration), ઇનપુટ માટે પૂછવું, ઇનપુટ વાંચવું, સરવાળો કરવો અને સરવાળો છાપવો. તેથી, આવા પ્રોગ્રામને ક્રમિક નિયંત્રણ સંરચના પ્રોગ્રામ તરીકે વર્ગીકૃત કરવામાં આવે છે

નિર્ણય લેવાના વિધાનો (Decision Making Statements)

C પ્રોગ્રામિંગમાં નિર્ણય લેવાના (પસંદગીના) વિધાનો પ્રોગ્રામને ચોક્કસ શરતોના આધારે પસંદગી કરવાની મંજૂરી આપે છે. આ વિધાનો પ્રોગ્રામને શરત સાચી કે ખોટી છે તેના આધારે કોડનો કયો બ્લોક ચલાવવો તે નક્કી કરવામાં મદદ કરે છે. સામાન્ય નિર્ણય લેવાના વિધાનોમાં *if, if-else,* નેસ્ટેડ *if* (nested *if*), અને *switch* નો સમાવેશ થાય છે. તે પ્રોગ્રામના પ્રવાહને નિયંત્રિત કરવા અને આપણી જરૂરિયાત મુજબની વિવિધ પરિસ્થિતિઓને સંભાળવા માટે ઉપયોગી છે. નિર્ણય લેવાના વિધાનોને પસંદગીના વિધાનો (Selection statements) અથવા બ્રાન્ચિંગ વિધાનો (Branching statements) તરીકે પણ ઓળખવામાં આવે છે.

ચાલો, નિર્ણય લેવાના દરેક પ્રકારના વિધાનને ઉદાહરણો સાથે સમજીએ.

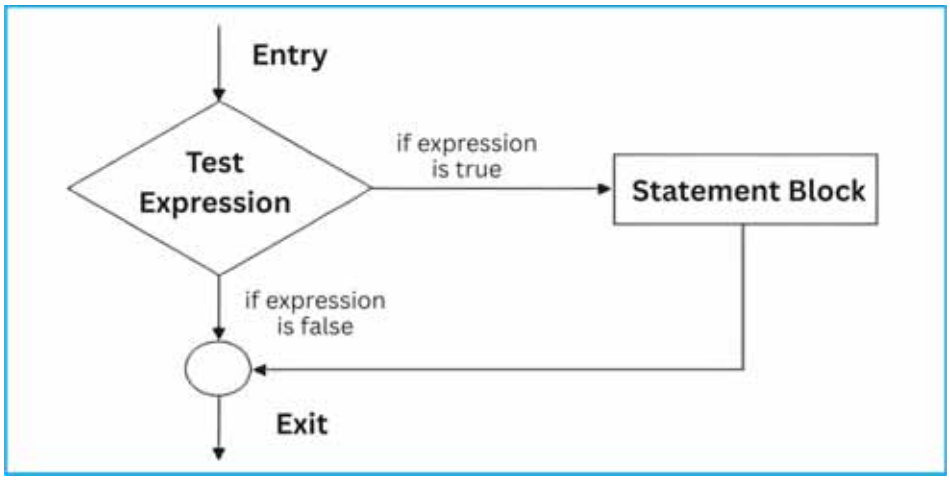
if વિધાન (if statement)

if વિધાન C પ્રોગ્રામિંગમાં સૌથી વધુ ઉપયોગમાં લેવાતું અને સરળ નિર્ણય લેવાનું વિધાન છે. *if* વિધાનની વાક્યરચના (Syntax) નીચે મુજબ છે:

```
if(test expression) {
  Statement block of code;
}
```

if વિધાન કૌંસ ( ) ની અંદરની પદાવલી (test expression) ને તપાસે છે. જો તે સાચું હોય, તો છગડિયા કૌંસ { } ની અંદરનો કોડ અમલ કરવામાં આવે છે. જો શરતી વિધાન ખોટું હોય તો બ્લોકની અંદરનો કોડ અવગણવામાં આવે છે અને પ્રોગ્રામની આગામી સૂચનાનો અમલ કરે છે.

અહીં, ચકાસણી માટેની પદાવલી C ભાષાની કોઈપણ માન્ય પદાવલી (એક્સપ્રેશન) હોઈ શકે છે. જો *if* બ્લોકમાં એક જ વિધાન (statement) હોય તો, { } વૈકલ્પિક છે. { } ની અંદરના કોડના બ્લોકને સ્ટેટમેન્ટ બ્લોક (statement block) કહેવામાં આવે છે. સ્ટેટમેન્ટ બ્લોકમાં C ભાષાનું કોઈપણ માન્ય વિધાન અથવા વિધાનો હોઈ શકે છે. નોંધ લો કે ( ) અંદરના શરતી વિધાન (test expression)નો ભાગ અર્ધવિરામથી સમાપ્ત થવો ન જોઈએ. આકૃતિ 10.2 સાદા *if* વિધાનનો ફ્લોચાર્ટ દર્શાવે છે.



આકૃતિ 10.2 : *if* વિધાનનો ફ્લોચાર્ટ

નોંધ: C પ્રોગ્રામિંગ ભાષા શૂન્ય સિવાયની અને નલ (null) સિવાયની કોઈપણ કિંમતને સાચી માને છે. તેથી, સૂત્ર (5 + 5) ને સાચું ગણવામાં આવે છે. તેવી જ રીતે, શૂન્ય અથવા નલ (null) કિંમતને ખોટી માનવામાં આવે છે. તેથી, જે શરત કે સૂત્રનું મૂલ્યાંકન શૂન્ય થાય છે (દા.ત., 5 - 5), તેને ખોટું ગણવામાં આવે છે.

if વિધાનનું ઉદાહરણ

```
/* Program to illustrate use of if  statement. It will check
   whether a given number is positive or not using if  statement.
*/
#include<stdio.h>
int main() {
     int number;
     printf("Enter number:");
     scanf("%d",&number);

     if (number > 0)
         printf("Number is positive.");

     if (number <= 0)
         printf("Number is not positive.");
     return 0;
}
Result:

Enter number:5
Number is positive.
```

આ પ્રોગ્રામ આપેલ સંખ્યા ધન છે કે નહીં તે ચકાસવા માટે *if* વિધાનના ઉપયોગનું નિદર્શન કરે છે. તે એક પૂર્ણાંક (integer) ચલ number ઘોષિત કરે છે. તે પછીના *printf()* અને *scanf()* ફંક્શન લેબલ સાથે એક સંખ્યા વાંચવા માટે ઉપયોગ થાય છે. પ્રથમ *if* તપાસે છે કે દાખલ કરેલ સંખ્યા 0 કરતાં મોટી છે કે નહીં (number > 0); જો તે સાચી હોય, તો તે "Number is positive." ("સંખ્યા ધન છે.") એમ પ્રિન્ટ કરે છે. બીજી *if* શરત તપાસે છે કે સંખ્યા 0 કરતાં નાની અથવા 0 જેટલી છે કે નહીં (number <= 0); જો તે સાચી

હોય, તો તે "Number is not positive." ("સંખ્યા ધન નથી.") એમ પ્રિન્ટ કરે છે. આ પ્રોગ્રામ બે અલગ *if* વિધાનોનો ઉપયોગ કરે છે, તેથી તે બંને શરતોનું મૂલ્યાંકન સ્વતંત્ર રીતે કરે છે.

ચાલો સમજીએ કે આ જ પ્રોગ્રામને *if-else* વિધાનનો ઉપયોગ કરીને વધુ સરળ અને કાર્યક્ષમ રીતે કેવી રીતે લખી શકાય છે.

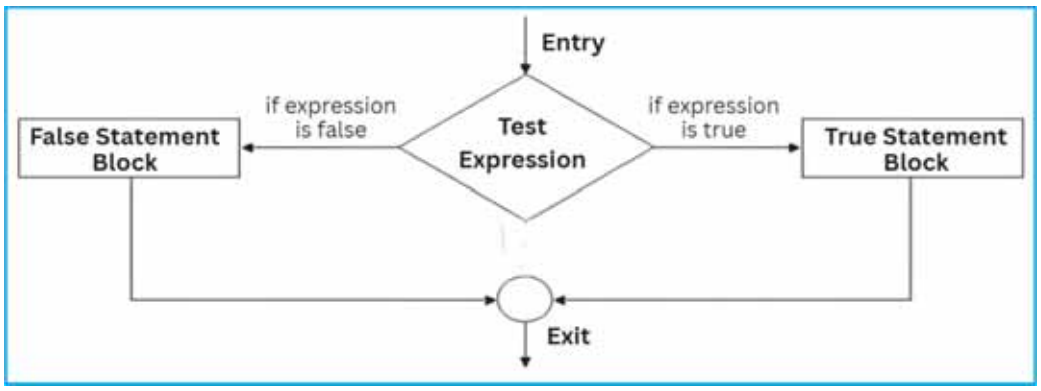
if-else વિધાન (if-else statement)

C પ્રોગ્રામિંગમાં *if-else* વિધાન એ એક પ્રકારનું નિર્ણય લેવાની નિયંત્રણ સંરચના છે, જેનો ઉપયોગ આપેલી પદાવલીના આધારે બે વિકલ્પોમાંથી એક કોડ બ્લોક ને અમલ કરવા માટે થાય છે. જો પદાવાલીમાંની શરત સાચી હોય તો *if* બ્લોકની અંદરનો કોડ અમલી થાય છે અન્યથા *else* બ્લોકની અંદરનો કોડ અમલી થાય છે. તે શરતી નિર્ણયોના આધારે પ્રોગ્રામના પ્રવાહને નિયંત્રિત કરવામાં મદદ કરે છે. આ વિધાનનો ઉપયોગ સામાન્ય રીતે તુલના, માન્યતા (validations) અને બ્રાન્ચિંગ (branching) જેવી શરતો તપાસવા માટે થાય છે.

if-else વિધાનની વાક્યરચના નીચે મુજબ છે:

```
if(test expression) {
    True statement-block of code;
}
else {
    False statement-block of code;
}
```

if વિધાન કૌંસ ( ) માં રહેલા ટેસ્ટ પદાવલી (test expression) ની તપાસ કરે છે. જો તે સાચી હોય તો છગડિયા કૌંસમાં આપેલો કોડ (True statement-block) અમલમાં મૂકાય છે. જો તે ખોટી હોય, તો *else* બ્લોકમાં આપેલો કોડ (False statement-block) અમલમાં મૂકાય છે. આકૃતિ 10.3 *if-else* વિધાનનો ફ્લોચાર્ટ દર્શાવે છે.



આકૃતિ 10.3 : *if-else* નો ફ્લોચાર્ટ

if-else વિધાનનું ઉદાહરણ

```
/* Program to check whether a given number is positive or not using
the if-else statement. */

#include<stdio.h>
int main() {
    int number;

    printf("Enter number:");
    scanf("%d",&number);
```

```
    if (number > 0)
        printf("Number is positive.");
    else
        printf("Number is not positive.");
    return 0;
}

Result:

Enter number:-5
Number is not positive.
```

આ પ્રોગ્રામ યુઝર પાસેથી એક પૂર્ણાંક ઈનપુટ વાંચે છે અને તે સંખ્યા ધન છે કે નહીં તે ચકાસવા માટે *if-else* વિધાનનો ઉપયોગ કરે છે. તે સૌપ્રથમ number નામનો એક પૂર્ણાંક ચલ ઘોષિત કરે છે અને પછી *printf()* નો ઉપયોગ કરીને યુઝરને મૂલ્ય દાખલ કરવા માટે પૂછે છે, ત્યારબાદ *scanf()* વડે ઈનપુટ વાંચે છે. *if* શરત તપાસે છે કે શું number 0 થી મોટો છે; જો તે સાચું હોય, તો તે "Number is positive." પ્રિન્ટ કરે છે. જો શરત ખોટી હોય (એટલે કે, સંખ્યા શૂન્ય અથવા ઋણ છે), તો *else* બ્લોક અમલમાં મુકાય છે અને "Number is not positive." પ્રિન્ટ કરે છે. આ નિયંત્રણ સંરચના ખાતરી કરે છે કે બે *printf()* માંથી ફક્ત એક જ અમલમાં મુકાય છે, જે અગાઉના *if* વિધાનના ઉદાહરણમાં બતાવ્યા પ્રમાણે બે અલગ અલગ *if* નિવેદનોનો ઉપયોગ કરવા કરતાં આઉટપુટને વધુ કાર્યક્ષમ અને વાંચવા યોગ્ય બનાવે છે.

નેસ્ટેડ-if વિધાન (Nested-if Statement)

નેસ્ટેડ-*if* વિધાન (Nested-*if* statement) એટલે એક *if* નિવેદન જે બીજા *if* અથવા *else* બ્લોક ની અંદર અસ્તિત્વ ધરાવે છે. આ માળખું જટિલ નિર્ણય લેવાની પ્રક્રિયા પર વધુ ચોક્કસ નિયંત્રણની મંજૂરી આપે છે, જ્યાં એક શરત બીજી શરતનાં પરિણામ પર આધારિત હોય છે. નેસ્ટેડ-*if* વિધાનો જ્યારે નિર્ણય શરતોના બહુવિધ સ્તરો પર આધારિત હોય ત્યારે ઉપયોગી થાય છે.

નેસ્ટેડ-if વિધાનનું ઉદાહરણ

```
/* Program to illustrate use of nested-if  statement.
   Find the largest of given 3 numbers. */

#include <stdio.h>
int main() {
    int number1, number2, number3;

    printf("Enter three numbers:");
    scanf("%d %d %d", &number1, &number2, &number3);

    if(number1 > number2)
    {
        if(number1 > number3)
            printf("Number1 is the biggest number.");
        else
            printf("Number3 is the biggest number.");
```

```
    }
    else
    {
        if(number2 > number3)
            printf("Number2 is the biggest number.");
        else
            printf("Number3 is the biggest number.");
    }
    return 0;
}

Result:
Enter three numbers:5 10 15
Number3 is the biggest number.
```

આ C પ્રોગ્રામ યુઝર પાસેથી ત્રણ સંખ્યાઓ ઇનપુટ તરીકે લે છે અને સૌથી મોટી સંખ્યા શોધવા અને પ્રિન્ટ કરવા માટે નેસ્ટેડ-*if* નિવેદનોનો ઉપયોગ કરે છે. સૌપ્રથમ, તે ચકાસે છે કે શું number1 એ number2 કરતાં મોટો છે. જો આ સાચું હોય, તો તે પછી ચકાસે છે કે શું number1 એ number3 કરતાં પણ મોટો છે. જો આ બંને શરતો સાચી હોય, તો તે પ્રિન્ટ કરે છે કે number1 સૌથી મોટો છે. જો નહીં, તો તે પ્રિન્ટ કરે છે કે number3 સૌથી મોટો છે. જ્યારે પ્રથમ શરત (number1 > number2) ખોટી હોય, ત્યારે તે બીજી શરત (number2 > number3) ચકાસે છે. જો હા, તો તે પ્રિન્ટ કરે છે કે number2 સૌથી મોટો છે. અન્યથા, તે પ્રિન્ટ કરે છે કે number3 સૌથી મોટો છે.

else-if લેડર (else-if Ladder)

else-if લેડરનો ઉપયોગ ત્યારે થાય છે જ્યારે આપણે એક પછી એક બહુવિધ શરતો ચકાસવાની જરૂર હોય. તે ઘણા સંભવિત વિકલ્પોમાંથી એક વિકલ્પ પસંદ કરવામાં મદદ કરે છે. શરતોની ચકાસણી ઉપરથી નીચે તરફ કરવામાં આવે છે, અને જે પ્રથમ સાચી શરત મળે, તેના માટેનો કોડનો બ્લોક અમલમાં મૂકાય છે. જો એક પણ શરત સાચી ન હોય, તો છેલ્લો *else* બ્લોક અમલમાં મૂકાય છે.

else-if લેડરનું ઉદાહરણ

ધારો કે આપણે વિદ્યાર્થીના ગુણના આધારે જુદા જુદા ગ્રેડ (A, B, C, D, F) પ્રિન્ટ કરવા માંગીએ છીએ. આ પ્રકારની પરિસ્થિતિઓમાં *else-if* લેડર ઉપયોગી છે.

```
/* Program to illustrate use of else-if ladder statement.
   Print student's grade based on marks. */

#include <stdio.h>
int main() {
    int marks = 75;     /* Assume that student has scored 75 */
    if(marks >= 90)
        printf("Grade: A");
    else if(marks >= 80)
            printf("Grade: B");
        else if(marks >= 70)
```

```
                printf("Grade: C"); /*This will be executed*/
            else if(marks >= 40)
                printf("Grade: D");
            else
                printf("Grade: F");
    return 0;
}
Result:
Grade: C
```

આ C પ્રોગ્રામ ગુણના આધારે વિદ્યાર્થીનો ગ્રેડ નક્કી કરવા અને પ્રિન્ટ કરવા માટે *else-if* લેડરનો ઉપયોગ કરે છે. marks ચલને 75 નું મૂલ્ય આપવામાં આવ્યું છે. પ્રોગ્રામ સૌપ્રથમ ચકાસે છે કે શું marks 90 કે તેથી વધુ છે (ગ્રેડ A), પછી ચકાસે છે કે શું તે 80 કે તેથી વધુ છે (ગ્રેડ B), અને પછી ચકાસે છે કે શું તે 70 કે તેથી વધુ છે (ગ્રેડ C). કેમકે 75 એ 70 કરતા વધારે છે પરંતુ 80 કરતા ઓછો છે, તેથી ગ્રેડ C માટેની શરત સાચી છે, તેથી તે "Grade: C" પ્રિન્ટ કરે છે. ઉપરથી નીચે સુધી એક સમયે માત્ર એક જ શરત ચકાસવામાં આવે છે, અને જ્યારે યોગ્ય મેચ મળી આવે છે, ત્યારે તે સંબંધિત બ્લોકનો અમલ થાય છે અને બાકીના છોડી દેવામાં આવે છે.

switch વિધાન (switch Statement)

C પ્રોગ્રામિંગમાં *switch* વિધાન વ્યાપકપણે ઉપયોગમાં લેવાતું નિયંત્રણ સંરચના છે જે આપણને મૂલ્યના આધારે કોડના વિવિધ બ્લોક્સને અમલમાં મૂકવાની મંજૂરી આપે છે. જ્યારે આપણી પાસે બહુવિધ સંભવિત અમલના માર્ગો હોય, ત્યારે તે *if-else*, *if-else* નિવેદનોની લાંબી શૃંખલાનો વૈકલ્પિક ઉપાય છે. *switch* વિધાનની વાક્યરચના નીચે પ્રમાણે છે.

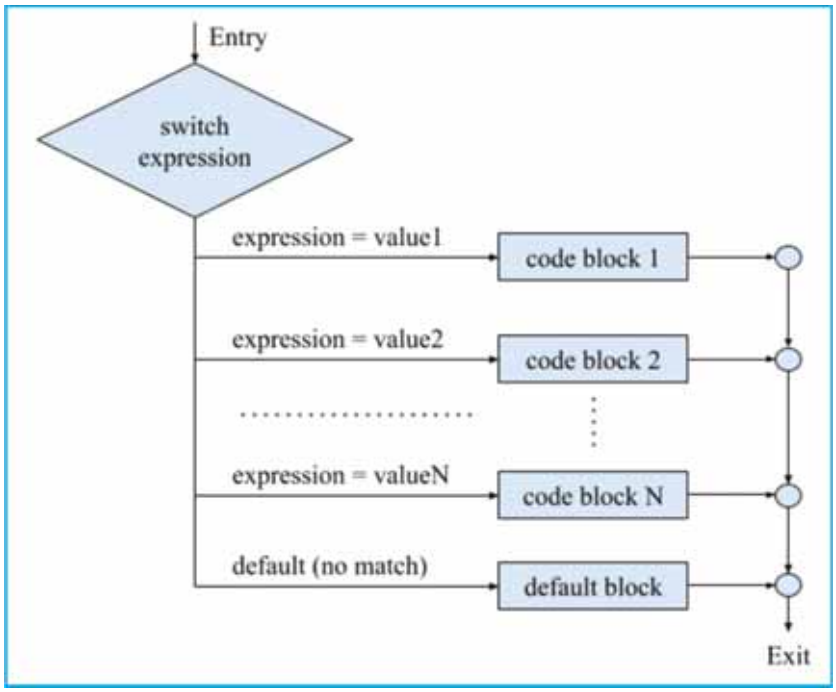
```
switch (expression)
{
    case value1:
        code block1
        break;
    case value2:
        code block2
        break;
      .....
      .....
    case valueN:
        code blockN
        break;
    default:
        default code block (optional)
}
```

switch વિધાનના વિવિધ અંગોની લાક્ષણિકતા નીચે મુજબ છે:

- **પદાવલી (Expression) :** *switch* સ્ટેટમેન્ટમાં પદાવલીની કિંમત integer, character કે enum ડેટા ટાઇપ હોવી જોઈએ, જેના પરથી નક્કી થશે કે કયા કેસ બ્લોકનો અમલ થશે.
- **બહુવિધ કેસ લેબલ્સ (Multiple *case* Labels) :** આપણે *switch* ની અંદર ઘણાં કેસ લેબલને વ્યાખ્યાયિત કરી શકીએ છીએ, જેમાં દરેક કેસ પછી એક અચલ મૂલ્ય (constant value) હોય છે. જ્યારે *switch* નું મૂલ્ય કોઈ કેસ લેબલના મૂલ્ય સાથે મેચ થાય છે, ત્યારે અનુરૂપ કોડ બ્લોકનો અમલ થાય છે.

- ***break* કીવર્ડ :** *break* વિધાન *switch* બ્લોકની અંદર મહત્વપૂર્ણ છે. એકવાર મેચ થતો કેસ મળી જાય અને તેનો કોડ અમલ થઈ જાય, પછી *break* વિધાન *switch* ને સમાપ્ત કરે છે. *break* વિના, ત્યારબાદના કેસનો કોડ પણ અમલી થશે, જે અનિચ્છનીય છે. જો *break* ભૂલી જવાય તો તે ઘણીવાર અનિચ્છનીય ભૂલ સાબિત થાય છે.
- ***default* કેસ (વૈકલ્પિક પરંતુ ભલામણપાત્ર) :** *default* કેસ વૈકલ્પિક છે પરંતુ તે ઇચ્છનીય છે. જ્યારે આપેલા કેસમાંથી એક પણ સાચો કેસ ન મળે ત્યારે આ કેસનો અમલ થાય છે.

આકૃતિ 10.4માં *switch* વિધાનનું કાર્ય ફ્લોચાર્ટ સાથે દર્શાવેલ છે.



આકૃતિ 10.4 : *switch* વિધાનનો ફ્લોચાર્ટ

ચાલો *switch* વિધાનને ઉદાહરણ સાથે સમજીએ.

switch વિધાનનું ઉદાહરણ

```
/* Program to illustrate use of Switch statement. It prints the
corresponding weekday based on a given number. */

#include <stdio.h>
int main() {
    int day = 3;

    switch(day) {
        case 1:
            printf("Monday");
            break;
        case 2:
            printf("Tuesday");
            break;
        case 3:
            printf("Wednesday");
```

```
            break;
        default:
            printf("Invalid day");
    }
    return 0;
}

Result:
Wednesday
```

આ C પ્રોગ્રામ day નામના ચલના મૂલ્યના આધારે અઠવાડિયાના દિવસનું નામ પ્રિન્ટ કરવા માટે *switch* વિધાનનો ઉપયોગ કરે છે. day ચલનું મૂલ્ય 3 પર સેટ કરેલું છે, તેથી પ્રોગ્રામ સામ્યતા શોધવા માટે દરેક *case* ની તપાસ કરે છે. જ્યારે તે case 3 પર પહોંચે છે, ત્યારે તે "Wednesday" પ્રિન્ટ કરે છે અને *break* નિવેદનનો ઉપયોગ કરીને *switch* માંથી બહાર નીકળી જાય છે. જો day નું મૂલ્ય એક પણ *case* સાથે મેચ ન થાય, તો *default* વિભાગ અમલમાં મુકાય છે અને "Invalid day" પ્રિન્ટ કરે છે. આ પ્રોગ્રામને સોમવાર (Monday) થી રવિવાર (Sunday) સુધીના અઠવાડિયાના દરેક દિવસને આવરી લેતા સાતેય *case* નો સમાવેશ કરવા માટે વિસ્તૃત કરી શકાય છે.

પુનરાવર્તન વિધાનો (Looping Statements)

C પ્રોગ્રામિંગમાં પુનરાવર્તન (લૂપિંગ) વિધાનો મૂળભૂત નિયંત્રણ પ્રવાહ સંરચનાઓ છે જે આપણને કોડના એક બ્લોકને વારંવાર અમલ કરવાની મંજૂરી આપે છે. જ્યારે આપણે એક જ કાર્ય ઘણી વખત કરવું હોય અને તે જ કોડ ફરીથી લખવો ન પડે, ત્યારે તે આવશ્યક છે. કલ્પના કરો કે આપણે 1 થી 100 સુધીના નંબરો પ્રિન્ટ કરવાની જરૂર છે; 100 વાર *printf()* લખવાને બદલે, આપણે તે કાર્યક્ષમ રીતે કરવા માટે લૂપનો ઉપયોગ કરી શકીએ છીએ. આ વિધાનો પુનરાવર્તિત કાર્યોને સ્વચાલિત કરવામાં મદદ કરે છે, જે આપણા પ્રોગ્રામને વધુ સંક્ષિપ્ત બનાવે છે.

લૂપીંગ વિધાનોના પ્રકાર

C ત્રણ પ્રકારના લૂપિંગ વિધાનો પ્રદાન કરે છે, જેમાંથી દરેક અલગ-અલગ પરિસ્થિતિઓ માટે યોગ્ય છે:

1. ***for* લૂપ :** *for* લૂપ એ એક પૂર્વ-પરીક્ષણ લૂપ (pre-test loop) છે, જેનો ઉપયોગ સામાન્ય રીતે ત્યારે થાય છે જ્યારે આપણે અગાઉથી જાણતા હોઈએ છીએ કે આપણે કોઈ કાર્યને કેટલી વાર પુનરાવર્તિત કરવા માંગીએ છીએ.
2. ***while* લૂપ :** *while* લૂપ પણ એક પૂર્વ-પરીક્ષણ લૂપ છે, પરંતુ તે ત્યારે વધુ યોગ્ય છે જ્યારે પુનરાવર્તનોની સંખ્યા અજ્ઞાત હોય અને તે કોઈ ચોક્કસ શરત પૂરી થવા પર આધાર રાખે છે.
3. ***do-while* લૂપ :** *do-while* લૂપ એ પશ્ચાત-પરીક્ષણ લૂપ (post-test loop) છે, જેનો અર્થ છે કે તેની શરતનું મૂલ્યાંકન લૂપનું બોડી ઓછામાં ઓછી એક વાર અમલ થયા પછી કરવામાં આવે છે.

ચાલો આપણે દરેક લૂપના અગત્યના લક્ષણો વિષે ચર્ચા કરીએ.

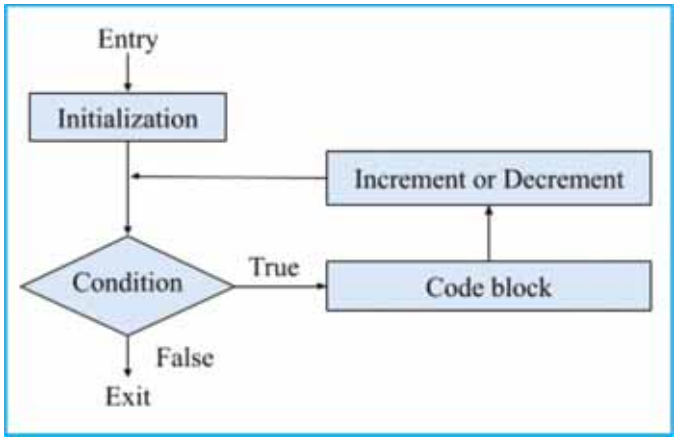
for લૂપ (for Loop)

for લૂપ એ એક પૂર્વ-પરીક્ષણ લૂપ છે જેનો ઉપયોગ સામાન્ય રીતે ત્યારે થાય છે જ્યારે આપણે કોઈ કાર્યને પુનરાવર્તિત કરવા માટે નિશ્ચિત સંખ્યામાં પુનરાવર્તનો ધ્યાનમાં રાખીએ છીએ. તે પ્રવેશ-નિયંત્રણ લૂપ (entry-controlled loop) તરીકે પણ ઓળખાય છે. તે સંક્ષિપ્ત છે અને શરૂઆત (initialization), શરત તપાસ (condition checking), અને વધારો/ઘટાડો (increment/decrement) ને એક લાઈનમાં જ ધરાવે છે.

for વિધાનની વાક્યરચના નીચે પ્રમાણે છે:

```
for (initialization; condition; increment/decrement) {
    code-block;  /* body of for loop */
}
```

આકૃતિ 10.5 ફ્લોચાર્ટની મદદથી for લૂપનું કાર્ય સમજાવે છે.



આકૃતિ 10.5 : *for* લૂપનો ફ્લોચાર્ટ

for લૂપ સંબંધિત મુખ્ય બાબતો

- **શરૂઆત (Initialization) :** આ ભાગનો લૂપની શરૂઆતમાં ફક્ત એક જ વાર અમલ થાય છે, જ્યાં લૂપને નિયંત્રિત કરતા ચલને મૂલ્ય આપવામાં આવે છે (જેમ કે i = 0;).
- **શરત (Condition) :** આની ચકાસણી દરેક પુનરાવર્તન (iteration) પહેલાં કરવામાં આવે છે. જો તે સાચી થાય, તો લૂપનું બોડી અમલ થાય છે; જો તે ખોટી થાય, તો લૂપ સમાપ્ત થઈ જાય છે.
- **વધારો/ઘટાડો (Increment/Decrement) :** આ ભાગનો લૂપનું બોડી અમલ થયા પછી દરેક પુનરાવર્તનના અંતે અમલ થાય છે. વધારો અથવા ઘટાડાની આ પ્રક્રિયા લૂપ નિયંત્રણ ચલને અપડેટ કરે છે, એટલે કે તેને પછીની શરત તપાસ માટે તૈયાર કરે છે. આ ચક્ર ત્યાં સુધી ચાલુ રહે છે જ્યાં સુધી શરત ખોટી ન થઈ જાય. શરત ખોટી થાય ત્યારે લૂપ સમાપ્ત થાય છે અને નિયંત્રણ લૂપ પછીના આગલા વિધાન પર જાય છે.

for લૂપનું ઉદાહરણ

```
/* Program to print 1 to 5 numbers using a for Loop. */

#include <stdio.h>
int main() {
    int i;
    for(i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
Result:
1 2 3 4 5
```

આ પ્રોગ્રામ 1 થી 5 સુધીની સંખ્યાઓ પ્રિન્ટ કરવા માટે *for* લૂપનો ઉપયોગ કરે છે. લૂપની શરૂઆત (i = 1) સાથે શરૂ થાય છે, જે પ્રારંભિક કિંમત સેટ કરે છે. શરત (i <= 5) સુનિશ્ચિત કરે છે કે લૂપ ત્યાં સુધી ચાલે જ્યાં સુધી i ની કિંમત 5 કરતાં નાની અથવા બરાબર હોય. દરેક પુનરાવર્તન પછી, i++ વિધાન i ની કિંમતમાં 1 નો વધારો કરે છે. લૂપની અંદર, *printf()* દ્વારા i ની વર્તમાન કિંમતને પ્રિન્ટ કરવામાં આવે છે. જ્યારે ચલ i ની કિંમત 6 થઈ જાય છે, ત્યારે શરત (i <= 5) ખોટી પડે છે અને લૂપ સમાપ્ત થઈ જાય છે.

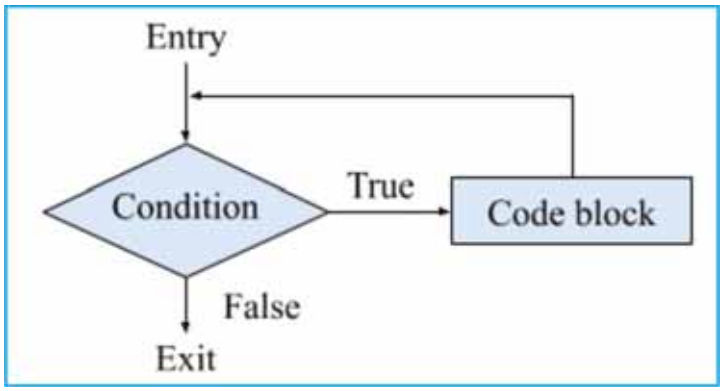
while લૂપ (while Loop)

while લૂપ પણ એક પૂર્વ-પરીક્ષણ લૂપ છે, પરંતુ તે ત્યારે વધુ યોગ્ય છે જ્યારે પુનરાવર્તનોની સંખ્યા અજ્ઞાત હોય અને તે કોઈ ચોક્કસ શરત પૂરી થવા પર આધાર રાખતી હોય. જ્યાં સુધી તેની શરત સાચી રહે છે, ત્યાં સુધી લૂપ ચાલુ રહે છે. *while* લૂપને પ્રવેશ-નિયંત્રિત લૂપ તરીકે પણ ઓળખવામાં આવે છે.

while વિધાનની વાક્યરચના નીચે પ્રમાણે છે:

```
while (condition) {
    code-block;      /* body of while loop */
}
```

શરતનું મૂલ્યાંકન દરેક પુનરાવર્તન પહેલાં કરવામાં આવે છે. જો તેનું મૂલ્યાંકન સાચું થાય, તો લૂપની બોડીનો અમલ થાય છે, નહીં તો લૂપ સમાપ્ત થાય છે. આકૃતિ 10.6 ફ્લોચાર્ટ દ્વારા *while* લૂપની કાર્ય પધ્ધતિ દર્શાવે છે.



આકૃતિ 10.6 : *while* લૂપનો ફ્લોચાર્ટ

ચાલો, નીચેના પ્રોગ્રામ દ્વારા *while* લૂપનું કાર્ય સમજીએ

while લૂપનું ઉદાહરણ

```
/* Program to illustrate use of while loop.
   Program will read and print numbers until a user enters 0.
*/
#include <stdio.h>
int main(){
    int num;

    printf("Enter a number (0 to quit): ");
    scanf("%d", &num);

    while (num != 0) {
        printf("You entered: %d \n", num);
        printf("Enter a number (0 to quit): ");
        scanf("%d", &num);
    }

    printf("Program exited as you have pressed 0.");
    return 0;
}
```

```
Result:

Enter a number (0 to quit): 1
You entered: 1
Enter a number (0 to quit): 5
You entered: 5
Enter a number (0 to quit): 0
Program exited as you have pressed 0.
```

આ પ્રોગ્રામ યુઝર પાસેથી નંબરો સ્વીકારવાનું ચાલુ રાખવા માટે *while* લૂપનો ઉપયોગ કરે છે જ્યાં સુધી વપરાશકર્તા 0 દાખલ ન કરે. તે યુઝરને એક નંબર દાખલ કરવા માટે કહીને શરૂ થાય છે અને *scanf()* નો ઉપયોગ કરીને ઇનપુટ વાંચે છે. જો દાખલ કરેલ નંબર 0 ન હોય, તો તે નંબરને *printf()* નો ઉપયોગ કરીને પ્રિન્ટ કરે છે અને બીજું ઇનપુટ માંગે છે. આ ચક્ર ત્યાં સુધી ચાલુ રહે છે જ્યાં સુધી ઇનપુટ 0 ન હોય. એકવાર યુઝર 0 દાખલ કરે, પછી શરત num != 0 (નંબર 0 નથી) ખોટી પડે છે, તેથી લૂપ અટકી જાય છે અને પ્રોગ્રામ એક સંદેશ પ્રિન્ટ કરે છે કે તે બહાર નીકળી ગયો છે.

do-while લૂપ (do-while Loop)

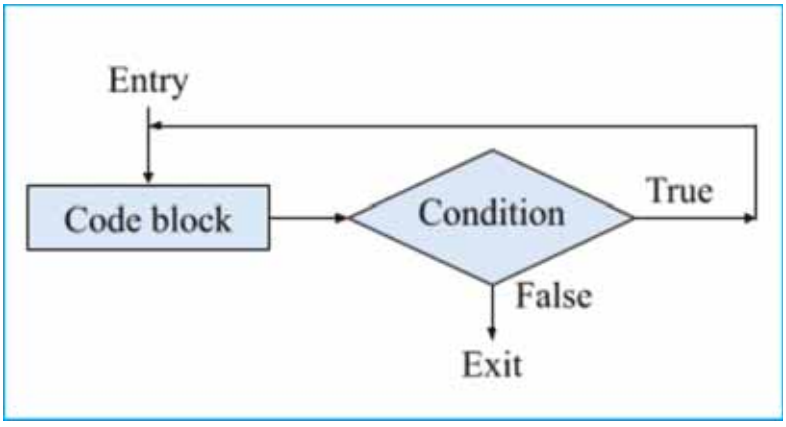
C ભાષામાં *do-while* લૂપ એક એવી લૂપ છે જેમાં કોડના બ્લોકનો ઓછામાં ઓછો એક વાર અમલ થાય, કારણ કે શરતનું મૂલ્યાંકન લૂપના બોડી પછી કરવામાં આવે છે. આપેલ શરત સાચી હોય ત્યાં સુધી તે અમલ થવાનું ચાલુ રાખે છે. આ લૂપ એવી પરિસ્થિતિઓ માટે ઉપયોગી છે જ્યાં શરત તપાસતા પહેલા કોડ ચાલવો જરૂરી હોય. ઉદાહરણ તરીકે, તેનો ઉપયોગ મેનૂ (menu) દર્શાવવા માટે થઈ શકે છે, અને ત્યારબાદ, યુઝરના ઇનપુટના આધારે પ્રોગ્રામનો બાકીનો ભાગ ચાલે છે. *do-while* લૂપની વાક્યરચના નીચે પ્રમાણે છે:

```
do {
    code-block;     /* body of do-while loop */
} while (condition);
```

શરતનું મૂલ્યાંકન દરેક પુનરાવર્તન પછી કરવામાં આવે છે. જો શરતનું મૂલ્યાંકન સાચું થાય, તો લૂપની અંદરના કોડનો ફરીવાર અમલ થાય છે, નહીં તો લૂપ સમાપ્ત થાય છે.

do-while લૂપને નિર્ગમ-નિયંત્રિત (exit-controlled) તરીકે વર્ગીકૃત કરવામાં આવે છે કારણ કે શરતનું મૂલ્યાંકન લૂપના બોડીના અમલ પછી કરવામાં આવે છે. તેનાથી વિપરીત, *for* લૂપ અને *while* લૂપને પ્રવેશ-નિયંત્રિત લૂપ ગણવામાં આવે છે કારણ કે લૂપની અંદરનો કોડ ચાલે તે પહેલાં શરતોનું મૂલ્યાંકન કરવામાં આવે છે.

આકૃતિ 10.7 *do-while* લૂપનું કાર્ય ફ્લોચાર્ટ સાથે દર્શાવે છે.



આકૃતિ 10.7 : *do-while* લૂપનો ફ્લોચાર્ટ

ચાલો, નીચેના પ્રોગ્રામ દ્વારા *do-while* લૂપનું કાર્ય સમજીએ.

do-while લૂપનું ઉદાહરણ

```
/* Program to illustration  use of do-while.
   Program performs Addition or Subtraction based on user choice. */
#include <stdio.h>
int main() {
    int n1=10, n2=5;
    int choice;

    do {
            printf("\n1. Addition \n2. Subtraction \n3. Exit\n");

            printf("Enter your choice: ");
            scanf("%d", &choice);

            if(choice == 1)
                  printf("Addition (n1+n2) = %d \n", n1+n2);

            if(choice == 2)
                  printf("Subtraction (n1-n2) = %d \n", n1-n2);

    } while (choice != 3);

    printf("You have pressed 3. Exiting Program...");
    return 0;
}
```

Result:

```
1. Addition
2. Subtraction
3. Exit
Enter your choice: 1
Addition (n1+n2) = 15

1. Addition
2. Subtraction
3. Exit
Enter your choice: 2
Subtraction (n1-n2) = 5

1. Addition
2. Subtraction
3. Exit
Enter your choice: 3
You have pressed 3. Exiting Program...
```

આ પ્રોગ્રામ યુઝરની પસંદગીના આધારે વારંવાર ગાણિતિક ક્રિયાઓ કરવા માટે *do-while* લૂપનો ઉપયોગ દર્શાવે છે. તે n1 = 10 અને n2 = 5 એમ બે ચલ વાપરે છે અને પછી ત્રણ વિકલ્પો સાથેનો એક મેનૂ દર્શાવે છેઃ Addition (સરવાળો), Subtraction (બાદબાકી), અને Exit (બહાર નીકળો). *do-while* લૂપની અંદર પ્રોગ્રામ યુઝર પાસેથી ઈનપુટ લે છે અને અનુરૂપ ઓપરેશન કરે છેઃ જો યુઝર 1 દાખલ કરે, તો તે n1 અને n2 નો સરવાળો પ્રિન્ટ કરે છે, જો 2 દાખલ કરે, તો તે તેમની બાદબાકી પ્રિન્ટ કરે છે. જ્યાં સુધી યુઝર 3 દાખલ ન કરે ત્યાં સુધી લૂપ ચાલુ રહે છે, આ સમયે પ્રોગ્રામ એક સંદેશ પ્રિન્ટ કરે છે અને સમાપ્ત થાય છે. *do-while* નો ઉપયોગ એ સુનિશ્ચિત કરે છે કે જો બહાર નીકળવાનો વિકલ્પ 3 સૌથી પહેલા દાખલ કરવામાં આવે તો પણ મેનૂ ઓછામાં ઓછું એક વાર દર્શાવવામાં આવે છે.

જમ્પિંગ વિધાનો (Jumping Statements)

C ભાષામાં જમ્પિંગ વિધાનોનો ઉપયોગ સામાન્ય ક્રમિક પ્રવાહને પ્રોગ્રામના નિયંત્રણને અવગણીને એક ભાગમાંથી બીજા ભાગમાં સ્થાનાંતરિત કરવા માટે થાય છે. આ વિધાનો ચોક્કસ પધ્ધતિના આધારે અમલના પ્રવાહને બદલવામાં મદદ કરે છે.

ચાલો, આપણે જમ્પિંગ વિધાનો *break, continue* અને *goto* ને ઉદાહરણ સાથે સમજીએ.

break

break નો ઉપયોગ લૂપ અથવા *switch-case* તરત જ સમાપ્ત કરવા માટે થાય છે. જ્યારે તેનો અમલ થાય છે, ત્યારે નિયંત્રણ લૂપ અથવા *switch* પછીના સ્ટેટમેન્ટ પર સ્થાનાંતરિત થાય છે. નીચે આપેલ C કોડનું અવલોકન કરોઃ

```
for (int i = 1; i <= 5; i++) {
    if  (i == 3)
        break;
    printf("%d ", i);
}
/* output: 1 2  */
```

આ C કોડમાં *for* લૂપ 1 થી 5 સુધીના નંબરો માટે લખેલી છે. જો કે, લૂપની અંદર એક *if* શરત છે જે તપાસે છે કે i નું મૂલ્ય 3 છે કે નહીં. જ્યારે આ શરત સાચી બને છે, ત્યારે *break* નો અમલ થાય છે, જે લૂપને તરત જ સમાપ્ત કરી દે છે. પરિણામે, લૂપ ફક્ત i = 1 અને i = 2 માટે જ ચાલે છે, અને તે મૂલ્યો પ્રિન્ટ કરે છે. જ્યારે i = 3 થાય છે, ત્યારે *break* લૂપના વધુ અમલને અટકાવે છે, તેથી નંબરો 3, 4, અને 5 પ્રિન્ટ થતા નથી. તેથી, કોડનો આઉટપુટ 1 2 છે.

continue

continue વિધાનનો ઉપયોગ લૂપના વર્તમાન પુનરાવર્તનને (current iteration) છોડીને, પછીના પુનરાવર્તન (next iteration) પર જવા માટે થાય છે. આ વિધાનમાં, લૂપ સમાપ્ત થતી નથી; તેમાં માત્ર અત્યારનું પુનરાવર્તન અવગણવામાં આવે છે.

continue નું ઉદાહરણ

```
for (int i = 1; i <= 5; i++) {
    if  (i == 3)
        continue;
    printf("%d ", i);
}
/* Output: 1 2 4 5 */
```

આ કોડમાં, લૂપની અંદર એક *if* વિધાન તપાસે છે કે શું ચલ i ની કિંમત 3 છે. જો શરત સાચી પડે છે, તો *continue* વિધાન અમલમાં આવે છે, જે વર્તમાન પુનરાવર્તનને છોડી દે છે અને *printf()* ને અમલમાં લાવ્યા વિના પછીના પુનરાવર્તન પર જાય છે. પરિણામે, નંબર 3 પ્રિન્ટ થતો નથી. અન્ય તમામ નંબરો (1, 2, 4, 5) સામાન્ય રીતે પ્રિન્ટ થાય છે. તેથી, આ કોડ પ્રિન્ટ કરશે: 1 2 4 5.

goto

goto વિધાનનો ઉપયોગ પ્રોગ્રામના લેબલ કરેલા ભાગ પર જવા માટે થાય છે. આધુનિક પ્રોગ્રામિંગમાં તેનો ઉપયોગ ટાળવામાં આવે છે, કારણ કે તે કોડને અસંરચિત (unstructured) અને ગૂંચવાડાભર્યો બનાવી શકે છે.

goto નું ઉદાહરણ

```
int RollNo = 1;
if   (RollNo == 1)
     goto ABC;
printf("My name is Purv");
ABC:
printf("My name is Mudra");

/* Output: My name is Mudra */
```

આ C કોડ *goto* વિધાનના ઉપયોગને દર્શાવે છે. ચલ RollNo ને 1 કિંમત આપવામાં આવે છે. *if* શરત તપાસે છે કે શું RollNo == 1 છે, જે સાચું છે, તેથી પ્રોગ્રામ *goto* ABC; નો અમલ કરે છે. આનાથી સીધું ABC: લેબલ પર જમ્પ થાય છે, અને આઉટપુટ તરીકે "My name is Mudra" પ્રિન્ટ થાય છે. *printf*("My name is Purv"); વાળી લાઈન છોડી દેવામાં આવે છે અને તેનો અમલ થતો નથી.

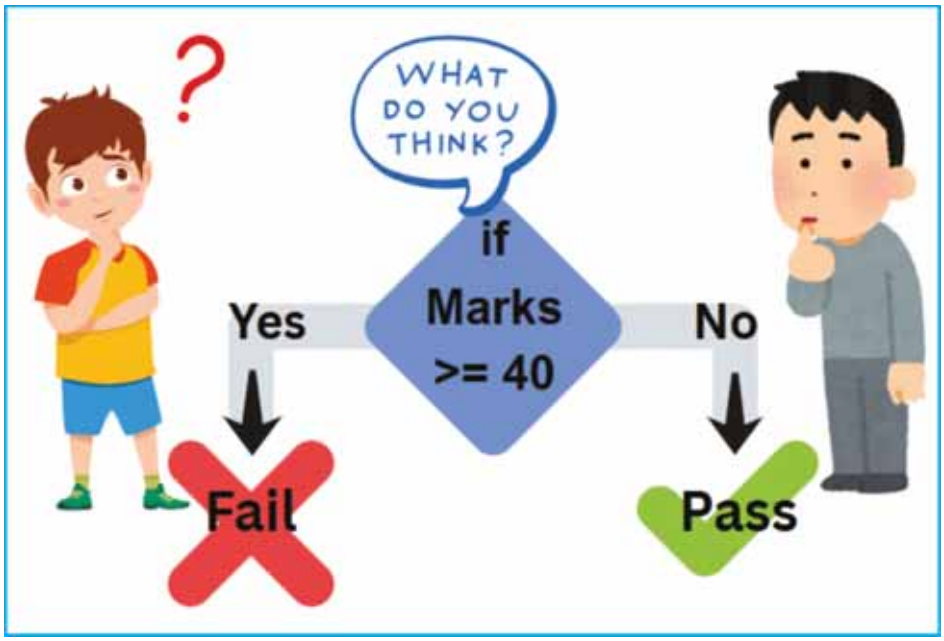
સારાંશ

આ પ્રકરણમાં, આપણે C પ્રોગ્રામિંગમાં આવશ્યક નિર્ણય લેવાની નિયંત્રણ સંરચનાઓ વિશે જાણ્યું, જે ચોક્કસ શરતોના આધારે પ્રોગ્રામના પ્રવાહને સંચાલિત કરવા માટે મહત્વપૂર્ણ છે. આપણે નિર્ણય લેવાના વિધાનો જેવા કે if, if-else, લેડર else-if અને switch નો અભ્યાસ કર્યો. આ વિધાનો પ્રોગ્રામરોને તાર્કિક માર્ગો બનાવવાની મંજૂરી આપે છે જે વિવિધ ઇનપુટ અને પરિસ્થિતિને જરૂર પ્રમાણે પ્રતિભાવ આપે છે, જેનાથી વાસ્તવિક દુનિયાની નિર્ણય પ્રક્રિયાઓનું નિરૂપણ કરવામાં મદદ મળે છે. આપણે લૂપિંગ વિધાનો જેવા કે for, while અને do-while નો પણ અભ્યાસ કર્યો. જેનો ઉપયોગ કાર્યોને પુનરાવર્તિત કરવા માટે થાય છે. વધુમાં, આપણે શીખ્યા કે break, continue અને goto જેવા જમ્પિંગ વિધાનોનો ઉપયોગ પ્રોગ્રામની અંદર નિયંત્રણના પ્રવાહને કેવી રીતે સંચાલિત કરવા માટે કરી શકાય છે. આ નિયંત્રણ સંરચનાઓને સમજી અને લાગુ કરીને, પ્રોગ્રામરો વાસ્તવિક દુનિયાની વિવિધ પરિસ્થિતિ પ્રમાણે પ્રતિભાવ આપતા ફ્લેક્ષિબલ અને કાર્યક્ષમ C પ્રોગ્રામ્સ બનાવી શકે છે.

સ્વાધ્યાય

1. C પ્રોગ્રામિંગના નિર્ણય લેવાની સંરચનાઓની યાદી બનાવો.
2. C માં if અને switch વિધાન વચ્ચેનો મૂળભૂત તફાવત સમજાવો. એવા સંજોગોની ચર્ચા કરો જ્યાં દરેકનો ઉપયોગ કરવો વધુ યોગ્ય રહેશે.
3. નેસ્ટેડ-if ના ખ્યાલનું વર્ણન કરો. એક વ્યવહારુ ઉદાહરણ આપો જ્યાં નેસ્ટેડ-if જરૂરી હોય.

4. નીચે આપેલા ચિત્રમાં શું ખોટું છે તે ઓળખો:



5. C પ્રોગ્રામિંગમાં while અને do-while લૂપની તુલના કરો. દરેક લૂપ પ્રકાર માટે, તેના મુખ્ય ઉપયોગને સમજાવો અને તેની વાક્યરચના દર્શાવતું એક નાનું કોડ સહિત ઉદાહરણ આપો. ઉપરાંત, કયા સંજોગોમાં while અને do-while લૂપનો ઉપયોગ કરવો જોઈએ તે યોગ્ય ઉદાહરણ સાથે સમજાવો.

6. C પ્રોગ્રામિંગમાં લૂપીંગ વિધાનોનો ઉપયોગ સમજાવો.

7. break અને continue વિધાનો લૂપની કામગીરીને કેવી રીતે પ્રભાવિત કરે છે? બંને માટે એક ઉદાહરણ આપો જે while લૂપની અંદર તેના પ્રભાવને દર્શાવે.

8. switch વિધાનનો ઉપયોગ યોગ્ય C પ્રોગ્રામ સાથે સમજાવો.

9. switch માં default નો ઉપયોગ શું છે?

10. C પ્રોગ્રામિંગમાં gotoની ભૂમિકા પર સંક્ષિપ્ત નોંધ લખો

11. **સાચું કે ખોટું જણાવો.**

(1) C પ્રોગ્રામિંગમાં if વિધાન શરત સાચી હોય ત્યારે જ કોડનો બ્લોક અમલ કરે છે.

(2) if-else વિધાનમાં else બ્લોક હંમેશા અમલ થાય છે, ભલે આપેલી શરત સાચી હોય કે ખોટી.

(3) for લૂપનો ઉપયોગ ત્યારે થાય છે જ્યારે પુનરાવર્તનની સંખ્યા પહેલેથી જ જાણીતી હોય.

(4) do-while લૂપ બોડીનો અમલ કરતા પહેલાં જ શરત ચકાસે છે.

(5) continue વિધાનનો ઉપયોગ પ્રોગ્રામમાંથી તરત જ બહાર નીકળવા માટે થાય છે.

12. **ખાલી જગ્યાઓ પૂરો.**

(1) C પ્રોગ્રામિંગમાં _____ નિર્ણય લેતું વિધાન આપેલી શરત સાચી હોય ત્યારે જ કોડ બ્લોકનો અમલ કરવા માટે ઉપયોગમાં લેવાય છે.

(2) if-else નિવેદનમાં, જો if શરત ખોટી હોય, તો _____ બ્લોકની અંદરનો કોડ અમલમાં આવે છે.

(3) switch બ્લોકની અંદર, દરેક અચલ મૂલ્ય _____ લેબલ પછી દર્શાવવામાં આવે છે.

(4) _____ લૂપ બોડીનો અમલ કરતા પહેલાં શરત ચકાસે છે.

(5) _____ વિધાનનો ઉપયોગ લૂપ અથવા switch બ્લોકમાંથી તરત બહાર નીકળવા માટે થાય છે.

13. બહુવિકલ્પી પ્રશ્નો. સૌથી યોગ્ય જવાબ પસંદ કરો.

(1) નીચેનામાંથી કઈ પદાવલીનું મુલ્ય સાચું (true) થશે?

(a) (10 - 10) (b) (0) (c) (2 * 0) (d) (7 % 2)

(2) જો C પ્રોગ્રામિંગમાં if વિધાનની શરત (3 - 3) હોય તો શું થશે?

(a) if ની અંદરનો કોડ અમલી થશે (b) else ની અંદરનો કોડ અમલી થશે

(c) કમ્પાઈલર ત્રુટી દર્શાવશે (d) 0 પ્રિન્ટ થશે

(3) નીચેના પૈકી કયું વિધાન લૂપનું હાલનું પુનરાવર્તન છોડીને આગળનું પુનરાવર્તન ચાલુ રાખવા માટે વપરાય છે?

(a) break (b) exit (c) continue (d) goto

(4) C ભાષામાં નીચેના પૈકી કઈ લૂપ ઓછામાં ઓછી એક વખત તો ચોક્કસ ચાલે છે?

(a) for લૂપ (b) while લૂપ (c) do-while લૂપ (d) switch

(5) નીચેના C કોડનો આઉટપુટ શું હશે?

```
int i;
for(i = 0; i < 3; i++) {
    printf("%d ", i);
}
```

(a) 1 2 3 (b) 0 1 2 (c) 0 1 2 3 (d) અનંત લૂપ

(6) જ્યારે પુનરાવર્તનની સંખ્યા પહેલેથી જ નક્કી હોય ત્યારે કઈ લૂપ સૌથી યોગ્ય ગણાય છે?

(a) switch (b) do-while (c) for (d) if

(7) break વિધાનનો લૂપની અંદર શો હેતુ છે?

(a) એક પુનરાવર્તન અવગણવું (b) પ્રોગ્રામનો અમલ બંધ કરવો

(c) હાલની લૂપમાંથી બહાર નીકળવું (d) લૂપની શરૂઆતમાં જવા

(8) switch પદાવલીમાં નીચેનામાંથી કયો ડેટા પ્રકાર યોગ્ય છે?

(a) float (b) double (c) int (d) string

(9) નીચેનામાંથી કઈ લૂપ પ્રવેશ-નિયંત્રિત છે?

(a) do-while (b) goto (c) while (d) break

(10) નીચનામાંથી કયું માન્ય લૂપીંગ વિધાન છે?

(a) select (b) return (c) switch (d) for

પ્રાયોગિક સ્વાધ્યાય

1. આપેલ સંખ્યા ધન, ઋણ કે શૂન્ય છે તે તપાસવા માટે if-else વિધાનનો ઉપયોગ કરીને એક પ્રોગ્રામ લખો.
2. લેડર if-else નો ઉપયોગ કરીને ગ્રેડ ગણતરી કરતો પ્રોગ્રામ લખો. માર્ક્સ સ્વીકારો અને નીચેના નિયમો મુજબ ગ્રેડ દર્શાવોઃ
   - 80 અથવા તેથી વધુ માર્ક્સ માટે ગ્રેડ A
   - 60 થી 79 માર્ક્સ માટે ગ્રેડ B
   - 40 થી 59 માર્ક્સ માટે ગ્રેડ C
   - 40 કરતાં ઓછા માર્ક્સ માટે ગ્રેડ D
3. switch-case નો ઉપયોગ કરીને સાદું કેલ્ક્યુલેટર બનાવવા માટેનો પ્રોગ્રામ લખો. પ્રોગ્રામ બે સંખ્યાઓ અને એક ઓપરેટર (+, –, *, /) સ્વીકારે છે, switch દ્વારા સંબંધિત ક્રિયા કરે છે અને પરિણામ દર્શાવે છે.
4. એક પ્રોગ્રામ લખો જે સંખ્યા વાંચે અને for loop નો ઉપયોગ કરીને તેનું ગુણાકાર ટેબલ દર્શાવે.
5. 1 થી 10 સુધીની સંખ્યાઓ દર્શાવતી લૂપ લખો. 5 દર્શાવવાનું ટાળો (સૂચન : continue નો ઉપયોગ કરો) અને જો સંખ્યા 8 થાય તો લૂપ બંધ કરો (સૂચન : break નો ઉપયોગ કરો).