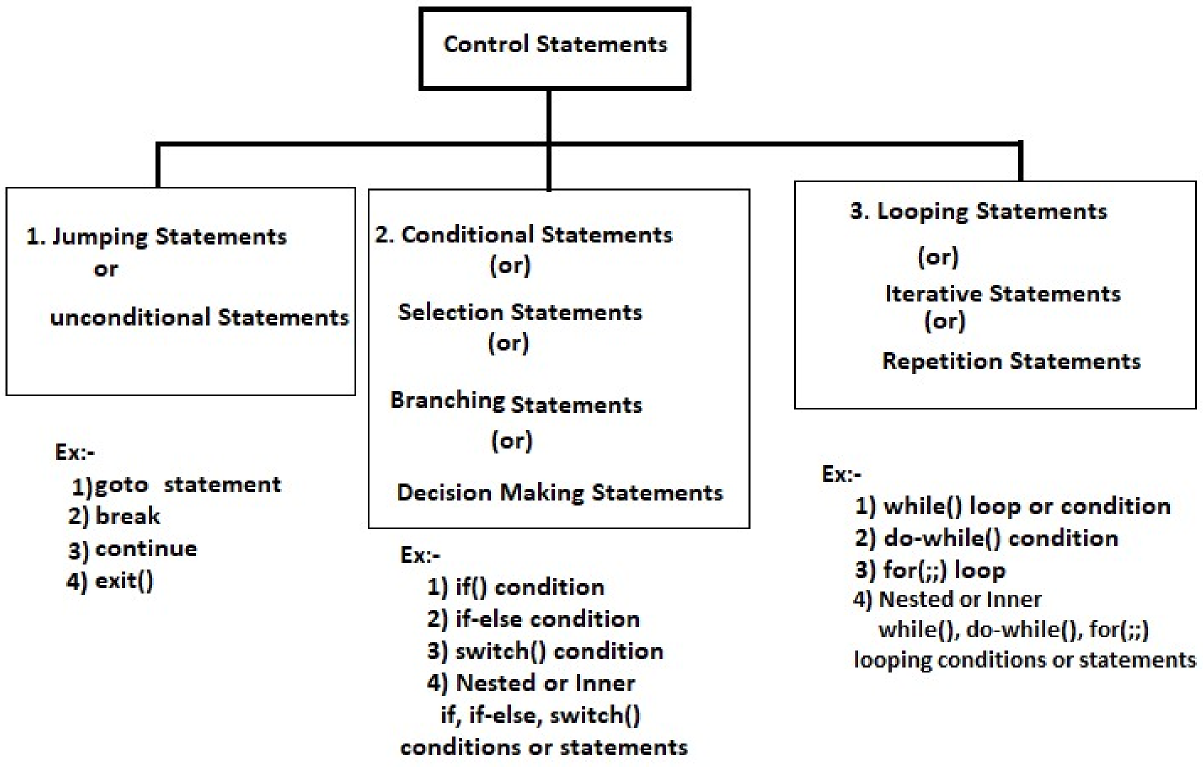
Unit II: Control Structures

Control structures in C are fundamental constructs that dictate the flow of execution in a program. They allow you to make decisions, repeat actions, and control the flow of your code. Here are the primary types of control structures in C:



Control Statements
1. Jumping Statements or unconditional Statements
2. Conditional Statements (or) Selection Statements (or) Branching Statements (or) Decision Making Statements
3. Looping Statements (or) Iterative Statements (or) Repetition Statements
Ex:- 1)goto statement 2) break 3) continue 4) exit()
Ex:- 1) if() condition 2) if-else condition 3) switch() condition 4) Nested or Inner if, if-else, switch() conditions or statements
Ex:- 1) while() loop or condition 2) do-while() condition 3) for(;;) loop 4) Nested or Inner while(), do-while(), for(;;) looping conditions or statements

1. Jumping Statements (Or) Unconditional Statements:

These are

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

a) goto statement:

The `goto` statement in C is used to transfer control to a labeled statement within the same function. While it can make certain control flow situations easier to handle, its use is generally discouraged because it can lead to code that is difficult to read and maintain. Here's a simple example of how `goto` works:

- **Labels**: Labels are defined by an identifier followed by a colon (`:`). You can jump to these labels using `goto`.

- **Scope**: `goto` can only jump to labels within the same function, not across different functions.

A) Forward Jumping
Executed from top to bottom flow.

Syntax:

```
goto LableName;
 -------------
 -------------
 -------------
LableName:
-------------
-------------
-------------
```

Examples:

```
goto hello;
printf("HI");
hello:
printf("Hello MOM");
```

Output:

```
Hello MOM
```

Program Example:

```
#include <stdio.h>
void main(){
        // goto is predefined keyword

        // calling the lable

        goto mom; // mom is lable

        printf("Hello World");

        // mom LableName

        mom:

        printf("MOM IT Solutions");

    }
```

Output:

```
MOM IT Solutions
```

B) Backward Jumping:-

Executed from bottom to top flow.

Syntax:

```
LableName:
 ------------
 ------------
goto LableName;
-------------
-------------
-------------
```

Examples:

```
hello:
printf("Hi");

goto hello;
printf("Hello MOM");
```

Output:

```
HiHiHiHiHi ...
```

Program Example:

```
#include <stdio.h>

void main(){
    mom:
    printf("Hi");
    goto mom;
    printf("Hello MOM");
}
```

Output:

```
HiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHi

HiHiHiHiHiHiHiHiHiHiHiHi …
```

b) break:-

The ‘break’ is a predefined keyword, to break the black of the loop.

The `break` keyword in C is used to exit from loops (`for`, `while`, `do-while`) and `switch` statements. When a `break` statement is encountered, control is transferred to the statement immediately following the loop or `switch`. Here's how it works in different contexts:

- **Immediate Exit**: `break` causes an immediate exit from the loop or `switch` without executing any remaining iterations or cases.
- **Nested Loops**: If used inside nested loops, `break` only exits the innermost loop.
- **Fall-Through in Switch**: Without a `break`, execution will fall through to the next case in a `switch` statement.

```
#include<stdio.h>

void main(){

    int i=1,n=5;

    while (i<=n){

        if(i = = 3){
            i++;

            break;

        }

        printf("i=%d\n",i++);

    }

}
```

Output:

```
i=1
i=2
```

Program Link: https://onlinegdb.com/7bs6KHpqga

C) Continue:-

To check the condition again or To skip the bellow statements.

The `continue` keyword in C is used to skip the current iteration of a loop and proceed to the next iteration. When `continue` is encountered, the remaining statements in the loop's body are skipped, and control goes directly to the loop's next iteration. This can be used in `for`, `while`, and `do-while` loops.

- **Immediate Skipping**: When `continue` is executed, the rest of the loop body is skipped, and the next iteration starts immediately.
- **Nested Loops**: If `continue` is used inside a nested loop, it only affects the innermost loop.
- **Usage**: `continue` can make your code cleaner in scenarios where certain conditions necessitate skipping the remainder of the loop iteration.

```
#include<stdio.h>

void main(){

    int i=1,n=5;

    while(i<=n){

        if(i= =3){

            i++;

            continue;
        }
        printf("i=%d\n",i++);
    }
}
```

Output:

```
            i=1
            i=2
            i=4
            i=5
```

2. **Conditional (or) Selection (or) branching (or) Decision Making Statements:**

These are as follow.

a) if() condition
b) if-else condition
c) switch() condition
d) Nested or Inner if, if-else, switch() conditions or statements

A) if() condition:-

The `if` statement in C is used to make decisions in your program by executing a block of code only if a specified condition evaluates to true. It is a fundamental control structure that allows for conditional execution.

Syntax:

```
if(condition){
    statements or True Part
}
```

If the if(condition) is true then execute the "True Part" otherwise bellow statements are executed.

➔ Single statement {,} are optional.
➔ Multiple statements {,} are mandatory.

- **Condition Evaluation**: The condition inside the parentheses must evaluate to a non-zero (true) or zero (false) value. Any non-zero value is considered true.

- **Code Blocks**: You can use curly braces `{}` to define a block of code to execute if the condition is true. If you only have one statement to execute, the braces are optional:

```
if (condition)
    printf("Condition is true.\n");
```

- **Nested `if` Statements**: You can nest `if` statements inside one another for more complex decision-making:

```
if (condition1) {
    if (condition2) {
        // Code to execute if both conditions are true
    }
}
```

```
#include<stdio.h>
void main(){
  int a=1,b=10;
  if(a<b){
    printf("True: A<B\n");
  }
  if(a+9==b)
    printf("TRUE: A+9 == B");
}
```

Output:

```
False: A>B

TRUE

Done
```

B) if-else condition:-

"Every condition should return either 'TRUE' or 'FALSE' ".

The `if-else` statement in C is used for conditional branching, allowing your program to execute different blocks of code based on whether a specified condition is true or false. This structure is fundamental for controlling the flow of execution in your program.

<u>**Syntax:**</u>

```
if(condition){
  True part
}else{
  False part
```

```
                    }
```

→If the if() condition is **TRUE** then execute the "**True Part**"
→If the if() condition is **FALSE** then execute the "**False Part**".
→ The {,} are optional for single statements in True Part & False Part.

Program Example: https://onlinegdb.com/-Tu-N5-od

```
#include<stdio.h>
void main(){
  int a=1,b=10;
  if(a>b){
    printf("True\n");
  }else{
    printf("False: A>B\n");
  }
  if(a+9 == b)
    printf("TRUE\n");
  else
    printf("FALSE: A+10 == B\n");
  printf("Done");
}
```

Key Points:

1. **Condition Evaluation**: The condition in the `if` statement is evaluated as true (non-zero) or false (zero). If the condition is true, the block of code in the `if` part is executed; otherwise, the code in the `else` part runs.
2. **Code Blocks**: You can use curly braces `{}` to define a block of code for both `if` and `else`. If there's only one statement to execute, the braces are optional:

```
if (condition)
    printf("This is true.\n");
else
    printf("This is false.\n");
```

3. **Nested `if-else` Statements**: You can nest `if-else` statements to handle more complex conditions:

```
if (condition1) {
    // Code for condition1
} else {
    if (condition2) {
        // Code for condition2
    } else {
        // Code if neither condition is true
    }
}
```

C) switch() condition:-

The `switch` statement in C is a control structure that allows you to execute one block of code among multiple options based on the value of a variable or expression. It's particularly useful when you have a variable that can take on a limited set of values.

➔ It is mostly used for selecting a one case from multiple cases or items.

➔ If it is not matched with any case then automatically executed the default case or option.

Syntax:

```
switch(choice){
   case lable1:
         ----------;
         break;
   case lable2:
         ----------;
         break;
   case lable3:
         ----------;
         break;
   case lable4:
         ----------;
         break;
   .
   .
   .
   default:
         ----------;
}
```

→ Here case, default and break are predefined keywords.

→ If the choice is not match with any case,

at that time executed the default option.

→ The 'choice' is must be int or char data type.

& case lables also be must be int or char data type.

→ “Choice” not be float data type & string also.

Key Points:

1. **Expression Evaluation**: The `switch` statement evaluates the expression (e.g., a variable) and matches it against the `case` constants.
2. **Case Constants**: Each `case` must be a constant expression. You cannot use variables or non-constant expressions in a `case`.
3. **Break Statement**: The `break` statement is crucial in a `switch` statement. It prevents fall-through behavior, which occurs when control continues into the next case. If you omit the `break`, the program will execute the following cases until it encounters a `break` or the end of the `switch`:

```
switch (day) {
    case 1:
        printf("Monday\n");
    case 2:
        printf("Tuesday\n"); // Will print if day is 1
        break;
    // ...
}
```

4. **Default Case**: The `default` case is optional and executes if none of the `case` values match the expression. It’s like the `else` in an `if-else` structure.

Program Example: https://onlinegdb.com/5YaEm1r7k

```
#include <stdio.h>
void  main(){
   int choice; // declaration of variable
   printf("1.CSE Branch\n");
   printf("2.IT Branch\n");
   printf("3.ECE Branch\n");
   printf("Enter choice(1 to 3):");
   scanf("%d",&choice);
   switch(choice){
      case 1:
            printf("CSE Branch\n");
            break;
      case 2:
            printf("IT Branch\n");
            break;
      case 3:
```

```
                printf("ECE Branch\n");
                break;
            default:
                printf("Plz select the any case from 1 to 3");
        }

    }
```

Output:

```
            1.CSE Branch

            2.IT Branch

            3.ECE Branch

            Enter choice(1 to 3):1

            CSE Branch
```

Program Example2: **https://onlinegdb.com/8evXjiqMR**

```
#include <stdio.h>

void  main(){

  char choice; // declaration of variable

  printf("Enter choice(a to c):");

  scanf("%c",&choice);

  switch(choice){

    case 'a':

        printf("Apple\n");

        break;

    case 'b':

        printf("Banana\n");

        break;

    case 'c':
        printf("Cherry\n");

        break;

    default:

        printf("Plz select the any case from 'a' to 'c'");

  }

}
```

Key Points:

1. **Expression Evaluation**: The `switch` statement evaluates the expression (e.g., a variable) and matches it against the `case` constants.
2. **Case Constants**: Each `case` must be a constant expression. You cannot use variables or non-constant expressions in a `case`.
3. **Break Statement**: The `break` statement is crucial in a `switch` statement. It prevents fall-through behavior, which occurs when control continues into the next case. If you omit the `break`, the program will execute the following cases until it encounters a `break` or the end of the `switch`:

```
switch (day) {
    case 1:
        printf("Monday\n");
    case 2:
        printf("Tuesday\n"); // Will print if day is 1
        break;
    // ...
}
```

4. **Default Case**: The `default` case is optional and executes if none of the `case` values match the expression. It's like the `else` in an `if-else` structure.

D) Nested or Inner if, if-else, switch() conditions or statements:-

Nested `if` statements occur when you place one `if` statement inside another. This allows you to check multiple conditions sequentially.

Nested `if-else` Statements

You can also nest `if-else` statements to manage multiple layers of decision-making.

Nested if-else:

Syntax:

```
  if(condition1){
   ------------
   ------------
  }else{
      ------------
      ------------
      if(condition2){
      ---------------
      ---------------
      }else{
          ------------
          ------------
      }
  }
```

```
#include <stdio.h>

void  main(){
    int i=1,j=1,n=5;
    printf("1\n");
    if(i>n){
        printf("2\n");
    }else{
        printf("3\n");
        if(j>n){
            printf("4\n");
        }else{
            printf("5\n");
        }
        printf("6\n");
    }
    printf("7\n");
}
```

<u>Output:</u>

```
1
3
5
6
7
```

Nested `switch` Statements

You can also nest `switch` statements inside each other, which can be useful for handling multiple levels of options.

```
#include <stdio.h>


int main() {
    int mainChoice, subChoice;


    printf("Menu:\n");
    printf("1. Fruits\n");
    printf("2. Vegetables\n");
    printf("Enter your choice (1-2): ");
    scanf("%d", &mainChoice);


    switch (mainChoice) {
        case 1: // Fruits
            printf("Fruits Menu:\n");
            printf("1. Apple\n");
            printf("2. Banana\n");
            printf("Enter your choice (1-2): ");
            scanf("%d", &subChoice);


            switch (subChoice) {
                case 1:
                    printf("You selected Apple.\n");
                    break;
        case 2:
            printf("You selected Banana.\n");
            break;
        default:
            printf("Invalid fruit choice!\n");
    }
    break;

case 2: // Vegetables
    printf("Vegetables Menu:\n");
    printf("1. Carrot\n");
    printf("2. Broccoli\n");
    printf("Enter your choice (1-2): ");
    scanf("%d", &subChoice);

    switch (subChoice) {
        case 1:
            printf("You selected Carrot.\n");
            break;
        case 2:
            printf("You selected Broccoli.\n");
            break;
        default:
            printf("Invalid vegetable choice!\n");
    }
    break;
```

```
        default:

            printf("Invalid main choice!\n");

    }

    return 0;

}
```

Output:

- If you enter `1` for fruits and then `2` for banana: `You selected Banana.`
- If you enter `2` for vegetables and then `1` for carrot: `You selected Carrot.`
- If you enter an invalid choice: `Invalid main choice!` or `Invalid fruit/vegetable choice!`

3. **Looping (or) Iterative (or) Repetation Statements:-**

   These are as follow.

   a) while() loop or Condition
   b) do-while() condition
   c) for(;;) loop
   d) Nested or Inner while(),do-while() and for(;;) loop

A) while() loop or Condition:-

A `while` loop lets you repeat a block of code as long as a condition is true.

Syntax:

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

➔ if the while condition is true then the body of the while() condition is execute until it fails.

➔ If the while() condition has single statement the {,} are optional.

➔ If the while() condition has more than one statement the {,} are mandatory,

How It Works

1. **Check the Condition**: Before each loop, the condition is checked.
2. **Execute Code**: If the condition is true, the code inside the loop runs.
3. **Repeat**: After running the code, it checks the condition again.
4. **Stop**: The loop ends when the condition is false

Program Example: **https://onlinegdb.com/mF819zRZn**

```
// W.A.P to print 1 to 5 number by using while().
#include "stdio.h"
void main(){
   int i=1,n=5;
   while(i<=n){
      printf("i=%d\n",i++);
   }
}
Output:
i=1
i=2
i=3
i=4
i=5
```

B) do-while() condition:-

In C, a `do-while` loop is a control flow statement that executes a block of code at least once and then repeats the loop as long as a specified condition is true.

Key Points:

- The block of code inside the `do` will execute once before the condition is tested.
- After executing the code, the condition is checked. If it evaluates to true, the loop runs again.
- If the condition is false, the loop terminates.

Explanation:

1. The program prompts the user to enter a positive number.
2. It reads the input and prints it if the number is not `-1`.

3. The loop continues until the user inputs `-1`, which causes the loop to terminate.

Advantages:

- The `do-while` loop guarantees that the code inside the loop will run at least once, which can be useful for input validation scenarios.

Syntax:

```
do{
    ……………..;
    statements;
    ……………...;
}while(condition);
```

→ It execute the body without checking the condition first time.

Program Example: **https://onlinegdb.com/7Nh0FqMQJ**

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

Output:

```
i=1
i=2
i=3
i=4
i=5
```

C) for(;;) loop:-

In C, a `for(;;)` loop is an infinite loop. The `for` statement in C typically has three parts: initialization, condition, and increment. However, when all three parts are omitted, as in `for(;;)`, it creates a loop that has no termination condition and will run indefinitely unless interrupted by a `break`, a `return`, or an external event (like a user interrupt).

Syntax:

```
for(initialization; condition ; increment/decrement){
    statement-1;
    statement-2;
  .
  .
  .
  statement-n;
}
```

1. Initialization and increment/decrement parts are optional
2. Condition is mandatory
3. {,} are optional for single statement.

Execution:

1. Initialization(part1) is executing one time only after
2. Executing condition (part2)
3. If the condition is true thenn executing body of the for(;;) loop
4. Executing increment/decrement (part3) after
5. Agian checking the condition upto it fails the condition.

Program Example: **https://onlinegdb.com/YjVcMNlFg**

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

Output:

```
i=1
i=2
i=3
i=4
i=5
```

Note:

```
/* for(i=1;i<=n;i++){
        for(j=1;j<=i;j++){
          printf("%d ",j);
        }
        printf("\n");
      }
      */
```

D) Nested or Inner while(),do-while() and for(;;) loop:-
It is used for dependency on upper condition.

Nested loops in C involve placing one loop inside another. This can be done with `while`, `do-while`, and `for` loops. Here's how you can use these types of loops together:

<u>Inner while() or Nexted while() Loop:</u>

A while() contains one more while() loop.

Syntax:

```
while(condition-1){

    -------

    -------

    while(condition-2){

        ---------

        ---------


    }

    ---------

    ---------


}
```

Program Example: **https://onlinegdb.com/PZF3k2owb**

```
#include <stdio.h>

void main(){

   int i,j,n;

   i=1;

   n=5;

   while(i<=n){

      j=1;

      while(j<=n){

         printf("i=%d j=%d\n",i,j);

         j++;

      }

      i++;

   printf("\n");

   }

}
```

Output:

```
i=1 j=1

i=1 j=2

i=1 j=3

i=1 j=4

i=1 j=5


i=2 j=1

i=2 j=2

i=2 j=3
```

```
i=2 j=4

i=2 j=5


i=3 j=1

i=3 j=2

i=3 j=3

i=3 j=4

i=3 j=5


i=4 j=1

i=4 j=2

i=4 j=3

i=4 j=4

i=4 j=5


i=5 j=1

i=5 j=2

i=5 j=3

i=5 j=4

i=5 j=5
```

<u>Inner do-while() condition:</u>

Syntax:

```
do{
    -------
    -------
    do{
      ---------
      ---------
    }while(condition);
    ---------
    ---------
}while(condition);
```

Program Example:

```
#include <stdio.h>

void  main(){
    int i,j,n=5;
    i=1;
    do{
        j=1;
        do{
            printf("i=%d j=%d\n",i,j);
            j++;
        }while(j<=n);
        i++;
        printf("\n");
    }while(i<=n);

}
```

Inner or nested for(;;) loop: **https://onlinegdb.com/jN0B4RCwyv**

Syntax:

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

```
            --------------
            --------------
          }
```

Program Example:

```
#include <stdio.h>
void main(){
 int i,j,n=5;
  for(i=1;i<=n;i++){
    for(j=1;j<=i;j++){
      printf("%d ",j);
    }
    printf("\n");
 }
}
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Mixed Nested Loops

You can also mix different types of loops together. Here's an example using a `for` loop inside a `while` loop:

```
#include <stdio.h>

void main() {
    int i = 1;

    while (i <= 3) { // Outer while loop
        printf("Outer loop iteration %d:\n", i);

        for (int j = 1; j <= 3; j++) { // Inner for loop
            printf("  Inner loop iteration %d\n", j);
        }

        i++;
    }
}
```

Explanation:

- **Outer Loop**: This loop controls how many times the inner loop will run.
- **Inner Loop**: This loop runs to completion for each iteration of the outer loop.
- **Printing**: In each iteration, the program prints which loop is currently executing.

Use Cases:

- Nested loops are commonly used for:
  - **Multidimensional Arrays**: Accessing elements in matrices.
  - **Complex Iterations**: When you need to perform a repeated operation that depends on multiple factors.