CHAPTER 3:
CONTROL FLOW STATEMENTS

Control flow structures in C allow you to control the execution order of statements based on conditions. In this session, we'll cover decision-making using if-else statements, nested if-else and switch-case statements, various types of loops (while, do-while, and for loops), and the use of break and continue statements.
Absolutely, I'll provide you with examples of each control flow construct along with explanations.

1. Simple `if` Statement:

Syntax:

```
if (condition) {
   // Code to execute if condition is true
}
```

Explanation:
The simple `if` statement checks whether a given condition is true. If the condition is true, the code inside the curly braces is executed.

Example:

```
#include <stdio.h>

int main() {
   int number = 10;

   if (number > 0) {
      printf("The number is positive.\n");
   }

   return 0;}
```

2. `if-else` Statement:

Syntax:

```
if (condition) {
   // Code to execute if condition is true
} else {
   // Code to execute if condition is false
}
```

Explanation:

The `if-else` statement checks a condition and executes one block of code if the condition is true, and a different block of code if the condition is false.

Example:

```
#include <stdio.h>

int main() {
   int age = 15;

   if (age >= 18) {
      printf("You are eligible to vote.\n");
   } else {
      printf("You are not eligible to vote.\n");
   }

   return 0;
}
```

3. Nested `if` Statement:

Syntax:

```
if (condition_1) {
   // Code for condition_1 being true
   if (condition_2) {
      // Code to execute if condition_2 is true
   } else {
      // Code to execute if condition_2 is false
```

```
    }
} else {
    // Code to execute if condition_1 is false
}
```

Explanation:
A nested `if` statement is an `if` statement inside another `if` statement. It allows you to check multiple conditions in a hierarchical manner.

Example:

```
#include <stdio.h>

int main() {
    int num = 6;

    if (num > 0) {
        if (num % 2 == 0) {
            printf("Number is positive and even.\n");
        } else {
            printf("Number is positive and odd.\n");
        }
    } else {
        printf("Number is not positive.\n");
    }

    return 0;
}
```

4. `else if` Ladder:

Syntax:

```
if (condition_1) {
    // Code for condition_1 being true
} else if (condition_2) {
    // Code for condition_2 being true
} else {
    // Code if all conditions are false
}
```

Explanation:
An `else if` ladder allows you to test multiple conditions sequentially. The code block associated with the first true condition is executed.

Example:

```
#include <stdio.h>

int main() {
    int score = 85;

    if (score >= 90) {
        printf("Grade A\n");
    } else if (score >= 80) {
        printf("Grade B\n");
    } else if (score >= 70) {
        printf("Grade C\n");
    } else {
        printf("Grade F\n");
    }

    return 0;
}
```

5. `switch-case` Statement:

Syntax:

```
switch (expression) {
    case value_1:
        // Code to execute if expression matches value_1
        break;
    case value_2:
        // Code to execute if expression matches value_2
        break;
    // More cases...
    default:
        // Code to execute if no cases match expression
}
```

Explanation:
The `switch-case` statement provides a way to execute different code blocks based on the value of an expression. The `break` statement is used to exit the switch block.

Example:

```
#include <stdio.h>

int main() {
   int day = 3;

   switch (day) {
      case 1:
         printf("Monday\n");
         break;
      case 2:
         printf("Tuesday\n");
         break;
      case 3:
         printf("Wednesday\n");
         break;
      default:
         printf("Unknown day\n");
   }
   return 0;
}
```


3. Looping with while, do-while, and for Loops:
Loops in C allow you to execute a block of code repeatedly.

a. while loop:

```
while (condition) {
   // Code to execute as long as the condition is true
}
```

Example:

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

b. do-while loop:

```
do {
   // Code to execute at least once
} while (condition);
```

Example:

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

c. for loop:

```
for (initialization; condition; increment/decrement) {
   // Code to execute for each iteration
}
```

Example:

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

4. Break and Continue Statements:

a. break:

The `break` statement is used to exit a loop prematurely.

Example:

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

b. continue:

The `continue` statement is used to skip the rest of the current loop iteration and move to the next iteration.

Example:

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