

Conditional Execution and Loops in C

Outline

- 1 Conditional Execution
- 2 Nested `if` Statements
- 3 Loop
- 4 `break` and `continue`
- 5 Nested Loops
- 6 The `switch` Statement
- 7 Exercise

Conditional Execution

Conditional Execution in C

- **Conditional execution** allows a program to take different actions based on certain conditions
- Conditions are expressed using `if`, `else`, and `else if` statements
- Condition expressions must evaluate to `true (non-zero)` or `false (zero)`

if Statement

Syntax:

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

Example:

```
1 #include <stdio.h>  
2 int main(){  
3     int x = 10;  
4     if(x > 0){  
5         printf("Positive number\n");  
6     }  
7 }
```

if-else Statement

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

Example: if-else

```
1  #include <stdio.h>
2
3  int main(){
4      int age = 18;
5
6      if(age >= 18){
7          printf("Eligible to vote\n");
8      } else{
9          printf("Not eligible to vote\n");
10     }
11 }
```

else if Ladder

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


Example: else if Ladder

```
1  #include <stdio.h>
2
3  int main(){
4      int marks = 75;
5
6      if(marks >= 90){
7          printf("Grade A");
8      } else if(marks >= 75){
9          printf("Grade B");
10     } else{
11         printf("Grade C");
12     }
13 }
```

Boolean Algebra in `if` Statements

- There can be multiple conditions
- Need to perform Boolean algebra on these conditions, because `if` statement expects only a single value
- Boolean operations on multiple conditions evaluate to a single value (true or false)
- Boolean operators:
 - AND, `&&`: Code runs only if all conditions are true
 - OR `||`: Code runs if at least one condition is true
 - NOT `!`: Negates a condition (flips true to false, and vice-versa)

Example: Loan Eligibility (AND operator)

```
1  #include <stdio.h>
2
3  int main(){
4      int age, income;
5      scanf("%d %d", &age, &income);
6
7      if((age >= 18) && (income >= 20000)){
8          printf("Eligible for loan");
9      } else{
10         printf("Ineligible for loan");
11     }
12 }
```

Example: Age Check (AND operator)

```
1  #include <stdio.h>
2
3  int main(){
4      int age;
5      scanf("%d", &age);
6      if((age >= 13) && (age <= 19)){
7          printf("The user is a teenager");
8      } else{
9          printf("The user is not a teenager");
10     }
11 }
```

Example: Sports Eligibility (OR operator)

```
1  #include <stdio.h>
2
3  int main(){
4      int age, weight;
5      scanf("%d %d", &age, &weight);
6
7      if((age >= 16) || (weight >= 60)){
8          printf("Eligible");
9      } else{
10         printf("Not eligible");
11     }
12 }
```

Nested `if` Statements

Nested if Statements

Sometimes, it necessary to put an `if` statement inside another. This is called nested statements. Can have as many levels of nesting as necessary.

```
if(cond1){  
    // code that's executed if cond1 is true  
    if(cond2){  
        // executed if both cond1 and cond2 are true  
    } else{  
        // executed if cond1 is true and cond2 is false  
    }  
    // code that's executed if cond1 is true  
}
```

Example: Loan Eligibility (revisited)

```
1  #include <stdio.h>
2  int main(){
3      int age, income;
4      scanf("%d %d", &age, &income);
5
6      if(age >= 18){
7          if(income >= 20000){
8              printf("Eligible for loan");
9          } else{
10             printf("Not eligible: income too low");
11         }
12     } else{
13         printf("Not eligible: under 18");
14     }
15 }
```


Loop

Loops in C

- Loops are used to execute a block of code repeatedly.
- Types of loops in C:
 - `for` loop: when number of iterations is known
 - `while` loop: when condition is checked before each iteration
 - `do-while` loop: condition checked *after* executing loop body

In do-while loop, the body of the loop is always executed at least once.

for Loop

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

The elements (initialization, condition and update) inside the for keyword, can be omitted. For example,

- Initialization can be performed before the **for** keyword
- Condition and update can be moved inside the loop body
- **for(;;){...}** creates an infinite loop

Example: for Loop

```
1 #include <stdio.h>
2
3 int main(){
4     for(int i = 1; i <= 5; i++){
5         printf("%d ", i);
6     }
7 }
```

Example: Another Way to Construct for Loops

```
1  #include <stdio.h>
2
3  int main(){
4      int i = 1;
5      for(;;){
6          if(i>5){
7              break;    // exits the loop when i>5
8          }
9          printf("%d\n", i);
10         i++;
11     }
12 }
```

Example: Sum Odd Integers (if inside for)

```
1  #include <stdio.h>
2
3  int main(){
4      int i, sum = 0;
5      for(i = 1; i <= 11; i++){
6          if (i%2 == 1){
7              sum += i;
8          }
9      }
10     printf("%d", sum);
11 }
```

Example: Sum Odd Integers (No if statement)

```
1  #include <stdio.h>
2
3  int main(){
4      int i, sum = 0;
5      for(i = 1; i <= 11; i+=2){
6          sum += i;
7      }
8      printf("%d", sum);
9  }
```

while Loop

Syntax:

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

Example:

```
1 #include <stdio.h>  
2 int main(){  
3     int i = 1;  
4     while(i <= 5){  
5         printf("%d ", i);  
6         i++;  
7     }  
8 }
```


Example: Greatest Common Divisor (GCD)

- The GCD of two integers a and b is c if both a and b are divisible by c
- First, assume that the smaller number is the GCD
- Then check if both a and b are divisible by the assumed GCD. If not, then decrement the assumed value by 1
- Keep repeating this process until both a and b are found to be divisible

Example: GCD (cont.)

```
1  #include <stdio.h>
2
3  int main(){
4      int a, b, gcd;
5      scanf("%d %d", &a, &b);
6
7      if(a < b){
8          gcd = a;
9      } else{
10         gcd = b;
11     }
```

Continued in the next page

Example: GCD (cont.)

```
12  
13     while((a%gcd!=0) || (b%gcd!=0)){  
14         gcd--;  
15     }  
16  
17     printf("%d", gcd);  
18 }
```

do-while Loop

Syntax:

```
do{  
    // statements  
} while(condition);    // don't forget this semicolon
```

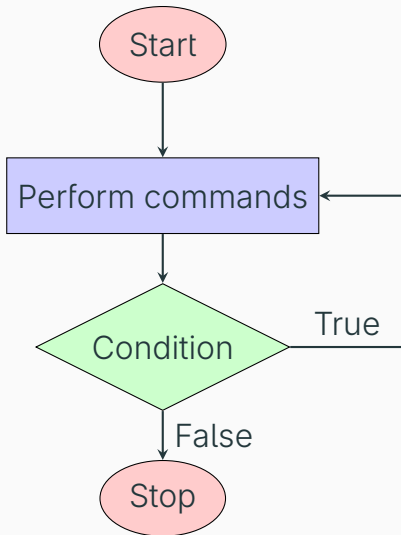
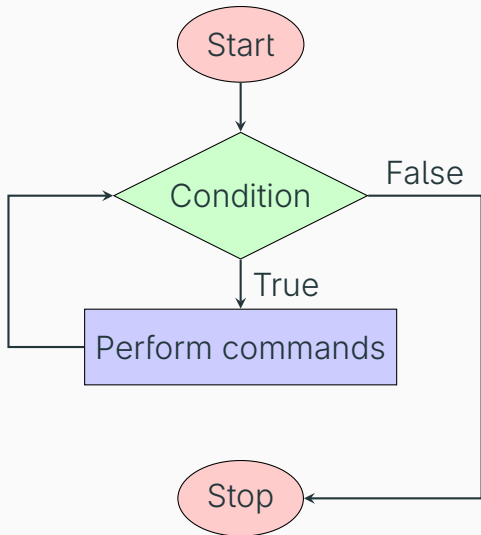
Example:

```
1 #include <stdio.h>  
2 int main(){  
3     int i = 1;  
4     do{  
5         printf("%d ", i);  
6         i++;  
7     } while(i <= 5);  
8 }
```

while vs do-while Loop

- `while`: condition checked *before* loop body
- `do-while`: condition checked *after* running the first iteration of the loop, so the loop runs at least once

Flowchart: While vs Do-While



Example: Input Validation Using do-while

```
1  #include <stdio.h>
2
3  int main(){
4      int input, pin = 12345;
5
6      do{
7          printf("Enter the pin: ");
8          scanf("%d", &input);
9      } while(input != pin);
10
11     printf("Access granted!\n");
12 }
```

break and continue

The break Statement

- The break statement immediately terminates the loop or switch statement in which it is encountered
- Control of the program then transfers to the statement immediately following the loop or switch
- It is commonly used to exit a loop prematurely based on a certain condition

Example: break

```
1  #include <stdio.h>
2
3  int main(){
4      for(int i = 1; i <= 10; i++){
5          if(i == 5){
6              break;      // Exit the loop when i is 5
7          }
8          printf("%d\n", i);
9      }
10     printf("\nLoop terminated");
11 }
```

The `continue` Statement

- The `continue` statement skips the remaining statements in the current iteration of a loop and proceeds to the next iteration
- It is used when you want to bypass certain parts of the loop's body for specific conditions without exiting the entire loop

Example: continue

```
1  #include <stdio.h>
2
3  int main(){
4      for(int i = 1; i <= 5; i++){
5          if(i == 3){
6              continue;    // Skip printing when i is 3
7          }
8          printf("%d\n", i);
9      }
10     printf("\nLoop finished");
11 }
```

Nested Loops

Nested Loops

- A **nested loop** means one loop inside another loop
- The **inner loop** executes completely for every single iteration of the **outer loop**
- Commonly used for:
 - Working with 2D data (like matrices)
 - Generating patterns
 - Performing repeated comparisons or calculations
- One can nest as many loops as necessary, but nesting more than two or three loops can lead to confusing or hard to understand codes

Nested Loops: Basic Syntax

```
for(initialization; condition; update){  
    for(initialization; condition; update){  
        // inner loop body  
    }  
    // outer loop body  
}
```

- You can nest `while` inside `for`, or any combination of loop types
- Be careful with initialization and loop conditions to avoid infinite loops

Example: (Non-nested, single loop) Multiplication table of 3

```
1  #include <stdio.h>
2
3  int main(){
4      for(int i=3, j = 1; j <= 10; j++){
5          printf("%d * %d = %d\n", i, j, i * j);
6      }
7  }
```


Example: (Nested loops) Multiplication tables of 1, 2 and 3

```
1  #include <stdio.h>
2
3  int main(){
4      for(int i = 1; i <= 3; i++){
5          for(int j = 1; j <= 10; j++){
6              printf("%d * %d = %d\n", i, j, i * j);
7          }
8          printf("\n");
9      }
10 }
```

Example: Triangle Pattern with *

```
1  #include <stdio.h>
2
3  int main(){
4      for(int i = 1; i <= 5; i++){
5          for(int j = 1; j <= i; j++){
6              printf("*");
7          }
8          printf("\n");
9      }
10 }
```

Example: Number Triangle

```
1  #include <stdio.h>
2
3  int main(){
4      for(int i = 1; i <= 4; i++){
5          for(int j = 1; j <= i; j++){
6              printf("%d ", j);
7          }
8          printf("\n");
9      }
10 }
```

Example: Number Pyramid

```
1  #include <stdio.h>
2
3  int main(){
4      int n_rows = 5;
5      for(int i = 1; i <= n_rows; i++){
6          for(int j = i; j < n_rows; j++){
7              printf(" ");    // initial spaces of each row
8          }
9          for(int k = 1; k <= i; k++){
10             printf("%d ", k);    // numbers of each row
11         }
12         printf("\n"); // newline after printing each row
13     }
14 }
```

The `switch` Statement

The `switch` Statement

- The `switch` statement allows multi-way branching based on the value of an expression
- It is an alternative to long chains of `if-else-if` statements
- It compares the given expression against multiple constant values given by (`case` labels)
- The `default` case handles unexpected or unmatched values

switch Statement General Syntax

```
switch(expression){  
    case value1:  
        // statements  
        break;  
    case value2:  
        // statements  
        break;  
    // ...  
    // ...  
    default:  
        // statements (optional)  
}
```

Why break is Necessary

- Without `break`, execution “falls through” to the next `case`
- This means all subsequent cases are executed until a `break` or the end of the switch
- To prevent this, use `break` at the end of each case

Example: Fall-Through Behavior of switch

```
int x = 2;  
switch(x){  
    case 1:  
        printf("A ");  
    case 2:  
        printf("B ");  
    case 3:  
        printf("C ");  
}  
// Output: B C
```

Example: Even - Odd

```
1  #include <stdio.h>
2
3  int main(){
4      int n;
5      scanf("%d", &n);
6      switch(n%2){
7          case 0:
8              printf("even\n");
9              break;
10         case 1:
11             printf("odd\n");
12             break;
13     }
14 }
```

Exercise

Exercise

Write C programs:

- ① To check whether a number (user input) is positive or negative or zero
- ② To check whether a year (user input) is a leap year
- ③ To check whether an integer is even or odd
- ④ To find the number of real-valued solution(s) to a quadratic equation, $(ax^2 + bx + c = 0)$. Take `a`, `b` and `c` as user inputs. Then calculate the value of the discriminant, then show the appropriate output
- ⑤ To print the first `n` (user input) natural numbers using a `for` loop. And another program to do the same using a `while` loop

Exercise (cont.)

- ⑥ To compute the sum of numbers from 1 to n using a `for` loop. And another program to do the same using a `while` loop
- ⑦ To find the factorial of an integer (user input)
- ⑧ To print the first n (user input) terms of the fibonacci series
- ⑨ To print the first n (user input) terms of the following arithmetic progression sequence: 1, 4, 7, 10, 13...
- ⑩ To repeatedly take user input and print its square, until a negative number is entered (use while loop)
- ⑪ To repeatedly take user input as exam marks and print the corresponding letter grade, until a negative number is entered (use `while` loop and `if` statement)
- ⑫ To find the GCD of two integers using the Euclidean algorithm

Exercise (cont.)

- 13 To find the LCM of two integers
- 14 To repeatedly take user input and print its square, until a negative number is entered (use `do-while` loop)
- 15 To repeatedly take user input as exam marks and print the corresponding letter grade, until a negative number is entered (use `do-while` loop)
- 16 To print the sum of the first n (user input) terms of the following arithmetic progression sequence: $1 + 4 + 7 + 10 + 13 \dots$
- 17 To print the first n (user input) terms of the following sequence: 1, 2, 4, 7, 11, 16...
- 18 To print the sum of the first n (user input) terms of the following series: $1 + 2 + 4 + 7 + 11 + 16 \dots$

Exercise (cont.)

- 19 To find all the prime numbers within a given range. The start and end integers of the range shall be user input
- 20 To print a right aligned triangle pattern with *, sample output:

```
      *
     * *
    * * *
   * * * *
  * * * * *
```

Exercise (cont.)

- 21 To generate a multiplication table up to 5×5 in grid format, sample output:

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

Exercise (cont.)

22 To generate an inverted number triangle, sample output:

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

Exercise (cont.)

- 23 To create a simple calculator using the `switch` statement. First, define a character variable (call it `op` for operator) using the `char` keyword, then use `scanf("%c", &op)`, the user shall input one of the following symbols: `+`, `-`, `*`, `/`. Then take two numbers (can be integers or floats) as user input. Finally, use the `switch` keyword to perform addition, subtraction, multiplication or division based on the input to `op`. If user inputs some unexpected character, then print `invalid input`