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:

```
#include <stdio.h>
int main(){
    int x = 10;
    if(x > 0){
        printf("Positive number\n");
    }
}
```

`if-else` Statement

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

Example: `if-else`

```c
#include <stdio.h>

int main(){
    int age = 18;

    if(age >= 18){
        printf("Eligible to vote\n");
    } else{
        printf("Not eligible to vote\n");
    }
}
```

else if Ladder

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

Example: `else if` Ladder

```c
#include <stdio.h>

int main(){
    int marks = 75;

    if(marks >= 90){
        printf("Grade A");
    } else if(marks >= 75){
        printf("Grade B");
    } else{
        printf("Grade C");
    }
}
```

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)

```c
#include <stdio.h>

int main(){
    int age, income;
    scanf("%d %d", &age, &income);

    if((age >= 18) && (income >= 20000)){
        printf("Eligible for loan");
    } else{
        printf("Ineligible for loan");
    }
}
```

Example: Age Check (AND operator)

```c
#include <stdio.h>

int main(){
    int age;
    scanf("%d", &age);
    if((age >= 13) && (age <= 19)){
        printf("The user is a teenager");
    } else{
        printf("The user is not a teenager");
    }
}
```

Example: Sports Eligibility (OR operator)

```c
#include <stdio.h>

int main(){
    int age, weight;
    scanf("%d %d", &age, &weight);

    if((age >= 16) || (weight >= 60)){
        printf("Eligible");
    } else{
        printf("Not eligible");
    }
}
```

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)

```c
#include <stdio.h>
int main(){
    int age, income;
    scanf("%d %d", &age, &income);

    if(age >= 18){
        if(income >= 20000){
            printf("Eligible for loan");
        } else{
            printf("Not eligible: income too low");
        }
    } else{
        printf("Not eligible: under 18");
    }
}
```

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 ommitted. For example,

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

Example: `for` Loop

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 5; i++){
        printf("%d ", i);
    }
}
```

Example: Another Way to Construct `for` Loops

```c
#include <stdio.h>

int main(){
    int i = 1;
    for(;;){
        if(i>5){
            break;    // exits the loop when i>5
        }
        printf("%d\n", i);
        i++;
    }
}
```

Example: Sum Odd Integers (`if` inside `for`)

```c
#include <stdio.h>

int main(){
    int i, sum = 0;
    for(i = 1; i <= 11; i++){
        if (i%2 == 1){
            sum += i;
        }
    }
    printf("%d", sum);
}
```

Example: Sum Odd Integers (No `if` statement)

```c
#include <stdio.h>

int main(){
    int i, sum = 0;
    for(i = 1; i <= 11; i+=2){
        sum += i;
    }
    printf("%d", sum);
}
```

while Loop

Syntax:

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

Example:

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

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.)

```c
#include <stdio.h>

int main(){
    int a, b, gcd;
    scanf("%d %d", &a, &b);

    if(a < b){
        gcd = a;
    } else{
        gcd = b;
    }
```

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
```

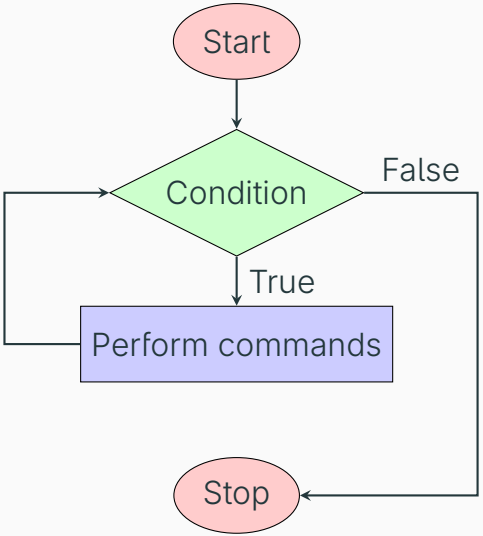
Example:

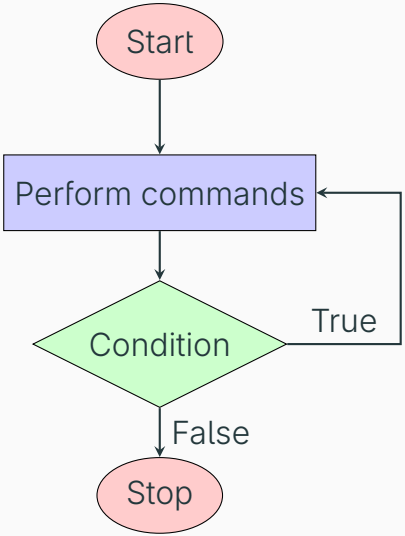
```
#include <stdio.h>
int main(){
    int i = 1;
    do{
        printf("%d ", i);
        i++;
    } while(i <= 5);
}
```

`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`

```c
#include <stdio.h>

int main(){
    int input, pin = 12345;

    do{
        printf("Enter the pin: ");
        scanf("%d", &input);
    } while(input != pin);

    printf("Access granted!\n");
    return 0;
}
```

`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

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 10; i++){
        if(i == 5){
            break;    // Exit the loop when i is 5
        }
        printf("%d\n", i);
    }
    printf("\nLoop terminated");
}
```

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

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 5; i++){
        if(i == 3){
            continue;    // Skip printing when i is 3
        }
        printf("%d\n", i);
    }
    printf("\nLoop finished");
}
```

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

```c
#include <stdio.h>

int main(){
    for(int i=3, j = 1; j <= 10; j++){
        printf("%d * %d = %d\n", i, j, i * j);
    }
    return 0;
}
```

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

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 3; i++){
        for(int j = 1; j <= 10; j++){
            printf("%d * %d = %d\n", i, j, i * j);
        }
        printf("\n");
    }
    return 0;
}
```

Example: Triangle Pattern with *

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 5; i++){
        for(int j = 1; j <= i; j++){
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

Example: Number Triangle

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 4; i++){
        for(int j = 1; j <= i; j++){
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Example: Number Pyramid

```c
#include <stdio.h>

int main(){
    int n_rows = 5;
    for(int i = 1; i <= n_rows; i++){
        for(int j = i; j < n_rows; j++){
            printf(" ");   // initial spaces of each row
        }
        for(int k = 1; k <= i; k++){
            printf("%d ", k);    // numbers of each row
        }
        printf("\n"); // newline after printing each row
    }
    return 0;
}
```

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`

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

Example: Even-Odd

```c
#include <stdio.h>

int main(){
    int n;
    scanf("%d", &n);
    switch(n%2){
        case 0:
            printf("even\n");
            break;
        case 1:
            printf("odd\n");
            break;
    }
    return 0;
}
```

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 intger (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...
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...

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`