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:

```c
#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
}
```

`if-else` Example

```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{
    ...
}
```

else if Ladder Example

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

Example: Loan Eligibility (revisited)

```c
#include <stdio.h>

int main(){
    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

for Loop Example

```c
#include <stdio.h>

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

for Loop Example (cont.)

```c
#include <stdio.h>

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

`while` Loop

Syntax:

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

Example:

```c
#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
```

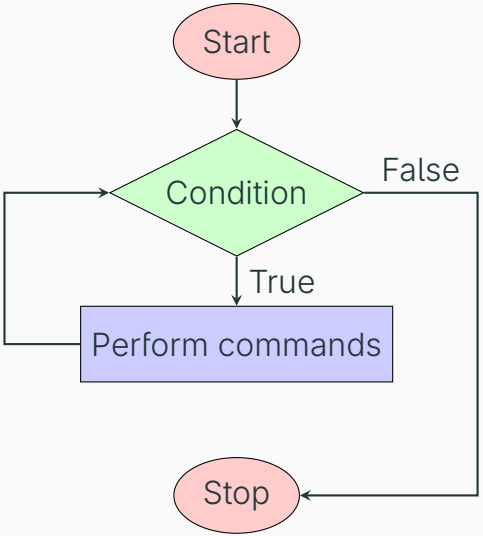
Example:

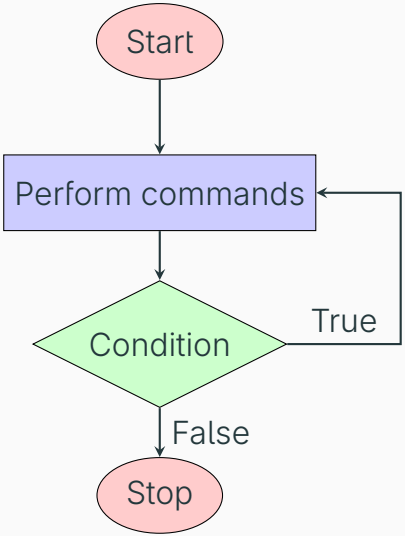
```c
#include <stdio.h>
int main(){
    int i = 1;
    do{
        printf("%d ", i);
        i++;
    } while(i <= 5);
}
```

`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





`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

break Example

```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

`continue` Example

```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

The `switch` Statement

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

Exercise (cont.)

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
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...

Exercise (cont.)

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
13. To find the LCM of two integers