Féidearthachtaí as Cuimse
Infinite Possibilities

Week 5 - Tutorial

OLLSCOIL TEICNEOLAÍOCHTA
BHAILE ÁTHA CLIATH
DUBLIN
TECHNOLOGICAL
UNIVERSITY DUBLIN

Programming - Week 5

Overview

- Revision: Decision making
  - If Statement
  - If / Else
  - Switch Statement
- While Loop
- Do While Loop
- For Loop

Revision: Decision making

- The flow of control in our programs is used break up the flow of execution using decision making, looping, and branching to execute particular blocks of code.
- This allows us to use logic to offer functionality in our programs.
- We can use control flow to make decisions based on different values and conditions.
- We will explore this further this tutorial.

Conditional Statements

- If Statement
- If / Else
- Switch Statement

Revision: Conditional statements

- A **conditional statement** in C is used to perform different actions or computations based on whether a certain condition or set of conditions is true or false.
- This allows the program to **make decisions** and execute code accordingly.
- Conditional statements are fundamental in **controlling the flow** of a program.

Revision: if Statement

- The **if statement** checks a condition and will execute a block of code only if the condition evaluates to true.

If the condition evaluates as true the code block will run

Syntax
```
if (condition) {
    // This code will be executed if condition is true
}
```

Advice:
Always use { and } braces around the statements inside the if statement.

Example_1.c > ...

```c
#include <stdio.h>

int main() {
    int age = 18;

    printf("Niteclub Example - The Roxbury\n");
    printf("Rules: only people 18 or over can enter....\n");

    if ( age >= 18 ) {
        printf("Welcome to the Roxbury\n");
    }

}

```

Revision: If / Else

- Instead of using 2 if statements we could use an if .. else block

Syntax:

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

Example_2a.c > ...

```c
#include <stdio.h>

int main() {
    int age;

    printf("Niteclub Example - The Roxbury\n");
    printf("Rules: only people 18 or over can enter....\n");

    printf("Hello, how old are you?:\n");
    scanf("%d", &age);

    if ( age >= 18 ) {
        printf("Welcome to the Roxbury\n");
    } else {
        printf("Sorry you can't enter, you are too young\n");
    }

    return 0;
}
```

Revision: Switch statement

- The switch statement is a control statement that allows you to test the value of a variable or expression against a list of values (called cases).
- It is often used as an alternative to multiple if-else conditions when checking a variable for equality with several constant values.

C Example_5.c > ...

```c
#include <stdio.h>

int main() {
    char grade = ' ';

    printf("Enter a grade: \n");
    scanf("%c", &grade);

    // Now use the switch statement
    switch(grade)
    {
        case 'A':
        {
            printf("Highest grade\n ");
            break;
        }
        case 'B':
        {
            printf("2nd Highest grade\n ");
            break;
        }
        case 'C':
        {
            printf("3rd Highest grade\n ");
            break;
        }
        default: {
            printf("Sorry, you failed...");
            break;
        } // end default
    }

    return 0;
}
```

Loops

- A loop in our program allows us to to perform a task a number of times.
- A condition is used to determine if the loop should run again.
- We will look at a number of different loops this week.

while loop

- The while loop is a control structure that allows you to repeat a block of code as long as a specified condition is true.
- The condition is evaluated at the start the loop iteration, if it's false the loop terminates.
- There is a possibility the while loop code block might never run.

```
while (condition) {

    // Code to execute in each iteration

}
```

A while loop will run if the condition evaluates as true.

The code block will run if the condition evaluates as true and when the block has completed the condition will be tested again.

The loop can:
- Never run (condition initially evaluates as false)
- Run once
- Run multiple times
- Run infinitely

while loop

- The condition is checked before each iteration. If it evaluates to true, the loop code block is executed. If it evaluates to false, the loop ends.
- If the condition first evaluates as false the loop never runs

```
while (condition) {

    // Code to execute in each iteration

}
```

while loop - example 1

- Create an int variable named i and store a value of 1.
- The condition for the while loop will check if i is <= to 10 for each iteration of the while loop.
- When the condition evaluates as false the loop ends and the program continues after the while loop structure.

Example_2.c > ...

```c
#include <stdio.h>

int main() {
    int i = 1;

    // Loop will continue while i is less than or equal to 10
    while (i <= 10) {
        printf("%d\n", i);
        i++;  // Increment i after each iteration
    }

    return 0;
}
```

while loop - example 2

- Create a program that will allow a user to enter a character.
- The program will end if the user enters a letter q
- The loop condition will check if the letter entered by the user is a letter q. The loop condition (c != ‘q’) will evaluate as false when a letter q is entered.

Example_3.c > ...

```c
#include <stdio.h>

int main() {
    char c;

    while (c != 'q') {
        printf("Enter a character: \n");
        scanf(" %c", &c);
    }

    return 0;
}
```

while loop - example 3

- Create a program that will allow a user to guess a number. The program will continue until the user guesses the correct number.
- Assumption: the user will only ever enter numbers as their guesses.

Example_4.c > ...

```c
#include <stdio.h>

int main() {
    int number = 47;
    int guess = 0;

    while (number != guess) {
        printf("Win a prize - guess the number\n");
        printf("Enter a guess: \n");
        scanf("%d", &guess);

        if (guess == number) {
            printf("Congratulations, you guessed correct\n");
        } else {
            printf("Sorry, try again....\n\n");
        }
    }

    return 0;
}
```

do-while loop

- The do-while loop is similar to the while loop
- There is one main difference. The do-while loop guarantees that the loop body will execute at least once, even if the condition is false.

Syntax:

```
do {
    // Code to execute in each iteration
} while (condition);

// You MUST place a semi-colon at the end of the do .. while
```

A do-while loop will run if the condition evaluates as true.

The code block will run and when the block has completed the condition will be tested again.

The loop can:
- Run once (guaranteed)
- Run multiple times
- Run infinitely

do-while loop - Example

- The do-while loop is a better choice for our guess a number program.
- The program will always run the guess a number at least once. This is specifically what the do-while loop is designed for.

Example_5.c > ...

```c
#include <stdio.h>

int main() {
    int number = 47;
    int guess = 0;

    do {
        printf("Win a prize - guess the number\n");
        printf("Enter a guess: \n");
        scanf("%d", &guess);

        if (guess == number) {
            printf("Congratulations, you guessed correct\n");
        } else {
            printf("Sorry, try again....\n\n");
        }
    } while (number != guess);

    return 0;
}
```

for loops

- A for loop is a control structure that allows you to **repeat a block of code a specified number of times.**
- The loop structure when we **know exactly how many times the loop should run.**

```
for (initialization; condition; update) {

    // Code to run in each iteration

}
```

for loops

- **Initialization:** this is the initialization of the loop control variable. This only happens once at the beginning of the loop.
- **Condition:** This is the expression that is checked before every iteration. The loop will continue as long as the condition evaluates as true. When the condition evaluates as false the loop terminates.
- **Update:** This statement is executed after each iteration of the loop (usually used to increment or decrement the loop control variable)

Syntax:

```
for (initialization; condition; update)
{

    // Code to run in each iteration

}
```

for loop - Example

- **Initialization:** set i to 1
- **Condition:** i is less than or equal to 10
- **Update:** Increment i by 1 for each iteration of the loop

Example_1.c > ...

```c
#include <stdio.h>

int main() {

    // This loop will print numbers from 1 to 10
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```

Break keywork

- The break keyword is used to break out a for loop, a while loop or a do-while loop.

Common Coding Interview Question

- What is the difference between a for loop, a while loop and a do-while loop?

Programming Pitfall

1. There is no semi-colon at the end of a while loop or for loop
2. You must be careful how you specify the termination condition in a loop.
3. Make sure that your condition is not false at the very start of a while loop or the code inside the loop will never execute.
4. Always use a whole number data type as your index variable, i.e., short, int or long. The default data type is an int

Questions