CS201 Introduction to Programming: Lecture 6 Summary

Key Concepts for Quiz Preparation

May 23, 2025

Repetition Structure: While Loop

This document summarizes Lecture 6 of CS201, focusing on the `while` loop in C programming. It covers key concepts, sample programs, overflow conditions, properties, flow chart, and tips for quiz preparation.

1 Repetition Structure (Loop)

Loops automate repetitive tasks, such as summing numbers or processing payroll. They eliminate the need for manual repetition (e.g., `cout « 1 + 2 + ... + 1000`), which is inefficient for large datasets.

2 While Loop

The `while` loop repeats statements as long as a condition is true.

2.1 Syntax

```
while (Logical Expression) {
    statement1;
    statement2;
    ...
}
```

- **Behavior**: Executes the block while the condition is `true`. Stops when the condition becomes `false`.
- **Braces**: Always use `{}` for clarity, even for a single statement.
- **Indentation**: Indent statements inside the loop for readability.

2.2 Example: Sum of First 1000 Integers

Objective: Calculate the sum of integers from 1 to 1000.

```
#include <iostream.h>
main() {
```

```
    int sum = 0, number = 1;
    while (number <= 1000) {
        sum = sum + number;     // Add number to sum
        number = number + 1;    // Increment number
    }
    cout << "The sum of first 1000 integers starting from 1 is "
       << sum;
}
```

- **How It Works**:
  1. Initialize `sum = 0`, `number = 1`.
  2. Loop: Add `number` to `sum`, increment `number`.
  3. Stop when `number = 1001` (condition `number <= 1000` becomes false).
  4. Output: `sum = 500500`.

3 Overflow Condition

- **Definition**: Occurs when a variable (e.g., `int`) stores a value exceeding its capacity (e.g., 32-bit limit: 2,147,483,647).
- **Consequences**:
  - **Run-time Error**: Program may crash.
  - **Incorrect Result**: Extra bits are discarded, leading to wrong values.
- **Example**: Summing numbers beyond 1000 (e.g., 10,000) may cause overflow if `sum` exceeds the `int` limit.

4 Sample Program 1: Sum with User-Defined Limit

Objective: Allow the user to input the upper limit for summing integers.

```
#include <iostream.h>
main() {
    int sum = 0, number = 1, upperLimit;
    cout << "Please enter the upper limit for which you want the
       sum ";
    cin >> upperLimit;
    while (number <= upperLimit) {
        sum = sum + number;
        number = number + 1;
    }
    cout << "The sum of first " << upperLimit << " integers is "
       << sum;
}
```

- **Benefit**: Reusable program for any upper limit without code changes.

5 Sample Program 2: Sum of Even Numbers

Objective: Sum even numbers up to a user-defined limit.

```
#include <iostream.h>
main() {
    int sum = 0, number = 1, upperLimit;
    cout << "Please enter the upper limit for which you want the
       sum ";
    cin >> upperLimit;
    while (number <= upperLimit) {
        if (number % 2 == 0) {
            sum = sum + number;  // Add only even numbers
        }
        number = number + 1;
    }
    cout << "The sum of even integers from 1 to " << upperLimit
       << " is " << sum;
}
```

- **Logic**: Use `number % 2 == 0` to check for even numbers.
- **Example**: For `upperLimit = 10`, sums $2 + 4 + 6 + 8 + 10 = 30$.

6 Sample Program 3: Factorial Calculation

Objective: Compute the factorial of a user-input number.

```
#include <iostream.h>
main() {
    int factorial = 1, number;
    cout << "Please enter the number for factorial ";
    cin >> number;
    while (number > 1) {
        factorial = factorial * number;
        number = number - 1;
    }
    cout << "The factorial is " << factorial;
}
```

- **How It Works**: For `number = 5`, computes 5! = 5 Œ 4 Œ 3 Œ 2 = 120.

7 Properties of While Loop

- **Zero or More Executions**: Loop may not execute if the condition is initially false (e.g., `upperLimit = 0`).
- **Termination**: Stops when the condition becomes false.
- **Infinite Loop**: Occurs if the condition never becomes false (e.g., missing `number = number + 1`).
- **Example of Infinite Loop**:

```
while (number <= 1000) {
    sum = sum + number;  // Missing number increment
}
```

8 Flow Chart

- **Structure**:
  - **Rectangle**: Marks the start of the `while` loop.
  - **Diamond**: Contains the condition (e.g., `number <= 1000`).
  - **True Path**: Leads to actions (e.g., `sum = sum + number`).
  - **Loop Back**: Actions reconnect to the condition.
  - **False Path**: Exits the loop.
- **Purpose**: Visualizes control flow.

9 Tips for Quiz Preparation

- Use **self-explanatory variable names** (e.g., `sum`, `number`).
- **Practice** writing and debugging `while` loop programs.
- Ensure the loop condition has an **exit** to avoid infinite loops.
- Understand **overflow** and the `int` limit.
- Always use **braces** `{}` in loops.

10 Practice Questions

1. Write a program to sum odd numbers from 1 to a user-defined limit.
2. Modify the factorial program to reject negative numbers.
3. Explain the effect of changing `number <= 1000` to `number < 1000`.
4. Calculate the output of Sample Program 2 for `upperLimit = 10`.
5. Draw a flowchart for Sample Program 3.
6. Identify and fix an infinite loop in the sum program.