Advanced Programming - Supplementary Examples

C++ Basic Structure, Declarations & Definitions

This document provides a set of examples and exercises to supplement the lecture material on C++ Basic Structure, Declarations & Definitions. It is designed to help you practice and deepen your understanding of the key concepts of lecture 2 such as expressions, operators, statements, iteration, functions, control flow, data types, declarations and definitions, and scope and namespaces.

Examples

Expressions and Operators

Combining Operators

```cpp
#include <iostream>
using namespace std;

int main() {
    // multiple variables can be defined in the same line
    // (as long as they have the same type)
    int a = 10, b = 20, c = 5;
    int result = (a + b) * c / (b - a);
    cout << "Result: " << result << endl;
    return 0;
}
```

Operator Precedence

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 5;
    // test yourself the different combinations of operators below
    int y = ++x * x--;
    //int y = --x * x++;
    //int y = x++ * --x;
    //int y = x-- * ++x;
    cout << "x: " << x << ", y: " << y << endl;
    return 0;
}
```

Control Flow

Nested Control Statements

```cpp
#include <iostream>
using namespace std;

int main() {
```

```
    // test for different values of x
    int x = 10;
    if (x > 5) {
        if (x < 15) {
            cout << "x is between 5 and 15" << endl;
        } else {
            cout << "x is greater than or equal to 15" << endl;
        }
    } else {
        cout << "x is less than or equal to 5" << endl;
    }
    return 0;
}
```

Functions

Function Overloading

Function overloading involves defining multiple functions with the same name in the same scope.

```
#include <iostream>
using namespace std;

int add(int a, int b) {
    return a + b;
}

double add(double a, double b) {
    return a + b;
}

// what do you think would happen when calling add...
int main() {
    cout << "add(3, 4): " << add(3, 4) << endl;
    cout << "add(3.5, 4.5): " << add(3.5, 4.5) << endl;
    return 0;
}
```

Recursion

Recursion involves defining a function that calls itself.

```
#include <iostream>
using namespace std;

int factorial(int n) {
    if (n <= 1) return 1;
    // notice the function calls itself in the else condition
    else return n * factorial(n - 1);
}

int main() {
    int number = 5;
    cout << "Factorial of " << number << " is " << factorial(number) << endl;
    return 0;
}
```