Introduction to C++

- C++ language
  - Structured programming
  - Object-oriented programming

Example

```cpp
#include <iostream>

int main()
{
    int integer1, integer2, result;
    std::cout << "Enter first integer\n";
    std::cin >> integer1;
    std::cout << "Enter second integer\n";
    std::cin >> integer2;
    result = integer1 / integer2;
    std::cout << "Result is " << result << std::endl;
    return 0;
}
```

Preprocessor directive to include input/output stream header file `<iostream>`. Function `main` appears exactly once in every C++ program. Function `main` returns an integer value. Left brace `{` begins function body. Corresponding right brace `}` ends function body. Statements end with a semicolon. Name `cout` belongs to namespace `std`. Stream insertion operator.
Memory Concepts

• Variables
  - Location in memory where value can be stored
  - Every variable has name, type, size and value
  - When new value placed into variable, overwrites previous value

```
std::cin >> integer1;
- Assume user entered 45

std::cin >> integer2;
- Assume user entered 7

res = integer1 / integer2;
```

Formulating Algorithms

• problem:
  *Develop a program that will compute the average of CS112 grades*

• Sentinel value
  - Indicates “end of data entry”
  - Sentinel chosen so it cannot be confused with regular input
Formulating Algorithms

• Top-down, stepwise refinement
• Many programs have three phases
  - Initialization
    • Initializes the program variables
  - Processing
    • Input data, adjusts program variables
  - Termination
    • Calculate and print the final results

Formulating Algorithms

• Refine the initialization phase
  Initialize total and counter to zero

• Processing
  Input, sum and count the grades -->
  Input the first grade (possibly the sentinel)
  While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
  Input the next grade (possibly the sentinel)

Formulating Algorithms

• Termination
  Calculate and print the class average -->
  If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
  Else
    Print “No grades were entered”
#include <iostream>
using std;
#include <iomanip>

int main(){
    int total, counter, grade;
    double average;
    total = counter = 0;
    cin >> grade;
    while ( grade != -1 ) {
        total += grade; counter++; cin >> grade;
    }
    if (counter) {
        average = static_cast<double>(total)/counter;
        cout << "Class average is " << setprecision(2) << average << endl;
    } else
        cout << "No grades were entered" << endl;
    return 0;
}

Review of Basic Object-Oriented C++ Programming
Introduction

- Object-oriented programming (OOP)
  - Encapsulates data (attributes) and functions (behavior) into packages called classes
- Information hiding
  - Class objects communicate across well-defined interfaces
  - Implementation details hidden within classes themselves

Classes

- User-defined types
  - Attributes: data members
  - Behaviors: member functions or methods
  - Class instance: object
- Member access specifiers
  - public
    - Accessible wherever object of class in scope
  - private
    - Accessible only to member functions of class
  - protected

Example

class Time {
    public:
        int hour;
        int minute;
        int second;
    private:
        int hour;
        int minute;
        int second;
    public:
        Time();
        Time(int, int, int);
        void setTime(int, int, int);
        void printTime();
    private:
        int hour;
        int minute;
        int second;
    protected:
        int hour;
        int minute;
        int second;
};
Constructor and Destructor Functions

- **Constructors**
  - Special member function
  - Initializes data members
  - Same name as class
  - Called when object instantiated
  - Several constructors
  - No return type

- **Destructors**
  - Performs “termination housekeeping”
  - Same name as class
  - Preceded with `~`
  - No arguments
  - Cannot be overloaded

Member Function Definition

- If member functions defined outside class:
  - Binary scope resolution operator (`::`)
    - “Ties” member name to class name
    - Uniquely identify functions of particular class
    - Different classes can have member functions with same name

- Format for defining member functions
  ```cpp
  ReturnType ClassName::MemberFunctionName() {
      ...
  }
  ```
  - Does not change whether function public or private

```cpp
class Time {
public:
    void setTime(int h, int m, int s)
    {
        hour = (h >= 0 && h < 24) ? h : 0;
        minute = (m >= 0 && m < 60) ? m : 0;
        second = (s >= 0 && s < 60) ? s : 0;
    }

    void printTime()
    {
        cout << hour << " : " << minute << " : " << second;
    }
};
```
Member Function Definition

- If member functions defined inside class
  - Do not need scope resolution operator, class name
  - Compiler attempts `inline`
    - Outside class, `inline` explicitly with keyword `inline`

Example

class Time {
public:
  Time();
  Time(int, int, int);
  void setTime(int, int, int);
  void printTime();
private:
  int hour;
  int minute;
  int second;
};

Objects of Class

- After class definition
  - Class name new type specifier

  Time sunset(6, 0, 0);
  Time arrayOfTimes[5];
  Time *pointerToTime;
  Time &dinnerTime = sunset;
Main Function

```c
int main() {
    Time t;
    t.printTime();
    t.setTime(13, 27, 6);
    t.printTime();
    t.setTime(99, 99, 99);
    t.printTime();
    return 0;
}
```