

## CISC 1600/1610 Computer Science I

### Functions: Overloading, Specifications

Professor Daniel Leeds  
dleeds@fordham.edu  
JMH 328A

## Function overloading

“Overloading” when multiple functions with same name but:

- different number of parameters
- different types of parameters

Compiler determines which function to use

2

## Overloaded averaging function

```
float average(int num1, int num2) {
    return (num1+num2)/2.0;
}

float average(int num1, int num2, int
num3) {
    return (num1+num2+num3)/3.0;
}
```

3

```
int main()
{
    int numInputs; float in1, in2, in3;
    cout << "How many inputs?";
    cin >> numInputs;
    if(numInputs==2) {
        cout << "Give 2 numbers: ";
        cin >> in1 >> in2;
        cout << "Average: "
            << average(in1,in2) << endl;
    } else {
        cout << "Give 3 numbers: ";
        cin >> in1 >> in2 >> in3;
        cout << "Average: "
            << average(in1,in2,in3) << endl;
    }
    return 0;
}
```

Overloaded average  
function in action

## Procedural abstraction

- Function name stands in for set of statements
- Can use a function without knowing how it is written

```
int a=abs(-5);
float b=sqrt(2);
```

5

## Procedural abstraction, continued

What do we need to know?

- Function name
- Inputs
- Outputs
- Results of performing function

} **Function  
declaration**

6

## Specifications

Preconditions:

- What is assumed to be true when function is called

Postconditions:

- What will be true after the function is called (presuming preconditions are met)
  - What values are returned
  - What call-by-reference parameters are changed
  - What other output is produced

7

## Example specification

- Include specs in comments of declaration

```
float sqrt(float inputNumber);  
// Precondition: inputNumber is a  
// positive float  
// Postcondition: Function returns  
// a float output such that output  
// is non-negative and  
// output*output=inputNumber
```

8