Functions: Specifications, void, Recursion, Overloading

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

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);

Procedural abstraction, continued

What do we need to know?
• Function name
• Inputs
• Outputs
• Results of performing function

Function declaration

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

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*output=inputNumber

void functions

• void function returns no value

Example definition:
void greetUser(string userName){
    cout << "Hello " << userName << endl;
    return;
}

Example call:
greetUser(userName);
NOT: cout<<greetUser(userName);}
Use of `return`;

- In `void` function, can use `return`;
- When evaluated, `return` terminates function

Recursion

When a function calls itself:
- Can be a simpler way to write a loop
- Can be used as a divide-and-conquer method

Alternate power function

```c
int power(int num, int expon)
{
    if(expon>0)
        return num*power(num,expon-1);
    else
        return 1;
}
```

Recursive function design

Must have:
- Base case(s) – to eventually stop recursion
- Simplified recursive calls – each new call must bring us closer to reaching base case(s)

```c
int funcC(int a);
int main() {
    int a;
    cout << "Enter a number: ";
    cin >> a;
    cout << funcC(a);
    return 0;
}
int funcC(int a) {
    if(a==0)
        return a;
    else
        return a+funcC(a-1);
}
```

Function overloading

“Overloading” when multiple functions with same name but:
- different number of parameters
- different types of parameters

Compiler determines which function to use
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;
}

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;
}