CISC 1600/1610
Computer Science I

Functions/modularity

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

Blocks of statements

Statements in a program are grouped:
• with curly braces { } for if, switch, and loops
• conceptually (with blank lines, indentations, and comments)

Good ----, world!

> ./timeGreetings
What is your name? Joe
What time is it? 0900
Good morning, Joe.
> ./timeGreetings
What is your name? Laura
What time is it? 1400
Good afternoon, Laura.
>

Code for timeGreetings.cpp

Get name and time

cout << "What is your name? ";
cin >> name;
cout << "What time is it? ";
cin >> time;

Output time-based greeting

• Outputs sub-divided into time-based blocks
if(time<noon)
  cout << "Good morning, " << name << endl;
else if(time<startEvening)
  cout << "Good afternoon," << name << endl;
else
  cout << "Good evening." << name << endl;

Write once, use repeatedly

cout << count << " mississippi\n";

Can print:
1 mississippi

Can print:
1 mississippi
2 mississippi
3 mississippi
Define operation once, use repeatedly

Circle area: \(3.14 \times r \times r\)

\[\text{float area}=3.14*r*r;\]

Functions

1. Identify a set of statements with a single keyword
2. Use single keyword to run the larger set of statements anywhere in your code

\[\text{float area}_r2=\text{circleArea}(2);\]

Defining a function

Similar to variable

• function declaration
  – must be declared before it is used
  – declaration provides overview of function behavior
• function definition
  – provides the statements performed by the function

Functions in your C++ file

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

float circleArea(float radius); // declaration

int main () {
    . . .
    float area_R2=\text{circleArea}(2); // usage
    . . .
}

float circleArea(float radius) { // definition
    float area=3.14*radius*radius;
    return area;
}
```

Function declaration

Establish:

• function name
• output type
• input types and names

```
return_type fcn_name(input_list);
```

```
float circleArea(float radius); // computes area of circle
```

Function definition

Provides the statements performed when function is used

```
return_type fcn_name(input_list){
    statement1;
    . . .
    statementN;
}
```

```
float circleArea(float radius){
    float area=3.14*radius*radius;
    return area;
}
```
### Function use – “function call”
- Names function to use
- Provides input arguments for the function
- (If appropriate) can assign output

```plaintext
float area_R2 = circleArea(2);
```
- Call types must be consistent with declaration and definition

### The return statement
- When function is “called”, information may be expected back
  ```plaintext
  float area_R2 = circleArea(2);
  ```
- `return` specifies what value to give the caller

### Alternate function declaration

```plaintext
return_type fcn_name(input_list);
```

**Example:**

```plaintext
float circleArea(float radius);
```

Only argument types **required** in declaration
But argument names **highly** recommended

### Call-declaration consistency
- Compiler forces match between call and declaration
  ```plaintext
  float final_price(int numItems, float single_cost);
  x = final_price(3.43,10);  // numItems*single_cost
  Will force type-conversion: 3.43->3, 10->10.000
  ```
- Does not check logical ordering of arguments
  ```plaintext
  int sum_range(int min, int max);
  a = sum_range(10,3);
  Will not re-order input: min=10, max=3
  ```

### Pre-defined functions

```plaintext
float y = sqrt(9);
```

Import functions with `#include<cmath>`

- `sqrt(x)` is a function that returns $\sqrt{x}$
- `abs(x)` is a function that returns $|x|$
- `ceil(x)` is a function that returns $\lceil x \rceil$
- `floor(x)` is a function that returns $\lfloor x \rfloor$
- `pow(x, y)` is a function that returns $x^y$

### More pre-defined functions: Random numbers

Import functions with `#include<cstdlib>`

```plaintext
rand() function returns a “random” number between 0 and RAND_MAX-1
(RAND_MAX==2,147,483,647 on storm)
```

Pseudo-random based on number-of-calls, e.g.:

```plaintext
return 2042 for call 1
return 43 for call 2
return 3205394 for call 3
```
Time-based “random” numbers

At start of program, call
`srand(time(0));`
To set the random-number “seed” to the
number of seconds elapsed since 1/1/1970

Smaller random numbers

- Use % and + to scale to desired number range

- Simulate rolling of die:
  ```c
  int roll = (rand() % 6) + 1;
  ```

- Simulate picking 1 of 26 students in our class:
  ```c
  int studentNum = ???
  ```