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

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

Code for timeGreetings.cpp

Get name and time

Output time-based greeting
• Outputs sub-divided into time-based blocks
```cpp
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

```cpp
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

```c
int 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

```c
int area_r2=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

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

int circleArea(int radius); // declaration

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

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

Function declaration

Establish:

- function name
- output type
- input types and names

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

```c
int circleArea(int radius);
// computes factorial of input
```

Function definition

Provides the statements performed when function is used

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

int circleArea(int radius){
    int 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

    int 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

    int area_R2 = circleArea(2);

• return specifies what value to give the caller

Alternate function declaration

`return_type fcn_name(input_list);`

    int circleArea(int radius);

Only argument types required in declaration
But argument names highly recommended

Call-declaration consistency

• Compiler forces match between call and declaration

    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

    int sum_range(int min, int max);
    a = sum_range(10, 3);

    Will not re-order input: min=10, max=3

Pre-defined functions

    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 ⌈x⌉
• `floor(x)` is a function that returns ⌊x⌋
• `pow(x, y)` is a function that returns $x^y$

More pre-defined functions:
Random numbers

    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.:
    return 2042 for call 1
    return 43 for call 2
    return 3205394 for call 3

    Import functions with
    #include<cstdlib>`
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:
  ```
  int roll = (rand() % 6) + 1;
  ```
- Simulate picking 1 of 26 students in our class:
  ```
  int studentNum = ???
  ```