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

Output time-based greeting
- Outputs sub-divided into time-based blocks

Write once, use repeatedly

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

Can print:
  1 mississippi

Can print:
  1 mississippi
  2 mississippi
  3 mississippi

Define equation once, use repeatedly

Factorial: n! = n x (n-1) x ... x 2 x 1
int product=1;
for ( int i=1; i<=5; i++)
{
    product = product*i;
}
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 a=factorial(5);
```

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 factorial(int number); // declaration

int main () {
  . . .
  int a=factorial(4); // usage
  . . .
}

int factorial(int number) { // definition
  int product=1;
  for ( int i=1; i<=5; i++)
    product = product*i;
  return product;
}
```

Functions in your C++ file

```c++
Functions in your C++ file

#include<iostream>
using namespace std;

int factorial(int number); // declaration

int factorial(int number) {
  int product=1;
  for ( int i=1; i<=5; i++)
    product = product*i;
  return product;
}
```

Function declaration

Establish:
- function name
- output type
- input types and names

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

```c++
// sum numbers from min to max
int sum_range(int min, int max);
```

Function definition

Provides the statements performed when function is used

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

```c++
int sum_range(int min, int max){
  int sum=0;
  for (int i=min; i<=max; i++)
    sum+=i;
  return sum;
}
```

Function use – “function call”

- Names function to use
- Provides input arguments for the function
- (If appropriate) can assign output

```
int a = factorial(6);
```

- Call types must be consistent with declaration and definition
The return statement

- When function is “called”, information may be expected back
  
  ```
  int a = factorial(3);
  ```

- `return` specifies what value to give the caller

Alternate function declaration

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

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);
  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
  ```

Implicit type conversions

- Optimally, variable types should be consistent in computations and value assignments:
  
  ```
  int a=2, b=3, c, d;
  c=a+b;
  ```

- When variables are inconsistent, **type-casting** is often performed automatically (in some systems, an error may occur)
  
  ```
  d=c-1.3; // result becomes int
  ```

Explicit type conversions

When variables are inconsistent, can explicitly type-cast with `static_cast<type>(..)`

```
int a=2, b=3, c, d;

c=a+b;
d=c-static_cast<int>(1.3);
```
**More pre-defined functions:**

**Random numbers**

- **Import functions with**
  ```c
  #include<cstdlib>
  ```
- 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
  ```

**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 = ???
  ```

**Review/clarification of int division**

- If a and b are ints
  ```
  a/b computes the integer number of times b goes into a
  a%b computes the remainder after dividing a by b
  ```
- 6/4 == 1
- 6%4 == 2