6. Functions

8. Using Functions
9. Writing Functions
10. Managing Functions and Data

Introduction to Using Functions (1)

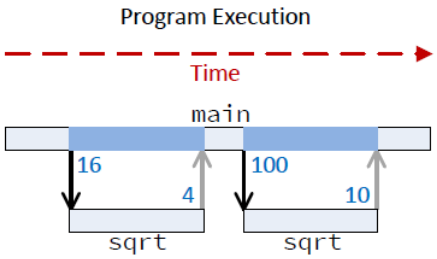
```
#include <iostream>
#include <cmath>
int main() {
   double input;
   std::cout << "Enter number: ";
   std::cin >> input;
   double root = sqrt(input);
   std::cout << "Square root of " << input << " = " << root << '\n';
}
```

```
int main() {
    double value;

    //  Assign variable
    value = 16;

    //  Compute s square root
    double root = sqrt(value);

    //  Compute another
    root = sqrt(100);
}
```



Introduction to Using Functions (2)

```
std::cout << sqrt(16.0) << '\n';

std::cout << sqrt(x) << '\n';

std::cout << sqrt(2 * x - 5) << '\n';

double y = sqrt(x);

y = 2 * sqrt(x + 16) - 4;

y = sqrt(sqrt(256.0));
```


Introduction to Using Functions (3)

```
// Function name, parameter list, return type
// double sqrt(double)
// double sqrt(double x)

// Prototype declaration
double sqrt(double);
double sqrt(double x);

std::cout << sqrt("16") << '\n';
std::cout << sqrt(4.0, 7.0) << '\n';


#include <algorithm>

std::cout << "The larger of " << 4 << " and " << 7
   << " is " << std::max(4, 7) << '\n';
```

Introduction to Using Functions (4)

```
int rand(); // generating pseudorandom number
void exit(int); // terminating the program's execution

std::cout << exit(8) << '\n';
```


Standard Math Functions (1)

```
// <cmath>
double sqrt(double x);
double exp(double x); // Natural Exponential
double log(double x); // Natural logarithm
double log10(double x);
double sin(double x); // x: radian
double cos(double x);
double tan(double x);
double pow(double x, double y);
double fabs(double x);


double asin(double x);
double acos(double x);
double atan(double x);
double atan2(double y, double x);
```

Standard Math Functions (2)

```
double sqrt(double);
float sqrt(float);
long double sqrt(long double);

// overloading or template
```


Maximum and Minimum

```
#include <iostream>
#include <algorithm>
int main() {
   int value1, value2;
   std::cout << "Please enter two integer values: ";
   std::cin >> value1 >> value2;
   std::cout << "max = " << std::max(value1, value2)
      << ", min = " << std::min(value1, value2) << '\n';
}
```

Character Functions

```
#include <iostream>
#include <cctype>
int main() {
   for (char lower = 'a'; lower <= 'z'; lower++) {
      char upper = toupper(lower);
      std::cout << lower << " => " << upper << '\n';
   }
}

// int toupper(int ch);
// int tolower(int ch);
// int isupper(int ch);
// int islower(int ch);
// int isalpha(int ch);
// int isdigit(int ch);
```


Random Numbers

```
// void srand(unsigned)
// int rand()

#include <iostream>
#include <cstdlib>
int main() {
   srand(23);
   for (int i = 0; i < 100; i++) {
      int r = rand();
      std::cout << r << " ";
   }
   std::cout << '\n';
}


// srand(static_cast<unsigned>(time(0)));
// #define RAND_MAX 0x7fff
// int r = rand() % 100;
```

Function Basics (1)

```
#include <iostream>
double Abs(double x) {
    return (x>0)?x:-x;
}
int main() {
    std::cout << Abs(10.5) << '\n';
    std::cout << Abs(-10.5) << '\n';
}
```

Function definition
- Function name
- Return type
- Parameter list
- Body

Function invocation (by call)

Custom Functions vs. Standard Functions

Function Basics (2)

```
#include <iostream>
// Definition of the prompt function
void prompt() {
   std::cout << "Please enter an integer value: ";
   // return;
}

int main() {
   int value1, value2, sum;
   std::cout << "This program adds together two integers.\n";
   prompt(); // Call the function
   std::cin >> value1;
   prompt(); // Call the function again
   std::cin >> value2;
   sum = value1 + value2;
   std::cout << value1 << " + " << value2 << " = " << sum << '\n';
}
```

Function Basics (3)

```
#include <iostream>
// Definition of the prompt function
int prompt() {
   int result;
   std::cout << "Please enter an integer value: ";
   std::cin >> result;
   return result;
}
int main() {
   int value1, value2, sum;
   std::cout << "This program adds together two integers.\n";
   value1 = prompt();
   value2 = prompt();
   sum = value1 + value2;
   std::cout << value1 << " + " << value2 << " = " << sum << '\n';
}
```


Function Basics (4)

```
#include <iostream>
int prompt(int n) {
   int result;
   std::cout << "Please enter integer #" << n << ": ";
   std::cin >> result;
   return result;
}
int main() {
   int value1, value2, sum;
   std::cout << "This program adds together two integers.\n";
   value1 = prompt(1); // Call the function
   value2 = prompt(2); // Call the function again
   sum = value1 + value2;
   std::cout << value1 << " + " << value2 << " = " << sum << '\n';
}
```

Using Functions

```
int gcd(int num1, int num2) // int gcd(int num1, num2)
{
   int min = (num1 < num2) ? num1 : num2;
   int largestFactor = 1;

   for (int i = 2; i <= min; i++)
      if (num1 % i == 0 && num2 % i == 0)
         largestFactor = i; // Found larger factor

   return largestFactor;
}
// std::cout << gcd(36, 24);
// x = gcd(x - 2, 24);
// x = gcd(x - 2, gcd(10, 8));

// Greatest common divisor
// Euclidean algorithm
```


Pass by Value (1)

```
#include <iostream>
void increment(int x) {
   std::cout << "Beginning execution of increment, x = "
      << x << '\n';
   x++; // Increment x
   std::cout << "Ending execution of increment, x = "
      << x << '\n';
}

int main() {
   int x = 5;
   std::cout << "Before increment, x = " << x << '\n';
   increment(x);
   std::cout << "After increment, x = " << x << '\n';
}
```

Pass by Value (2)

```
// Local variables
#include <iostream>
void increment() {
   int x = 15;
   std::cout << "x = " << x << '\n';
   x++;
   std::cout << "x = " << x << '\n';
}
int main() {
   int x = 5;
   std::cout << "x = " << x << '\n';
   increment();
   std::cout << "x = " << x << '\n';
   {
      int x = 10;
      std::cout << "x = " << x << '\n';
   }
   std::cout << "x = " << x << '\n';
}
```


Function Example

```
#include <iostream>
#include <cmath>
bool is_prime(int n) {
   bool result = true;
   double r = n, root = sqrt(r);

   for (int trial_factor = 2;
      result && trial_factor <= root; trial_factor++)
      result = (n % trial_factor != 0);

   return result;
}
```

Organizing Functions

```
#include <iostream>
int main() {
   std::cout << twice(5) << '\n';
}
int twice(int n) {     // Define function twice
   return 2 * n;
}

-------------------------------------------------------------

#include <iostream>
int twice(int);        // Declare function named twice
int main() {
   std::cout << twice(5) << '\n';
}
int twice(int n) {     // Define function named twice
   return 2 * n;
}
```


Commenting Functions

```
/*
*       distance(x1, y1, x2, y2)
*       Computes the distance between two geometric points
*       x1 is the x coordinate of the first point
*       y1 is the y coordinate of the first point
*       x2 is the x coordinate of the second point
*       y2 is the y coordinate of the second point
*       Returns the distance between (x1,y1) and (x2,y2)
*       Author: Joe Algori (joe@eng-sys.net)
*       Last modified: 2010-01-06
*       Adapted from a formula published at
*       http://en.wikipedia.org/wiki/Distance
*/
double distance(double x1, double y1, double x2, double y2) {
   …
}
```

Global Variables (1)

```
#include <iostream>

double result = 0.0, arg1, arg2;
void get_input() {
   std::cin >> arg1 >> arg2;
}
void report() {
   std::cout << result << '\n';
}
void add()      {       result = arg1 + arg2;          }
void subtract(){       result = arg1 - arg2;          }

int main() {
   get_input();
   add();
   report();
}
```


Global Variables (2)

```
#include <iostream>
int x; // = 0?

int main() {
   int x = 10;
   std::cout << x << std::endl;
   std::cout << ::x << std::endl;     // scope resolution operator
}



extern int i; // declaration of a variable named i of type int,
              // defined somewhere in the program.
```

Static Variables

```
#include <iostream>
#include <iomanip>
int count() {
   static int cnt = 0;  // static int cnt;
   return ++cnt;        // Increment and return current count
}
int main() {
   // Count to ten
   for (int i = 0; i < 10; i++)
      std::cout << count() << ' ';

   std::cout << '\n';
}
```

A global static variable/function is one that can only be accessed in the file where it is created. This variable is said to have file scope.

Overloaded Functions

```
// Function overloading is a C++ programming feature that allows us
// to have more than one function having same name but different
// parameter list

void f() { /* ... */ }
void f(int x) { /* ... */ }
void f(double x) { /* ... */ }
void f(int x, double y) { /* ... */ }
void f(double x, int y) { /* ... */ }
```

Default Arguments (1)

```
#include <iostream>
void countdown(int n=10) {
  while (n >= 0) // Count down from n to zero
  std::cout << n-- << '\n';
}

int main() {
  countdown(5);
  std::cout << "----------" << '\n';
  countdown();
}

-------------------------------------------------------------------
int sum_range(int n=0, int m=100);
int sum_range(int n, int m=100);
int sum_range(int n=0, int m);
```


Default Arguments (2)

```
#include <iostream>
void countdown(int n=10);
int main() {
  countdown(5);
  std::cout << "----------" << '\n';
  countdown();
}
void countdown(int n) {
  while (n >= 0) // Count down from n to zero
  std::cout << n-- << '\n';
}
```

Default Arguments (3)

```
// Mixing overloading and default arguments can produce ambiguities
// that the compiler will not allow.

void f() { /* .... */ }
void f(int n=0) { /* .... */ }

-------------------------------------------------------------------

void f(int m) { /* .... */ }
void f(int n=0) { /* .... */ }
```


Recursion (1)

```
// Recursion: function calls itself directly or indirectly


#include <iostream>
void F(int n) {
   std::cout << n%10 << std::endl;
   if(n/10 > 0) F(n/10);
}
int main() {
   F(123);
}
```

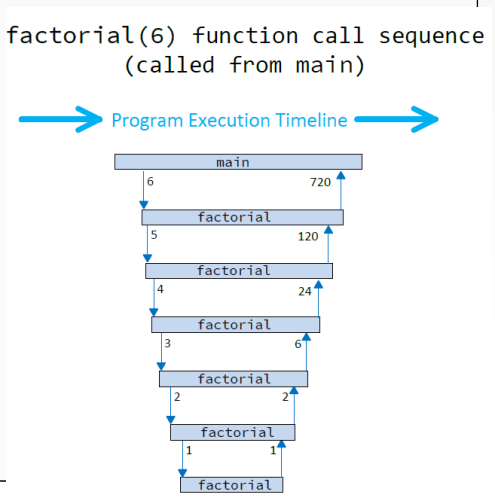
Recursion (2)

```
// Factorial

int factorial(int n) {
   if (n == 0)
      return 1;
   else
      return n * factorial(n - 1);
}
```


Recursion (3)

```
factorial(6)    = 6 * factorial(5)
                = 6 * 5 * factorial(4)
                = 6 * 5 * 4 * factorial(3)
                = 6 * 5 * 4 * 3 * factorial(2)
                = 6 * 5 * 4 * 3 * 2 * factorial(1)
                = 6 * 5 * 4 * 3 * 2 * 1 * factorial(0)
                = 6 * 5 * 4 * 3 * 2 * 1 * 1
                = 6 * 5 * 4 * 3 * 2 * 1
                = 6 * 5 * 4 * 3 * 2
                = 6 * 5 * 4 * 6
                = 6 * 5 * 24
                = 6 * 120
                = 720
```



Recursion (4)

```
#include <iostream>
int gcd(int m, int n) {
  if (n == 0)
    return m;
  else
    return gcd(n, m % n);
}

-----------------------------------------------------------------------

int fibonacci(int n) {
   if (n <= 0)
      return 0;
   else if (n == 1)
      return 1;
   else
      return fibonacci(n - 2) + fibonacci(n - 1);
}
```


Making Functions Reusable

```
// prime.cpp
#include <cmath>
bool is_prime(int n) {
   bool result = true;
   double r = n, root = sqrt(r);
   for (int trial_factor = 2; result && trial_factor <= root;
      trial_factor++)
      result = (n % trial_factor != 0);
   return result;
}

// prime.h
bool is_prime(int);

// test.cpp
#include "prime.h"
int main() { /*    */ }
```

Pointers (1)

```
// &: address operator
#include <iostream>
int main() {
   int x = 10;

   std::cout << &x << std::endl;
}

// A pointer is a variable that holds a memory address
// where a value lives.
int *p1;
double *p2;
char *p3;
```

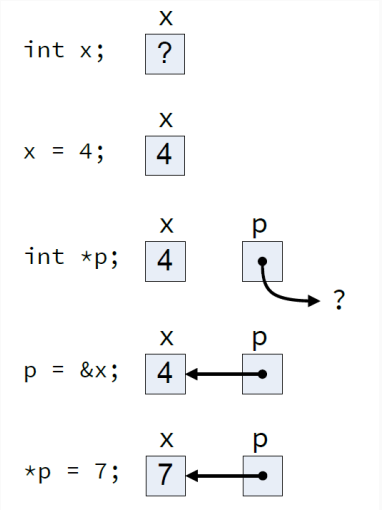

Pointers (2)

```
#include <iostream>
int main() {
   int x;
   x = 4;
   int *p;
   p = &x;

   std::cout << &x << std::endl;
   std::cout << p << std::endl;

   std::cout << x << std::endl;
   std::cout << *p << std::endl;

   *p = 7; // dereferencing/indirect operator
   std::cout << x << std::endl;
   std::cout << *p << std::endl;
}
```



Pointers (3)

```
// nullptr

// reinterpret_cast: conversion between unrelated types
//   int → ptr, ptr → int, ptr → ptr
//
// int *p = reinterpret_cast<int *>(5);
```


Reference Variable

```
int x;
int& r = x; // r is a reference variable. r aliases x, not address
-------------------------------------------------------
#include <iostream>
int main() {
   int x = 5;  int y = x;     int& r = x;
   std::cout << "x = " << x << '\n';
   std::cout << "y = " << y << '\n';
   std::cout << "r = " << r << '\n';
   x = 7;
   std::cout << "x = " << x << '\n';
   std::cout << "y = " << y << '\n';
   std::cout << "r = " << r << '\n';
   y = 8;
   std::cout << "x = " << x << '\n';
   std::cout << "y = " << y << '\n';
   std::cout << "r = " << r << '\n';
}
```

Pass by Reference (1)

```
#include <iostream>
void swap(int a, int b) {
   int temp = a;
   a = b;
   b = temp;
}

int main() {
   int var1 = 5, var2 = 19;
   std::cout << "var1 = " << var1 << ", var2 = " << var2 << '\n';
   swap(var1, var2);
   std::cout << "var1 = " << var1 << ", var2 = " << var2 << '\n';
}
```


Pass by Reference (2)

```
#include <iostream>
void swap(int& a, int& b) {
   int temp = a;
   a = b;
   b = temp;
}

int main() {
   int var1 = 5, var2 = 19;
   std::cout << "var1 = " << var1 << ", var2 = " << var2 << '\n';
   swap(var1, var2);
   std::cout << "var1 = " << var1 << ", var2 = " << var2 << '\n';
   // swap(5, 19);
}
```

Pass by Reference (3)

```
#include <iostream>
void swap(int *a, int *b) {    // Pass by reference via pointers
   int temp = *a;              // Pass by address
   *a = *b;
   *b = temp;
}

int main() {
   int var1 = 5, var2 = 19;
   std::cout << "var1 = " << var1 << ", var2 = " << var2 << '\n';
   swap(&var1, &var2);
   std::cout << "var1 = " << var1 << ", var2 = " << var2 << '\n';
}
```


Higher-order Functions (1)

```
// parameter or return

#include <iostream>
int add(int x, int y) {
   return x + y;
}
int multiply(int x, int y) {
   return x * y;
}
int evaluate(int (*f)(int, int), int x, int y) { // function pointer
   return f(x, y);
}
int main() {
   std::cout << add(2, 3) << '\n';
   std::cout << evaluate(&add, 2, 3) << '\n';
   std::cout << evaluate(&multiply, 2, 3) << '\n';
}
```

Higher-order Functions (2)

```
#include <iostream>
int add(int x, int y) {
   return x + y;
}

int main() {
   int (*func)(int, int);
   func = add;
   std::cout << func(7, 2) << '\n';
}
```