Functions

Kornchawal Chaipah

Computer Engineering Department
Khon Kaen University

Agenda

- Functions
  - Pre-defined functions
  - User-defined functions
- Locality and Scope
- Return values
- Passing by value vs. by reference

Functions

- Function = modularized sub-program
- Divide a large program into small pieces
  - To make a large program manageable
- Can be compiled and tested separately
- Can be reused in different programs

Function Types

- Types by definitions
  - Pre-defined function
  - User-defined function
- Types by returned value
  - Return a value: different data types
  - Do not return a value: void

Standard C++ Library

- A collection of pre-defined functions
- Accessed through header files
- Examples
  - sqrt() is in <cmath>
  - rand() is in <cstdlib>
- http://www.cplusplus.com/reference/

Standard C++ Library Examples

```
#include <library>
```

For this class:	<cmath>	mathematical functions
	<cstring>	string functions
	<cstdlib>	utility functions
	<ctime>	date/time functions
	<climits>	the integer limit
Extra:	<cassert>	the assert()function
	<cctype>	functions to test characters
	<cfloat>	constants relevant to float
	<cstdio>	functions for standard I/O

Function Examples

- Math functions
  - sqrt()
  - sin()
  - cos()
- Random number functions
  - rand()
  - srand()
  - time()

Some Functions in <cmath>

`sin(x)`	sine
`cos(x)`	cosine
`tan(x)`	tangent
`asin(x)`	inverse sine
`acos(x)`	inverse cosine
`atan(x)`	inverse tangent
`ceil(x)`	ceiling
`floor(x)`	floor
`log(x)`	natural logarithm
`log10(x)`	base-10 logarithm
`pow(x, p)`	x to the power p

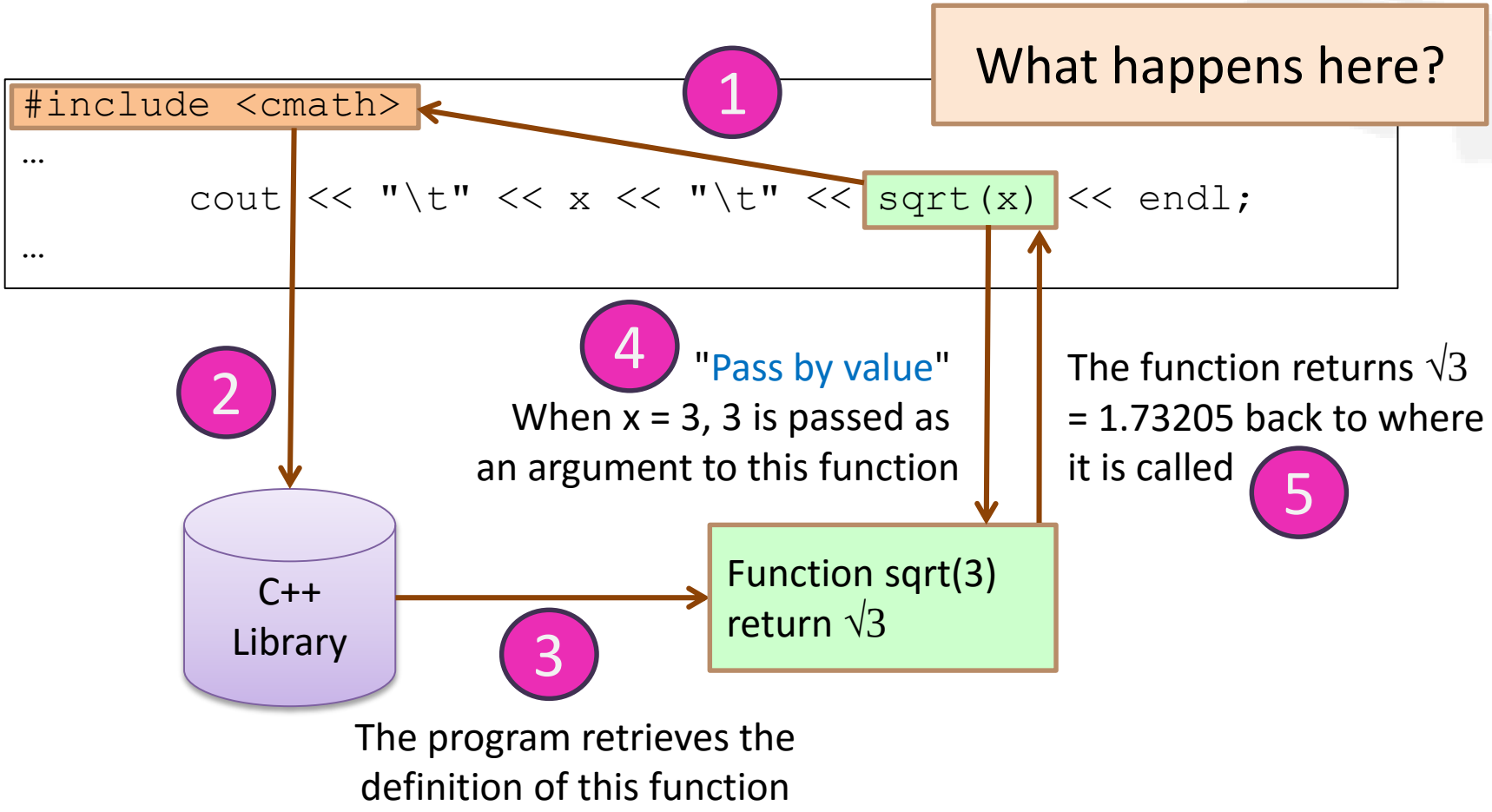
The Square Root Function sqrt()

```
#include <cmath>

int main(){
   for(int x = 0; x < 6; x++){
        cout << "\t" << x << "\t" << sqrt(x) << endl;
   }
   return 0;
}
```

What happens here?

- sqrt( ) is a function
- sqrt is a name of the function
- x is an argument or actual parameter of the function call
  - an argument can be a variable (e.g. x) or a value (e.g. 3)

The Square Root Function sqrt()



Another <cmath> Example: Trigonometry

Code:

```
#include <cmath>
int main(){
   for(float x = 0; x < 5; x += 2){
      cout << x << "\t\t"
           << sin(2*x) << "\t"
           << 2 * sin(x) * cos(x)
           << endl;
   }
   return 0;
}
```

Output:

Nested Function Calls

Code:

```
#include <cmath>
int main(){
   for(float x = 0; x < 10; x++){
        float y = sqrt(1 + 2 * sqrt(3 + 4* sqrt(x)));
        cout << y << endl;
   }
   return 0;
}
```

Do the most inner one first

Generating Pseudo-Random Numbers

Code:

```
#include <cstdlib>

int main(){
   for(int i = 0; i<8; i++){
        cout << rand() << endl;
   }
   cout << "RAND_MAX = " << RAND_MAX
        << endl;
   return 0;
}
```

Must include a header when using a pre-defined function (rand()) and other element (RAND_MAX)

"pseudo"

- The series of returned values of rand() is pre-determined.
- It will give the same series of numbers when called again.
- We need different seeds to generate different series of pseudo-random numbers

rand() returns a pseudo-random number each time it is called

Setting the Seed

A user defines a seed

```
#include <cstdlib>

int main(){
   unsigned seed;
   cout << "Enter seed: ";
   cin >> seed;
   srand(seed);
   for(int i = 0; i<8; i++){
        cout << rand() << endl;
   }
   return 0;
}
```

Use current time (system clock) as a seed

```
#include <cstdlib>
#include <ctime>

int main(){
   unsigned seed = time(NULL);
   cout << "seed = " << seed
        << endl;
   srand(seed);
   for(int i = 0; i<8; i++){
        cout << rand() << endl;
   }
   return 0;
}
```

Pseudo-Random Numbers in Range

Code:

```
#include <cstdlib>
#include <ctime>
int main(){
   unsigned seed = time(NULL);
   cout << "seed = " << seed << endl;
   srand(seed);
   int min, max;
   cout << "Enter min and max: ";
   cin >> min >> max;
   int range = max - min + 1;
   for(int i = 0; i < 20; i++){
      int r = rand() / 100 % range + min;
      cout << r << endl;
   }
   cout << endl;
   return 0;
}
```

Output:

Why this formula?

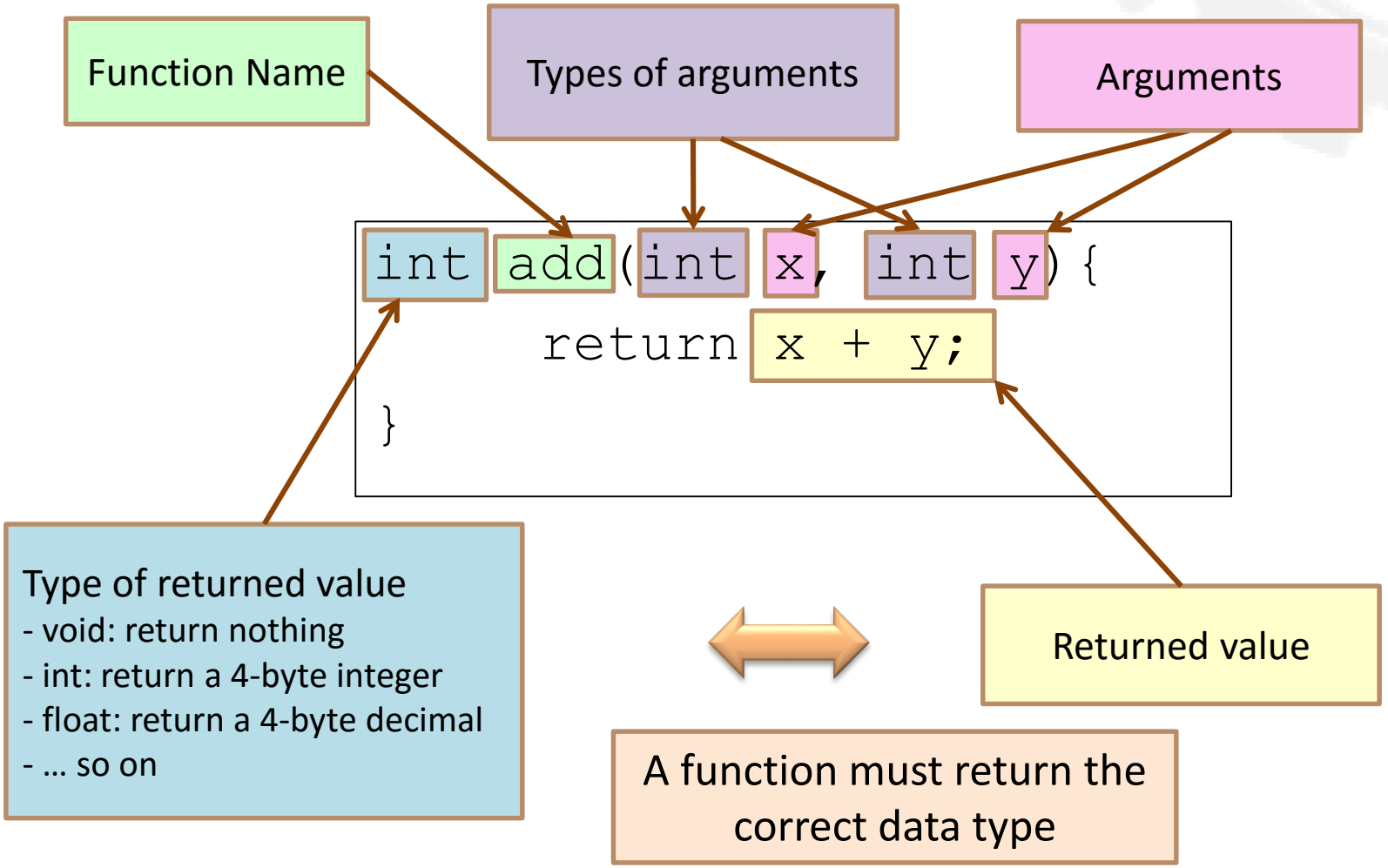
User-Defined Functions

- The standard C++ library is still not sufficient for most programming tasks
  - Programmers need to define their own functions
- Example: summation of 2 integers: x + y

Code:

```
int add(int x, int y){
        return x + y;
}
```

Function Components



Defining and Calling a Function #1

Code:

```
int add(int x, int y){
        return x + y;
}

int main(){
   int a = 1, b = 1;
   while(a != 0 && b != 0){
        cin >> a >> b;
        cout << "\t" << a << " + " << b << " = "
             << add(a,b) << endl;
   }
   return 0;
}
```

1

Function Definition: must define a function before calling it

2

When calling a function, the program will look for the definition defined earlier

Your First User-Defined Function

- Problem: write a function to find max between a given 2 integers
- Need to know
  - What are arguments and returned value?
  - What are their types?

My max( )

Code:

```
[          ] max([                ]) {
    [                                    ]
}

int main(){
   int a, b;
   do{
        cin >> a >> b;
        cout << "\tmax(" << a << ","
             << b << ") = " << max(a,b)
             << endl;
   }while(a != 0);
   return 0;
}
```

Output:

My max( )

Code:

```
int max( int x, int y ){

    if(x > y) return x;
    return y;
}

int main(){
    int a, b;
    do{
        cin >> a >> b;
        cout << "\tmax(" << a << ","
             << b << ") = " << max(a,b)
             << endl;
    }while(a != 0);
    return 0;
}
```

Output:

Function Mess!

- What if a function is long or there are many functions?
- The main() will be kicked down to the bottom of the file
  - Not quite good programming practice
  - Prefer a top-down fashion
    - We want to see the main() function FIRST

Defining and Calling a Function #2

Code:

```
int max(int x, int y);

int main(){
   int a, b;
   do{
        cin >> a >> b;
        cout << "\tmax(" << a << ","
             << b << ") = " << max(a,b)
             << endl;
   }while(a != 0);
   return 0;
}
int max(int x, int y){
   if(x < y) return y;
   return x;
}
```

1

Declare a function before calling it

3

When calling a function, the program will look for the definition declared earlier

2

Define a function after the main( ) function instead

Area of a Circle

- Problem: get a radius r, then calculate an area of a circle with radius r
- What do you know?
  - Area = πr^2
- Input: r
- Output: area of a circle

Code: Area of a Circle v1

```
float fiArea(float r) {
    const float PI = 3.14159;
    float area;
    area = PI * r * r;
    return area;
}

int main() {
    float r;
    cout << "Enter radius: ";
    cin >> r;

    cout << "Area of a circle is " << fiArea(r) << endl;
    return 0;
}
```

Code: Area of a Circle v2

```
float fiArea(float);

int main() {
    float r;
    cout << "Enter radius: ";
    cin >> r;

    cout << "Area of a circle is " << fiArea(r) << endl;
    return 0;
}

float fiArea(float r) {
    const float PI = 3.14159;
    float area;
    area = PI * r * r;
    return area;
}
```

The main( ) function

```
int main() {
    ...
   return 0;
}
```

- Return type = int
  - Therefore, return 0; at the end of the function
- Some compilers allow return 0; to be omitted
  - Return value of 0 is default
  - Some compilers will complain when this is omitted
  - But please include it as a good practice

Terminate a Program

```
int main() {
   int n, d;
   cout << "Enter two integers: ";
   cin >> n >> d;
   if (d == 0) {
        return 0;
   }
   cout << n << "/" << d << " = " << n/d << endl;
    return 0;
}
```

- Four ways to terminate a program (in main() function)
  - return 0;
  - exit(0);
  - abort();
  - Throwing an uncaught exception

void Function

- Returns no value
  - Empty set of value
- Also called a procedure or a subroutine
- Used when we do not return any value to the caller
  - Such as a printing routine

A Function that Prints Menu

```
void printMenu(){
        cout << "Today's menu!" << endl;
        cout << "1. Coca Cola" << endl;
        cout << "2. Sprite" << endl;
        cout << "3. Fanta" << endl;
        cout << "4. Expresso shot" << endl;
        cout << "5. Latte" << endl;
        cout << "6. Cappuccino" << endl;
        cout << "7. Hot chocolate" << endl;
        cout << "8. Simply milk" << endl;
}

int main(){
        printMenu();
        return 0;
}
```

- Just print the message and done
- No return value

Boolean Function

- Returns true or false
- Helps with evaluating values for conditions within if and while statements
  - `if( func(x) ) statement;`
  - `while( func(x) ) statement;`
- Named after George Boole who developed Boolean algebra

isEven( ) Function

```
bool isEven(int x){
	if(x%2 == 0) {
		return true;
	}
	return false;
}
int main() {
	int x;
	cout << "Enter integer x: ";
	cin >> x;
	if (isEven(x)){
		cout << x << " is even." << endl;
	}
	else{
		cout << x << " is odd." << endl;
	}
	return 0;
}
```

Leap Year

- Problem: write a function to test whether an input is a leap year
- Input: a year in A.D.
- Output: whether a year is a leap year
- What we know about a leap year
  - The year is divisible by 4 but not 100 or
  - The year is divisible by 400

A Leap Year Function

```
bool isLeapYear(int y){
   return y%4 == 0 && y%100 != 0 || y%400 == 0;
}

int main(){
   int n = 1;
    while(n >= 1){
        cin >> n;
        if(isLeapYear(n)){
                cout << n << " is a leap year.\n";
        }
        else{
                cout << n << " is not a leap year.\n";
        }
   }
   return 0;
}
```

I/O Functions

- Use functions to encapsulate messy tasks such as taking inputs
  - Reading inputs
  - Checking the validity of inputs
    - E.g. make sure a user's age is correct
- Sending inputs back by
  - Returning an input
  - Passing one or more inputs by reference
    - More about passing by reference a little later

Reading an age

```
int getAge();

int main(){
   int age = getInput();
   cout << "Your age is " << age << endl;
   return 0;
}

int getAge(){
   int n;
   while(true){
        cout << "How old are you: ";
        cin >> n;
        if(n < 0) cout << "\a\tnegative.";
        else if(n > 120) cout << "\a\tover 120.";
        else return n;
        cout << "\n\tTry again.\n";
   }
}
```

Getting an input

Validating an input

Variable Scope

- A scope of a name
  - Begins where the name is declared
  - Ends at the end of the innermost block in which it is declared
- A program can have the same variable name *if* their scopes are nested or disjoint
  - But is it a good practice? Probably not.

Local Variables

- Local variables = declared inside a block
  - Accessible only from within that block
  - Parameters (arguments) are also local variables

```
int main(){
    int a, b;
    cin >> a >> b;
    cout << a << b << endl;
    if(a > b)
    {
        int m = 2;
    }
    return 0;
}
```

A function is a block:
a, b are local variables +
can be used in this function +
Exist only while this function is executing

Local Variables

- Local variables = declared inside a block
  - Accessible only from within that block
  - Parameters (arguments) are also local variables

```
int main(){
    int a, b;
    cin >> a >> b;
    cout << a << b << endl;
    if(a > b)
    {
        int m = 2;
    }
    return 0;
}
```

"if block" is also a block:
m is local to the "if block" + can be used only in this block

change( ) Function

```
void change(int x) {
        x = x + 100;
        cout  <<  "Value of x in change() is: " << x << '\n';
}

int main( ){
        int x;
        x = 100;
        cout  <<  "Value of x in  before change() is: "
        << x << '\n';

        change(x);

        cout  <<  "Value of x in  after change() is: "
        << x << '\n';
        return 0;
}
```

This x is a local variable in block of change( ) function

This x is a local variable in block of main( ) function

Not the same x!!!

x is not changed!

-- more in the passing by value topic

Nested and Parallel Scopes 1/2

```
void f();
void g();
int x = 11;
```

Block 1: Program Global Block

Global Variable x: can be used in any function

```
int main() {
   int x = 22;
   {    int x = 33;
        cout << "In block inside main(): x = " << x << endl;
   }
   cout << "In main(): x = " << x << endl;
   cout << "In main(): ::x = " << ::x << endl;
   f();
   g();
}
...
```

Block 2: main ( ) Function Block

Block 3: Inner Block

How to access the value of a global variable x

Nested and Parallel Scopes 2/2

```
...

void f() {
   int x = 44;
   cout << "In f(): x = " << x << endl;
}

void g() {
   cout << "In g(): x = " << x << endl;
}
```

Block 4: f( ) function Block

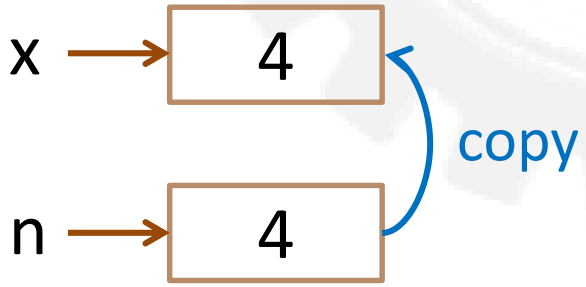
Block 5: g( ) function Block

Passing Parameters to a Function

- Passing by Value
  - Passing the copy of value of a variable
  - Variable value changed in the function, NOT changed in the calling function
- Passing by Reference
  - Passing the address of a variable
  - Variable value changed in the function, also changed in the calling function

Passing by Value

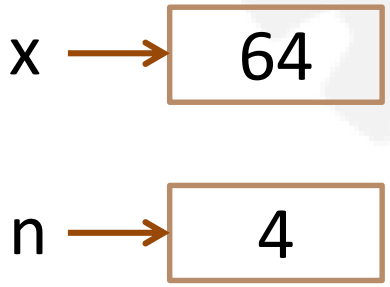
```
int cube(int x){
   x = x * x * x;
   return x;
}
int main(){
   int n = 4;
   cout << "\tcube(" << n << ") = " << cube(n) << endl;
   cout << "\t current n = " << n << endl;
   return 0;
}
```



- When calling cube(n) in main(), 4 is passed to the local variable x in cube().
- 4 is used only inside the function, n is unaffected by the function cube()
  - n is a read-only parameter

Passing by Value

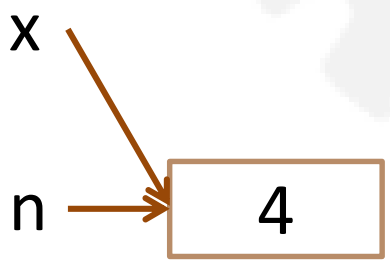
```
int cube(int x){
   x = x * x * x;
   return x;
}
int main(){
   int n = 4;
   cout << "\tcube(" << n << ") = " << cube(n) << endl;
   cout << "\t current n = " << n << endl;
   return 0;
}
```



- When calling cube(n) in main(), 4 is passed to the local variable x in cube().
- 4 is used only inside the function, n is unaffected by the function cube()
  - n is a read-only parameter

Passing by Reference

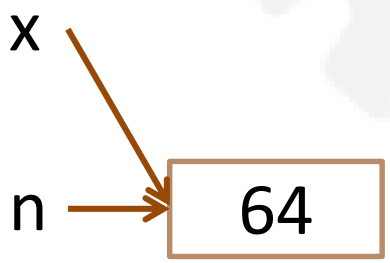
```
int cube2(int &x){
   x = x * x * x;
   return x;
}
int main(){
   int n = 4;
   cout << "\tcube(" << n << ") = " << cube2(n) << endl;
   cout << "\t current n = " << n << endl;
   return 0;
}
```



- Passing by value: n is NOT changed after calling cube(n)
- Passing by reference: n is CHANGED after calling cube2(n)

Passing by Reference

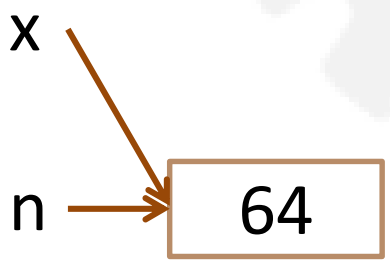
```
int cube2(int &x){
   x = x * x * x;
   return x;
}
int main(){
   int n = 4;
   cout << "\tcube(" << n << ") = " << cube(n) << endl;
   cout << "\t current n = " << n << endl;
   return 0;
}
```



- Passing by value: n is NOT changed after calling cube(n)
- Passing by reference: n is CHANGED after calling cube2(n)

Passing by Reference

```
int cube2(int &x){
   x = x * x * x;
   return x;
}
int main(){
   int n = 4;
   cout << "\tcube(" << n << ") = " << cube(n) << endl;
   cout << "\t current n = " << n << endl;
   return 0;
}
```



- Sometimes a function needs to change the value of parameter(s) passed to it.
  - So we can pass it by reference
- Marked by putting & in front of the parameter in the function definition

swap( )

```
void swap(float&, float&);

int main(){
   float a = 22.2, b = 44.4;
   cout << "a = " << a << " b = " << b << endl;
   swap(a,b);
   cout << "a = " << a << " b = " << b << endl;
   return 0;
}

void swap(float &x, float &y){
   float temp = x;
   x = y;
   y = temp;
}
```

Passing by Reference

2

Also changed here in the calling function

1

Values changed here in the function

Passing by Value and Passing by Reference

```
void f(int, int&);
int main(){
   int a = 22, b = 44;
   cout << "a = " << a << " b = " << b << endl;
   f(a, b);
   cout << "a = " << a << " b = " << b << endl;
   f(2*a-3, b);
   cout << "a = " << a << " b = " << b << endl;
   return 0;
}
void f(int x, int &y){
   x = 88;
   y = 99;
}
```

Passed by reference

Passed by value

Passing Parameter Comparison

Passing by Value

- int x;
- x is a local variable
- A copy of an argument
- Cannot be changed in the function (read-only)
- Passed using a constant, a variable or an expression

Passing by Reference

- int &x;
- x is a local variable
- An address of an argument
- Can be changed in the function (read-write)
- Passed using a variable

Returning More than One Value

- Sometimes we need to return more than one value from a function
  - E.g. a function that reads more than one input
- But a function can return only one value
- Use passing parameters by reference!

A Circle Area and Circumference

```
void computeCircle(double &area, double &circum, double r){
   const double PI = 3.141592653589793;
   area = PI * r * r;
   circum = 2 * PI * r;
}

int main(){
   double r,a,c;
   cout << "Enter a radius: ";
   cin >> r;
   computeCircle(a,c,r);
   cout << "area = " << a << " circumference = " << c <<
   endl;
   return 0;
}
```

Passed by reference to return these 2 values to the main( ) function

Passing by Constant Reference

- Reasons for passing a parameter by reference
  - Allows a function to change the value of an argument
  - Allows a program to save storage space of an argument
    - Creating a copy of a value needs an extra space
- Passing by constant reference
  - Works the same way as passing by reference
  - But a function CANNOT change the value of the parameter

Passing by Constant Reference Example

```
void f(int x, int &y, const int &z){
   x += z;
   y += z;
   cout << "x = " << x << " y = " << y << " z = " << z
   << endl;
}
int main(){
   int a = 22, b = 33, c = 44;
   cout << a << " " << b << " " << c << endl;
   f(a, b, c);
   cout << a << " " << b << " " << c << endl;
   f(2*a-3, b, c);
   cout << a << " " << b << " " << c << endl;
   return 0;
}
```

Use passing by constant reference to save memory space, but cannot change the value

Take Home Messages 1/2

- To use a pre-defined function, we need to put #include <library>
- Type of a function must be the same as the type of a returned value
- We can write a function definition before or after main()
  - If after, we must declare the function before main()
- We must pass parameters in order with correct types

Take Home Messages 2/2

- Local vs Global variable
  - Use local variable only within a block
  - Use global variable anywhere
- Passing a value to a function
  - By value: passing a copy of value
  - By reference: passing an address of a variable

References

- คณาจารย์คณะวิศวกรรมศาสตร์. (n.d.). **เอกสารชุดนำเสนอภาพและบรรยายวิชาการเขียนโปรแกรม (ส่วนกลาง)**
- คณาจารย์คณะวิศวกรรมศาสตร์. (n.d.). **เอกสารประกอบการสอนวิชาการเขียนโปรแกรม (ส่วนกลาง)**
- รศ. วิโรจน์ ทวีปวรเดช. (2554). **การเขียนโปรแกรมคอมพิวเตอร์ Computer Programming.** พิมพ์ครั้งที่ 2 โรงพิมพ์มหาวิทยาลัยขอนแก่น
- Cplusplus.com. (n.d.). **C++ Documentation**. สืบค้นเมื่อ 18 กุมภาพันธ์ 2555, http://www.cplusplus.com/
- Cplusplus.com. (n.d.). **Library Reference**. สืบค้นเมื่อ 18 กุมภาพันธ์ 2555, http://www.cplusplus.com/
- ISO/IEC 14882 Programming Language – C++

EXTRA SLIDES

Permutation

- Problem: write a function to compute permutation

$$p(n,k)=\frac{n!}{(n-k)!}$$

- Also write a factorial function
  - Use it in the permutation function

Factorial function

```
long fact(int);

int main(){
   for(int i = -1; i < 6; i++)
        cout << " " << fact(i);
   cout << endl;
   return 0;
}

long fact(int n){
   if(n < 0) return 0;
   long f = 1;
   while(n > 1){
        f *= n--;
   return f;
}
```

i is local to the for loop

n and f are local to the fact function

Permutation Function

```
long perm(int, int);

int main(){
   for(int i = -1; i < 8; i++)
        for(int j = -1; j <= i+1; j++)
            cout << " " << perm(i,j);
   cout << endl;
   return 0;
}

long perm(int n, int k){
   if(n < 0 || k < 0 || k > n) return 0;
   return fact(n) / fact(n-k);
}
```

- i is local to the outer and inner for loop
- j is local to the inner for loop

n and k are local to the perm function

printDate( )

```
void printDate(int, int, int);

int main(){
   printDate(5, 12, 2012);
   return 0;
}

long printDate(int m, int d, int y){
   if(m < 1 || m > 12 || d < 1 || d > 31 || y < 0){
        cout << "Error: parameter out of range.\n";
        return;
   }
   switch(m){
        case 1: cout << "January "; break;
        case 2: cout << "February "; break;
        ……
        case 12: cout << "December "; break;
   }
   cout << d << ", " << y << endl;
}
```

Check inputs

Print corresponding months

Print date and year

Classifying a Character

```
void printCharCategory(char);
bool isDigit(char);
bool isLower(char);
bool isUpper(char);
bool isSpace(char);
bool isCntrl(char);
bool isPunct(char);

int main(){
   printCharCategory('a');
   printCharCategory('9');
   printCharCategory('-');
   printCharCategory(' ');
   return 0;
}
```

printCharCategory( )

```
void printCharCategory(char c){
   cout << "The character [" << c << "] is ";

   if(isDigit(c)) cout << "a digit.\n";
   else if(isLower(c)) cout << "a lowercase letter.\n";
   else if(isUpper(c)) cout << "a capital letter.\n";
   else if(isSpace(c)) cout << "a white space.\n";
   else if(isCntrl(c)) cout << "a control.\n";
   else if(isPunct(c)) cout << "a punctuation\n";
   else cout << "Error.\n";
}
```

Boolean Functions

```
bool isLower(char c){
   if(c >= 'a' && c <= 'z') return true;
   return false;
}

bool isUpper(char c){
   if(                          ) return true;
   return false;
}

bool isSpace(char c){
   if(                          ) return true;
   return false;
}

bool isPunct(char c){
   if(                          ) return true;
   return false;
}
```

What about isDigit( ) and isCntrl( )?

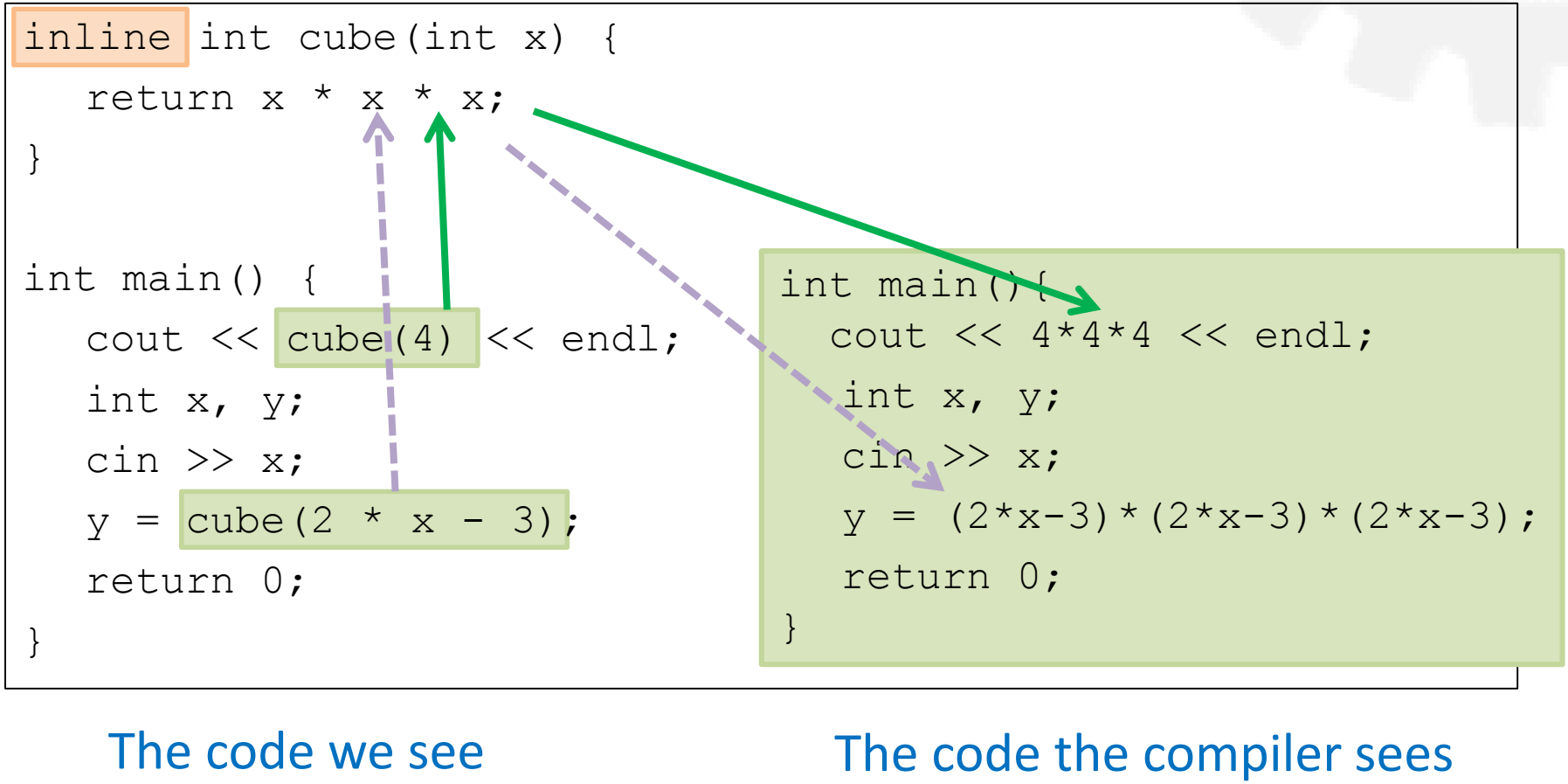
isPrime()

```
bool isPrime(int);

int main(){
   for(int i = 1; i <= 10; i++)
        if(isPrime(i))
              cout << i << " is a prime number." << endl;
   return 0;
}

bool isPrime(int n){
   float sqrtn = sqrt(n);
   if(n < 2) return false;
   if(n < 4) return true;
   if(n % 2 == 0) return false;
   for(int d = 3; d < sqrtn; d += 2)
        if(n % d == 0) return false;
   return true;
}
```

Inline Functions

- A function call requires substantial overhead.
  - Time and space to invoke the function and pass parameters to it
  - Storage for local variables in the function
  - Storage for variables and current location of execution in the main program
- In some cases, like for a short routine, it is better to avoid calling a function
  - Use inline function!
- Inline function: replace each function call with explicit code

Inline the cube( ) function



The code we see

The code the compiler sees