CSC404 PROGRAMMING II
FUNCTION PART 2

Fadilah Ezlina Bt. Shahbudin (FSKM, UiTM Kampus Jasin)

Local Variables

- **Local variable:** a variable defined inside a function.
- You can declare a local variable with the same name in different blocks.

Local Variables

A variable declared in the initial action part of a <u>for</u> loop header has its scope in the entire loop. But a variable declared inside a <u>for</u> loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

```
                                  void method1() {
                                    .
                                    .
                                  for (int i = 1; i < 10; i++)
                                  {
The scope of i                      .
                                    int j;
                                    .
The scope of j                      .
                                    .
                                  }
                                  }
```

Block Variables

- Variables declared within the body of a compound statement (block).
- They only exist within the block of statement.
- Example:

```
{
//x is a block variable;
int x;

}
```

Global Variables

- Variables declared outside of any function definition.
- Any function can access/change the global variables.
- Example:

```
float f1(int, char)
int x;

int main(){
//variable x can be accessed here
}

 float f1(int a, char b){
 //variable x also can be accessed here
}
```

Scope

- **Scope**: the part of the program where the variable can be referenced.
- The scope of a variable starts from its declaration and continues to the end of the block that contains the variable.
- A global variable has global (unlimited) scope.
- A local variable's scope is restricted to the function that declares the variable.
- A block variable's scope is restricted to the block in which the variable is declared.

User-Defined Functions

- Functions written by user to solve certain problem.
- Functions need to define before they can be used.

User-Defined Functions

- <u>Value-returning functions</u>: have a return type
  - Return a value of a specific data type using the `return` statement
- <u>Void functions</u>: do not have a return type
  - *Do not* use a `return` statement to return a value

Skeleton of Program using Functions

```
void printOne(int x);  ←  //function prototype
int main(){
int a=5;
printOne(5);  ←  //function call
return 0;
}

void printOne(int x){  ←  //function definition
cout<<x<<endl;
return;
}
```

Function Prototype

- Function prototype: function heading without the body of the function.
- Used to declare the function.
- Syntax: `functionType functionName(parameter list);`
- Example:

```
float f1(int a,char b);
```

- Not necessary to specify the variable name in the parameter list. Example:

```
float f1(int,char);
```

- Data type of each parameter must be specified.

Function Definition

- Used to define the function.
- Syntax:

```
functionType functionName(formal parameter list)
{
    statements
}
```

- Example:

```
double sqr(double x){
    //function body
    double y;
    y=x*x;
    return y;
}
```

Value-Returning Functions

- The function of a type other than void.
- Has a return statement that specifies the function's return value.
- Can use the value returned by a value-returning function by:
  - Saving it for further calculation
  - Using it in some calculation
  - Printing it
- A value-returning function is used in an assignment or in an output statement

Value-Returning Functions (cont'd.)

- Heading (or function header): first line of the function
  - Example: `int abs(int number)`
- Formal parameter: variable declared in the heading
  - Example: `number`
- Actual parameter: variable or expression listed in a call to a function
  - Example: `x = pow(u, v)`

Syntax: Formal Parameter List

```
dataType identifier, dataType identifier, ...
```

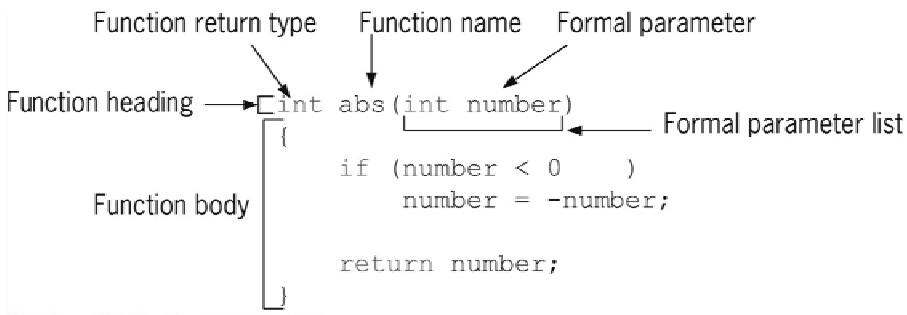


FIGURE 6-1 Various parts of the function abs

Syntax: Actual Parameter List

- Syntax of the actual parameter list:

```
expression or variable, expression or variable, ...
```

- Formal parameter list can be empty:

```
functionType functionName()
```

- A call to a value-returning function with an empty formal parameter list is:

```
functionName()
```

`return` Statement

- Function returns its value via the `return` statement
  - It passes this value outside the function
- Syntax: `return expr;`
- In C++, `return` is a reserved word
- When a return statement executes
  - Function immediately terminates
  - Control goes back to the caller
- When a `return` statement executes in the function `main`, the program terminates

Syntax: `return` Statement (cont'd.)

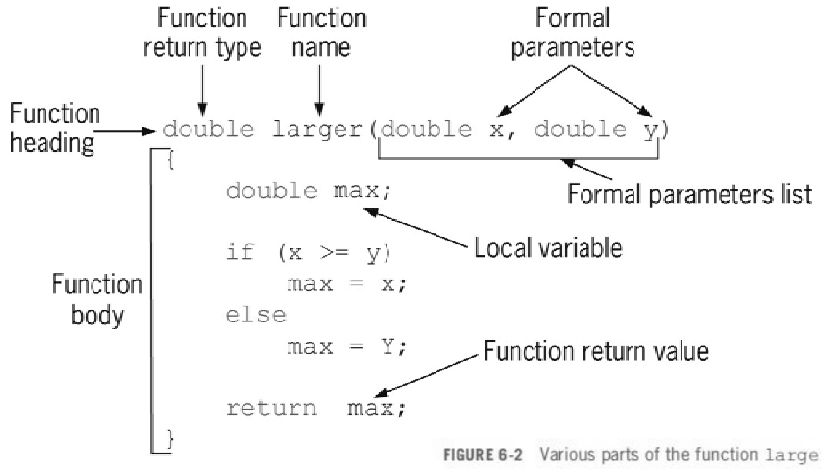


FIGURE 6-2 Various parts of the function `larger`

Value-Returning Functions: Some Peculiarities

```
int secret(int x)
{
    if (x > 5)          //Line 1
        return 2 * x;   //Line 2
}
```

A correct definition of the function **secret** is:

```
int secret(int x)
{
    if (x > 5)          //Line 1
        return 2 * x;   //Line 2

    return x;           //Line 3
}
```

Value-Returning Functions: Some Peculiarities (cont'd.)

```
return x, y;  //only the value of y will be returned


int funcRet1()
{
     int x = 45;

     return 23, x;  //only the value of x is returned
}

int funcRet2(int z)
{
     int a = 2;
     int b = 3;

     return 2 * a + b, z + b; //only the value of z + b is returned
}
```

Void Functions

- Can be placed either before or after the function `main`
- If void functions are placed after the function `main`
  - The function prototype must be placed before the function `main`
- Void function does not have a return type
  - `return` statement without any value is typically used to exit the function early

Void Functions (cont'd.)

- Formal parameters are optional.
- A call to a void function is a stand-alone statement. Example:

```
printTitle();
clrscr();
exit(n);
```

- Void function definition syntax:

```
void functionName(formal parameter list)
{
    statements
}
```

Void Functions (cont'd.)

- Formal parameter list syntax:

```
dataType& variable, dataType& variable, ...
```

- Function call syntax:

```
functionName(actual parameter list);
```

- Actual parameter list syntax:

```
expression or variable, expression or variable, ...
```

Passing by Value

- **Pass by value:** when an argument is passed to a function, a copy of its value is placed in the parameter.
- The function cannot access the original argument.
- Changes to the parameter in the function do not affect the value of the argument in the calling function.

Passing by Value

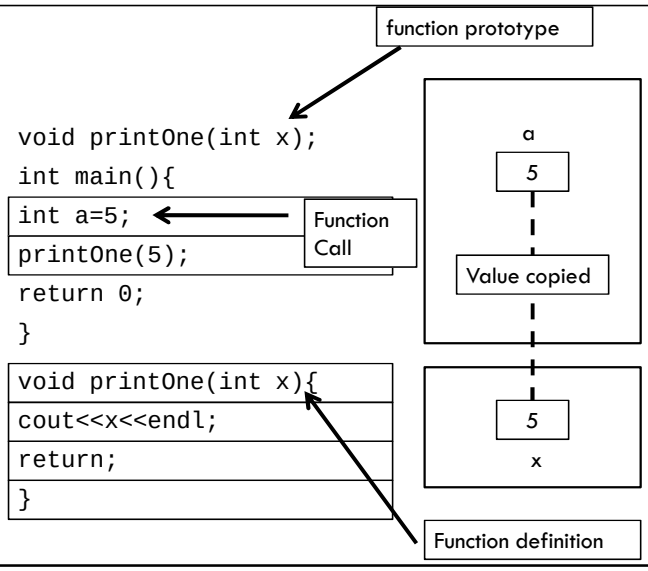
- Pass the value of argument to the function during function call.
- Every value of the arguments will be passed to the function parameter according to their sequence in the list.
- The initial value of the parameter is equal to the value of argument.

Passing by Value

- Example: `int val = 5;`
  `evenOrOdd(val);`

- **`evenOrOdd`** can change variable **`num`**, but it will have no effect on variable **`val`**

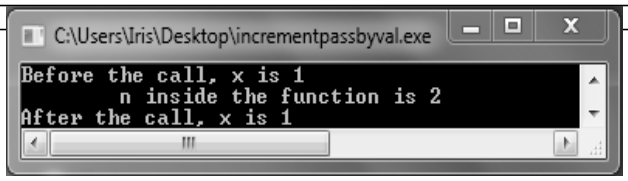
Passing by Value Example: void function with parameter



Passing by Value Example

- The following program creates a function for swapping two variables. The swap function is invoke by passing 2 arguments. However values of the arguments are not changed after the function is invoked.

Passing by Value Example



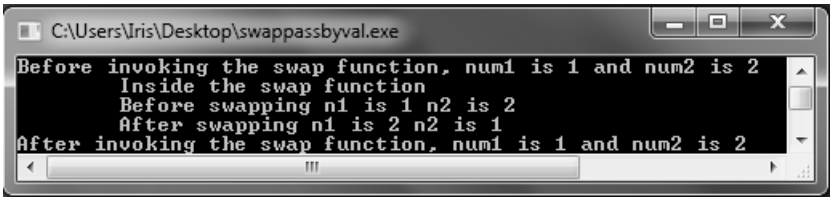

```
#include <iostream>
using namespace std;

int main(){
  int x = 1;
  cout << "Before the call, x is " << x << endl;
  increment(x);
  cout << “After the call, x is " << x << endl;
  return 0;
}

void increment(int n) {
  n++;
  cout << “\tn inside the function is " << n << endl;
}
```

Passing by Value Example

```
#include <iostream>
using namespace std;
void swap(int n1,int n2);        //function prototype
int main(){
  int num1 = 1;
  int num2 = 2;
  cout << "Before invoking the swap function, num1 is " << num1
   << " and num2 is "<< num2 << endl;
  //invoke the swap function to attempt to swap two variables
  swap(num1,num2);
  cout << "After invoking the swap function, num1 is " << num1
   << " and num2 is "<< num2 << endl;
  system("PAUSE");
  return 0;
}
```

Passing by Value Example

```
//Attempt to swap two variables does not work
void swap(int n1,int n2){  //function definition
cout<<"\tInside the swap function"<< endl;
cout<<"\tBefore swapping n1 is "<< n1 <<" n2 is "<< n2 <<endl;
//swap n1 with n2
int temp=n1;
n1=n2;
n2=temp;
cout<<"\tAfter swapping n1 is "<< n1 <<" n2 is "<< n2 <<endl;
}
```



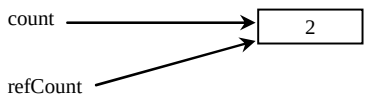
Passing by Reference

- C++ provides a special type of variable, called a reference variable, which can be used as a function parameter to reference the original variable.
- A reference variable is an alias for another variable.
- To declare a reference variable, place the ampersand (&) in front of the name.
- Example:

```
void getDimensions(int &, int &);
```

Passing by Reference

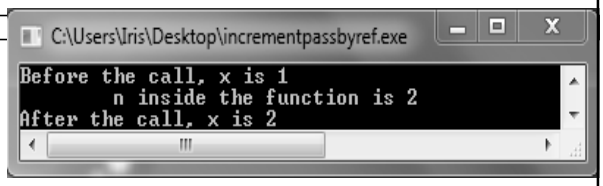
- Any changes made through the reference variable are actually performed on the original variable.



Passing by Reference Example

```
#include <iostream>
using namespace std;
void squareIt(int &);//function prototype

int main(){
int localVar = 5;
squareIt(localVar);  // function call,localVar now contains 25
cout<<localVar<<endl;
return 0;
}

void squareIt(int &num){ //function definition
      num *= num;
}
```

Passing by Reference Example




```
#include <iostream>
using namespace std;
void increment(int &n);

int main(){
  int x = 1;
  cout << "Before the call, x is " << x << endl;
  increment(x);              //function call
  cout << “After the call, x is " << x << endl;
  return 0;
}

void increment(int &n) {    //function definition
  n++;
  cout << “\tn inside the function is " << n << endl;
}
```

Passing by Reference Example

```
#include <iostream>
using namespace std;
void swap(int &n1,int &n2);  //function prototype

int main(){
  int num1 = 1;
  int num2 = 2;
  cout << "Before invoking the swap function, num1 is " << num1
   << " and num2 is "<< num2 << endl;
  //invoke the swap function to attempt to swap two variables
  swap(num1,num2);
  cout << "After invoking the swap function, num1 is " << num1
   << " and num2 is "<< num2 << endl;
  system("PAUSE");
  return 0;
}
```

Passing by Reference Example

```
void swap(int &n1,int &n2){             //function definition
cout<<"\tInside the swap function"<< endl;
cout<<"\tBefore swapping n1 is "<< n1 <<" n2 is "<<n2<<endl;

//swap n1 with n2
int temp=n1;
n1=n2;
n2=temp;
cout<<"\tAfter swapping n1 is "<< n1 <<" n2 is "<< n2 <<endl;
}
```