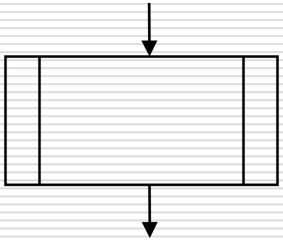
Ch6: Functions

- ❑Modular Programming – Functions
  - ❑Function Definition and Call
  - ❑Parameters and Return Statement
  - ❑Default Arguments
  - ❑Reference Variables
  - ❑Overloaded Functions
- ❑Local and Global Variables
- ❑Static Variables

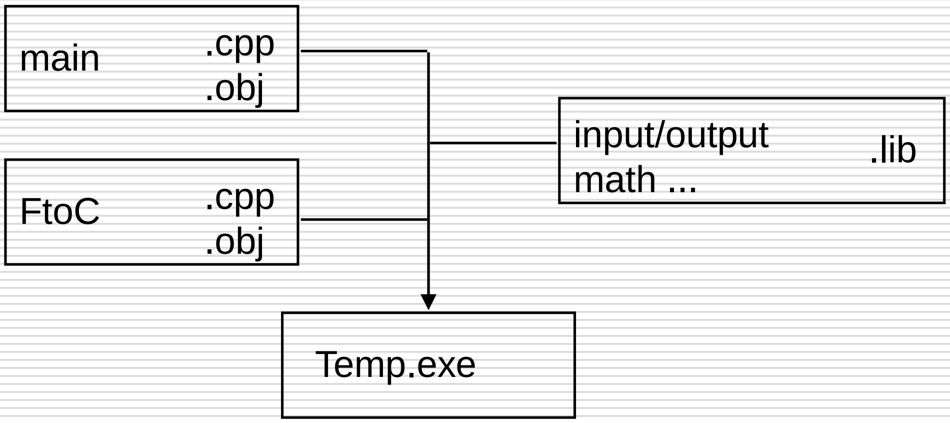
Modular Programming

- Program can be broken into manageable functions
  - Collection of statements that performs a specific task
- Independent functions can be reused
  - within same program
  - by another program



Modular Programming

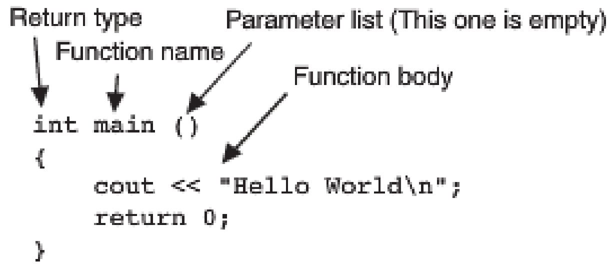
- Function definitions can exist in same or another file
  - object files are linked together to create executable



Function Definition

- Consists of:
  - return type → data sent *back* from function
  - name
  - parameter list → data sent *to* function
  - body → statements to execute

Figure 6-2



```
int main ()
{
      cout << "Hello World\n";
      return 0;
}
```

Function Call

- Function body will execute when called
  - main function called automatically when program starts

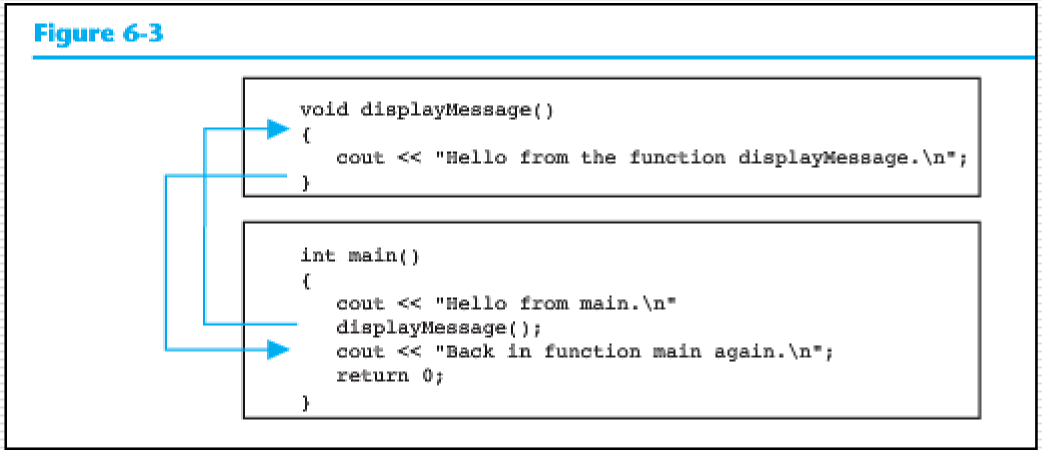


Figure 6-3

```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}

int main()
{
   cout << "Hello from main.\n"
   displayMessage();
   cout << "Back in function main again.\n";
   return 0;
}
```

Functions

```
#include <iostream>
using namespace std;
float ftoc(float);

int main() {
    float inpFahr, outCels;
    cout << “Input a Fahrenheit value: “;
    cin >> inpFahr;
    outCels = ftoc(inpFahr);
    cout << “Celsius = “ << outCels
         << endl;
    return 0;
}

float ftoc(float fahr) {
    float cels;
    cels = 5 * (fahr - 32) / 9;
    return cels;
}
```

function prototype

function call

function must be defined, or prototype given, before function is called

function header = return type + name + parameter list

function body

function definition

Functions

```
#include <iostream>
using namespace std;
float ftoc(float);

int main() {
   float inpFahr, outCels;
   cout << “Input a Fahrenheit value: “;
   cin >> inpFahr;
   outCels = ftoc(inpFahr);
   cout << “Celsius = “ << outCels
        << endl;
   return 0;
}

float ftoc(float fahr) {
   float cels;
   cels = 5 * (fahr - 32) / 9;
   return cels;
}
```

parameter names optional

argument

parameter names required

parameters are initialized local variables

Parameters

- Parameters are *(optional)* data transferred to function
  - Parentheses required in definition and call
  - Data type and parameter name required for *each* parameter in function definition
    - Implicit type coercion if different types
  - Multiple parameters/arguments are separated by commas
  - Parameters/arguments matched by relative positions
  - Used as initialized local variables

Return Statement

- Return statement causes function to end
  - Can give *(optional)* data back to calling function
  - Use of ‘void’ data type in header if no value returned
  - Return expression is converted to type returned by the function in which it appears
  - exit() function causes program to terminate without returning to calling function
  - Validation functions return boolean value

Functions

```
#include <iostream>
using namespace std;
void menuPrompt();
float ftoc(float);

int main(){
   float inpFahr, outCels;
   char inpChar;
   menuPrompt();
   cin >> inpChar;
   if(toupper(inpChar) == ‘F’)
   {
        cout << “Please input value: “;
        cin >> inpFahr;
        outCels = ftoc(inpFahr);
        cout << “Celsius = “ << outCels
                << endl;
   }
   return 0;
}
```

void function invocation

value-returning function invocation

Functions

```
float ftoc(float fahr) {
   float cels;
   cels = 5 * (fahr - 32) / 9;
   return cels;
}
```

value-returning function definition

```
void menuPrompt() {
   cout << “Input Menu\n“;
   cout << “ F: Fahr to Cels\n“;
   cout << “ Q: Quit\n“;
   return;
}
```

void function definition

Default Arguments

- Passed to parameters *automatically* if **no** argument is provided in the function call
- Assigned at *earliest* function occurrence
  - Prototype or definition, *but not both*
- Can have more than one default argument
  - Must be at the *end* of the argument list

Default Arguments

```
#include <iostream>
using namespace std;
float ftoc(float fahr = 32.0);

int main() {
    float inpFahr, outCels;
    cout << “Input a Fahrenheit value: “;
    cin >> inpFahr;
    outCels = ftoc(inpFahr);
    cout << “Celsius = “ << outCels;
    outCels = ftoc();
    cout << “Celsius = “ << outCels;
    return 0;
}

float ftoc(float fahr) {
    float cels;
    cels = 5 * (fahr - 32) / 9;
    return cels;
}
```

default argument

default argument used in function call

Default Arguments

```
#include <iostream>
using namespace std;
float ftoc(float = 32.0);

int main() {
    float inpFahr, outCels;
    cout << “Input a Fahrenheit value: “;
    cin >> inpFahr;
    outCels = ftoc(inpFahr);
    cout << “Celsius = “ << outCels;
    outCels = ftoc();
    cout << “Celsius = “ << outCels;
    return 0;
}

float ftoc(float fahr) {
    float cels;
    cels = 5 * (fahr - 32) / 9;
    return cels;
}
```

default argument

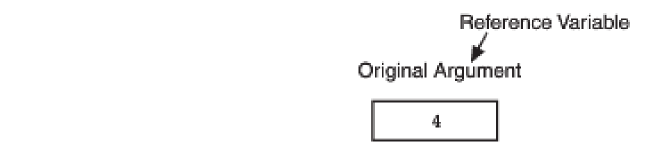
default argument used in function call

Reference Variables

- Allows function access to parameter’s original argument
  - changes to the parameter are also made to the argument
  - use & after data type in parameter list
  - argument must be a variable

```
   …
   int value = 4;
   DoubleNum(value);
   …
void DoubleNum(int &refVar) {
   refVar *= 2;
   return;
}
```

Figure 6-15



Reference Variables

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

void getVal(float&);

int main() {
    float inpFahr, outCels;
    cout << “Input a Fahrenheit value: “;
    getVal(inpFahr);
    outCels = ftoc(inpFahr);
    cout << “Celsius = “ << outCels;
    outCels = ftoc();
    cout << “Celsius = “ << outCels;
    return 0;
}

void getVal(float& fVar)
    {cin >> fVar;}
```

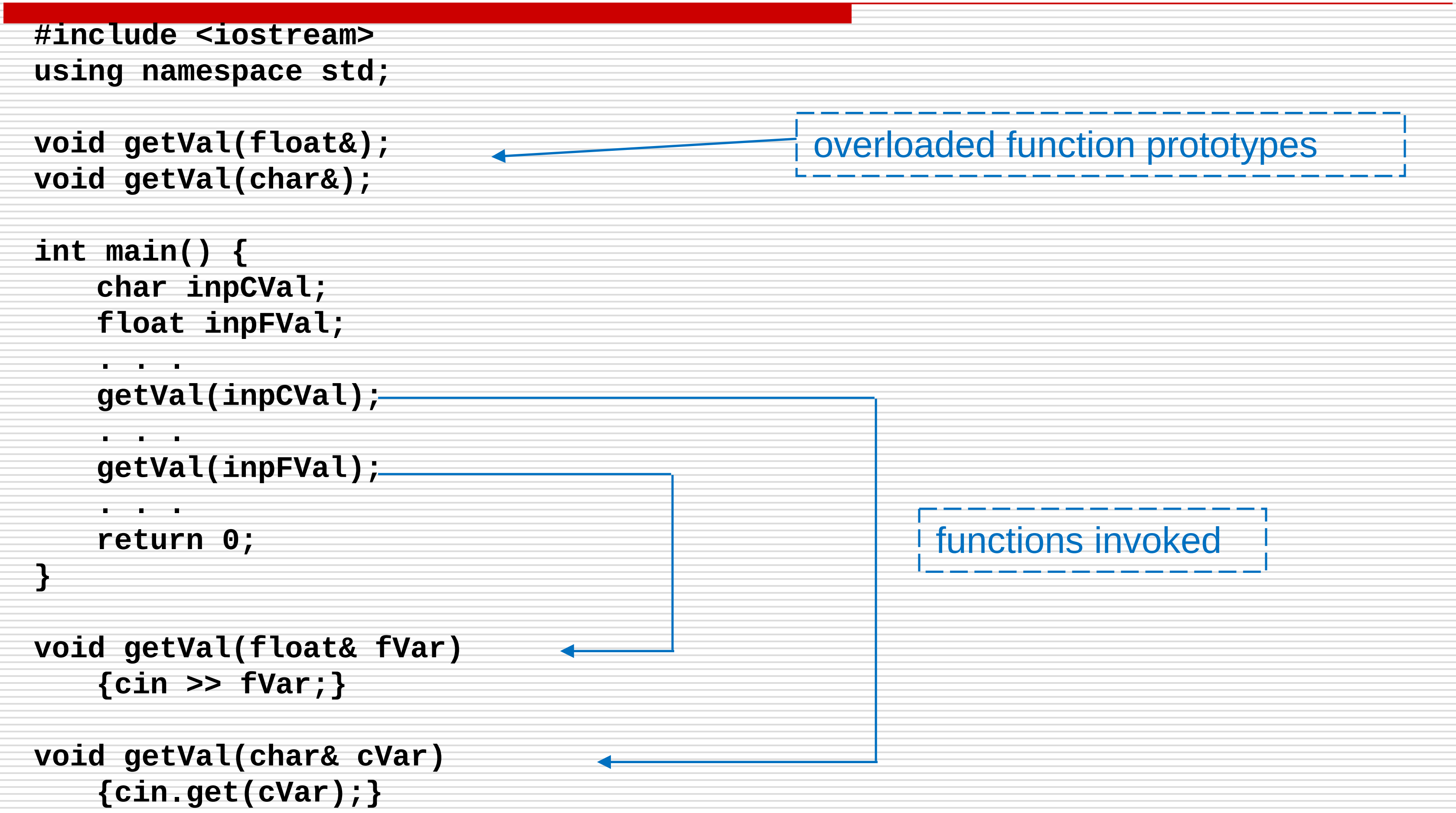
reference parameter

function invoked

Overloaded Functions

- More than one function may have the *same* name
  - each differs in the number or type of non-default parameters
- Function selection is determined at compile time
  - parameter type and number determines which of the functions is invoke

Overloaded Functions



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

void getVal(float&);
void getVal(char&);

int main() {
    char inpCVal;
    float inpFVal;
    . . .
    getVal(inpCVal);
    . . .
    getVal(inpFVal);
    . . .
    return 0;
}

void getVal(float& fVar)
    {cin >> fVar;}

void getVal(char& cVar)
    {cin.get(cVar);}
```

overloaded function prototypes

functions invoked

Local Variables

- Default for variables defined *within* function bodies and parameters
- Memory storage
  - allocated when control enters the variable containing block
  - released when control leaves its containing block
- Can nest blocks { } with function
- Variables in *different* functions/blocks can have *same* name

Global Variables

- Variables defined *outside* a function body
- Memory storage
  - allocated for life of program
  - automatically initialized to binary zero
- Global to all functions declared after it in current source file
  - may be “hidden” if redefined locally
- If local variable with *same name* then **local** variable takes **precedence**
  - use unary scope resolution operator to access global variable ::
- Global variables discouraged, but global constants generally permitted

Global and Local Variables

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

int gVar;

int main() {
    int gVar = 5;
    cout << “gVar = “ << gVar << endl;
    cout << “global gVar = “ << ::gVar << endl;
    return 0;
}
```

global variable

local variable

```
gVar = 5
global gVar = 0
```

Global Constants and Local Variables

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

const float FTOC_FACTOR = static_cast<float>(5)/9;
float ftoc(float);

int main() {
    float inpFahr, outCels;
    cout << “Input a Fahrenheit value: “;
    cin >> inpFahr;
    outCels = ftoc(inpFahr);
    cout << “Celsius = “ << outCels;
    outCels = ftoc();
    cout << “Celsius = “ << outCels;
    return 0;
}

float ftoc(float fahr) {
    float cels;
    cels = FTOC_FACTOR * (fahr - 32);
    return cels;
}
```

global **FTOC_FACTOR** constant

parameter **fahr** as local variable

local variable **cels**

Static Variables

- Keyword static must be included in definition
- Memory storage
  - allocated for life of program
  - automatically initialized to binary zero
- When declared inside function body
  - scope *only* in containing block (i.e. function)
  - only initialized *once* if value given in declaration

Static Variables

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

static int sgVar;

void testStaticVar(void);

int main() {
   cout << “sgVar = “ << sgVar << endl;
   for (int iVar = 0; iVar < 5; iVar++)
         testStaticVar();
   cout << “sgVar = “ << sgVar << endl;
   return 0;
}

void testStaticVar(void) {
   static int slVar = 10;
   cout << “slVar = “ << slVar--
         << “\n\tsgVar = “ << ++sgVar << endl;
   return;
}
```

static global variable

static local variable

```
sgVar = 0
slVar = 10
   sgVar = 1
slVar = 9
   sgVar = 2
slVar = 8
   sgVar = 3
slVar = 7
   sgVar = 4
slVar = 6
   sgVar = 5
sgVar = 5
```