

## Review

- Library Functions
  - #include <iostream>
  - #include <cmath>
- Overview User Defined Functions
  - Prototypes
  - Function Definition
  - Function Call
  - Parameter Passing
  - Return Value

Copyright © 2001 R.M. Laurie 1

## New

- More Details on User Functions
- Return by reference
- Scope and Lifetime
- Side effects

Copyright © 2001 R.M. Laurie 2

## Function Prototypes

- All identifiers must be declared in C++
- Prototype is a function declaration
- Parameter data type
  - blank ( ) type if none
- Return value data type
  - void type if none
- Always ends with a ;
- Types must match function definition

Copyright © 2001 R.M. Laurie 3

## Function Definition

- Function Definition: describes what the function does and returns
- Parameter List
  - Describes what data function needs
  - ( ) if no parameters are required
- Local Variables
  - Created for function
  - Includes parameter list

Copyright © 2001 R.M. Laurie 4

## Function Call (Invocation)

- Function call from main or another function.
- Argument List: Describes what data values function will use for call
  - ( ) if declared none
  - Must Match the type specified for Parameter List
    - ◆ Prototype
    - ◆ Function Definition

Copyright © 2001 R.M. Laurie 5

## Function Parameters

- Parameters: Variable declared in function definition heading.
 

```
void FindMaximum(int nX, int nY, int nZ)
```

  - Also called: Formal Argument, Formal Parameter
  - Data types must match the prototype
- Argument: Variable or expression listed in call to function.
 

```
FindMaximum(nA, nB, nC);
```

  - Also called: Actual Argument, Actual Parameters

Copyright © 2001 R.M. Laurie 6

## return(Value); Statement

- If function returns a data value it is described with `return(expression);`
- More than one return statement can be used in a function
- However, a function can only return one value
- `return;` is used in function to return to call if type void function

Copyright © 2001 R.M. Laurie 7

## Sum of Squares Example 1

```
#include <iostream>
void Introduction();
int InputNumber();
int CalcSumSquare(int);
void PrintPos(int);

using namespace std;
```

Copyright © 2001 R.M. Laurie 8

## Sum of Squares Example 2

```
int main()
{
    int nEntry, nResult;

    Introduction();
    while(1)
    {
        nEntry = InputNumber();
        if(nEntry < 0)
            break;
        nResult = CalcSumSquare(nEntry);
        cout << nResult << " <- The Sum of Squares\n\n";
    }
    cout << "Done\n";
    return 0;
}
```

Copyright © 2001 R.M. Laurie 9

## Sum of Squares Example 3

```
void Introduction()
{
    cout << "Welcome to the Sum of Squares Program\n"
         << "      by Bob Laurie\n\n"
         << "-----\n\n";
    return;
}

int InputNumber()
{
    int nInput;
    cout << "Enter a positive number (Quit = -1)\n>";
    cin >> nInput;
    return(nInput);
}
```

Copyright © 2001 R.M. Laurie 10

## Sum of Squares Example 4

```
int CalcSumSquare(int nVal)
{
    int nProduct, nSum = 0;
    while(nVal > 0)
    {
        nProduct = nVal * nVal;
        cout << nProduct;
        PrintPos(nVal);
        nSum = nSum + nProduct;
        nVal--;
    }
    return nSum;
}

void PrintPos(int nValue)
{
    if(nValue > 1)
        cout << " + ";
    else
        cout << endl;
    return;
}
```

Copyright © 2001 R.M. Laurie 11

## Example using multiple returns

```
#include <iostream>
char FindGrade(int);
using namespace std;
main()
{
    int nGrade;
    cout << "Enter Score: \n>";
    cin >> nGrade;
    cout << "Grade is: "
         << FindGrade(nGrade);
    return 0;
}
```

Copyright © 2001 R.M. Laurie 12

```

char FindGrade(int nPercent)
{
    if (nPercent >= 90) return 'A';
    if (nPercent >= 80) return 'B';
    if (nPercent >= 70) return 'C';
    if (nPercent >= 60) return 'D';
    return 'F';
}

```

Copyright © 2001 R.M. Laurie 13

## Scope

- Local Scope: Variable or constant declared within a function is only accessible in function.
- Global Scope: Variable or constant declared outside of all function is accessible in all.
- Good to denote global variables with a "g" prescript. e.g. gnMaximum
- Name Precedence = Local
- Avoid global variables as break encapsulation
- extern is used to declare an external global variable that is declared in another file

Copyright © 2001 R.M. Laurie 14

## Lifetime

- Lifetime is the duration for which an identifier remains allocated in memory.
  - Function Lifetime
  - Program Lifetime
- Automatic Variable (default)
  - Function Lifetime
  - Created at function call and destroyed on return
- Static Variable
  - Scope is within function
  - Program Lifetime -Value is retained between calls
- Global Variables have Program Lifetime

Copyright © 2001 R.M. Laurie 15

## Global Variable Example

```

// Find the Maximum of Three Integers
#include <iostream>
void FindMaximum(int, int, int); // Function Prototype
void PrintMaximum(void); // Function Prototype
using namespace std;
int gnMax; // Global Variable
main()
{
    int nA, nB, nC;
    cout << "Enter three integers: ";
    cin >> nA >> nB >> nC;
    FindMaximum(nA, nB, nC); // Function Call
    PrintMaximum(); // Function Call
    return 0;
}

```

Copyright © 2001 R.M. Laurie 16

```

Enter three integers: 35 50 45
Maximum is: 50

```

```

// Function FindMaximum Definition
void FindMaximum(int nX, int nY, int nZ)
{
    gnMax = nX;
    if (nY > gnMax)
        gnMax = nY;
    if (nZ > gnMax)
        gnMax = nZ;
}
// Function PrintMaximum Definition
void PrintMaximum(void)
{
    cout << "Maximum is: " << gnMax << endl;
}

```

Copyright © 2001 R.M. Laurie 17

## Types of Parameters

- Value Parameters
  - Formal parameter receives a copy of the contents of the actual parameter
  - May include expressions, variables, or constants
  - Can not modify actual parameter value
- Reference Parameters
  - Passes the address of a variable
  - No Expressions or constants
  - Can utilize and modify actual parameter
  - Primary use is when a function produces more than one result

Copyright © 2001 R.M. Laurie 18

## Reference Example - 1

```
#include <iostream>
void PlusMinus(int&, int&);
using namespace std;
int main()
{
    int nA, nB;
    cout << "Enter two values A and B: \n>";
    cin >> nA >> nB;
    PlusMinus(nA, nB);
    cout << "A + B = " << nA
         << "\nA - B = " << nB;
    return 0;
}
```

Copyright © 2001 R.M. Laurie 19

## Reference Example - 2

```
void PlusMinus(int& nFirst, int& nSecond)
{
    int nX = nFirst;
    int nY = nSecond;
    nFirst = nX + nY;
    nSecond = nX - nY;
}
```

Copyright © 2001 R.M. Laurie 20

## Reference Example - 3

```
#include <iostream>
using namespace std;
void PlusMinus(int&, int&);
int main(void)
{
    int nA, nB;
    cout << "Enter two values A and B: \n>";
    cin >> nA >> nB;
    PlusMinus(nA, nB);
    cout << "A + B = " << nA << "\nA - B = " << nB;
    return 0;
}
void PlusMinus(int& nFirst, int& nSecond)
{
    int nX = nFirst;
    int nY = nSecond;
    nFirst = nX + nY;
    nSecond = nX - nY;
}
```

Copyright © 2001 R.M. Laurie 21

## Designing Functions - 1

- Top Down Approach to Programs
- Code Reuse
- Encapsulation
  - Hides implementation details
  - Higher level of design (call and use)
  - Specified Interface
- Interface
  - Formal description of purpose for function
  - Precondition (Parameter requirements)
  - Postcondition (actions, return values, references)

Copyright © 2001 R.M. Laurie 22

## Designing Functions - 2

- Use verb-noun function name
  - Use C style comments /\* Comment \*/ to define parameter data flow p. 359
    - /\*in\*/ Pass by value
    - /\*out\*/ Pass by reference
    - /\*inout\*/ Pass by value
- ```
void FindMaximum(/*in*/ int nX,
                /*in*/ int nY, /*in*/ int nZ)
char FindGrade(/*in*/ int nPercent)
void PlusMinus(/*inout*/ int& nFirst,
              /*inout*/ int& nSecond)
```

Copyright © 2001 R.M. Laurie 23

## Designing Functions - 3

- Describe Interface using Comments
  - Formal description of purpose for function
  - Precondition (Parameter requirements)
  - Postcondition (actions, return values, references)

```
void PlusMinus(/*inout*/ int& nFirst, /*inout*/ int& nSecond)
{
    // This function will determine the sum and difference .
    // Postcondition: Reference to the first parameter is sum.
    // Reference to the second parameter is the difference.
    int nX = nFirst;
    int nY = nSecond;
    nFirst = nX + nY;
    nSecond = nX - nY;
}
```

Copyright © 2001 R.M. Laurie 24

## Test and Debugging Functions

- Test functions:
  - Individually using program with function call.
  - Test for different parameter values
- Test Programs
  - Verify the proper type match for prototypes, function definition, and function calls
  - Ensure preconditions are met at function call
- Avoid use of global variables
- Debugging function
  - Use break points, watch, and trace
  - Use assert to halt if logic expression False p.374
  - See hints p. 376

Copyright © 2001 R.M. Laurie 25

## Side Effects

- Any effect of a function that is not part of the explicitly defined interface.
- Caused by careless coding.
- Prevent by using Pass by Value for all incoming data flow parameters.
- Prevent by not utilizing or modifying global variables in functions.
- Often difficult to debug as may effect another region of program.
- If you want to make something global, make it a constant. Other functions cannot modify.

Copyright © 2001 R.M. Laurie 26