Functions

CSPROG2
Computer Programming 2 for CS

Specific Objective

- Learn about standard (predefined) functions and discover how to use them in a program
- Learn about user defined functions
- Examine value-returning functions, including actual and formal parameters
- Explore how to construct and use a value-returning, user-defined function in a program

Functions

- Procedure, subprograms, and method
- A function may return a value (produce a value) or may perform some action without returning a value.
- Functions that do not return a value are called **void functions**.

PREDEFINED FUNCTIONS

- C++ comes with libraries of predefined functions that you can use in your programs

Functions

NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0,3.0)	8.0	cmath
abs	Absolute value for int	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for long	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for double	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath
ceil	Ceiling (round up)	double	double	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor	Floor (round down)	double	double	floor(3.2) floor(3.9)	3.0 3.0	cmath
exit	End program	int	void	exit(1)	None	cstdlib
rnd	Random Number	None	int	rand( )	Varies	cstdlib
srand	Set seed for rand	unsigned int	void	srand(42)	None	cstdlib

Predefined Function

- **Example Program:**
- **Source Code:**

```
//Computes the size of a doghouse that can be purchased
//given the user's budget
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   const double COST_PER_SQ_FT = 100.5;
   double budget, area, lengthSide;

   cout << "Enter the amount budgeted for your doghouse PhP ";
   cin >> budget;

   area = budget/COST_PER_SQ_FT;
   lengthSide = sqrt(area);

   cout.setf(ios::fixed);
   cout.setf(ios::showpoint);
   cout.precision(2);

   cout << "For a price of PhP " << budget << endl
        << "I can build you a luxurious square doghouse\n"
        << "That is " << lengthSide
        << " feet on each side.\n";

   system("pause > 0");
   return 0;
}
```

Predefined Function

- Output

```
Enter the amount budgeted for your doghouse PhP 1200
For a price of PhP 1200.00
I can build you a luxurious square doghouse
That is 3.46 feet on each side.
```

Predefined Function

- **Random Number Generator**
- You can use a random number generator to simulate random events, such as the result throwing a dice or plopping a coin.

- rand( ) - A function that returns a “randomly chosen” number.
- srand(int value) - A function used to set the seed number of a pseudorandom numbers.

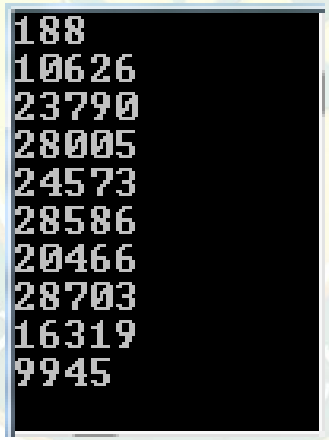
Predefined Function

Example Program:

Source Code:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    srand(46);
    for(int i=0;i<10;i++)
        cout << rand() << endl;
    system("pause > 0");
    return 0;
}
```

Output:



User Defined Function

You can define your own functions, either in the same file as main part of your program or in a separate file so that the functions can be used by several different programs.

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

double totalCost(int numberParameter, double priceParameter);
//or
//double totalCost(int, double);

//Computes the total cost, include 5% sales tax
//on numberParameter items at a cost of priceParameter each.
```

Function declaration; also called the function prototype

User Defined Function (cont.)

```
int main()
{
   double price, bill;
   int number;

   cout << "Enter the number of items purchased: ";
   cin >> number;
   cout << "Enter the price per item PhP ";
   cin >> price;

   bill = totalCost(number, price);

   cout.setf(ios::fixed);
   cout.setf(ios::showpoint);
   cout.precision(2);

   cout << number << " items at "
        << "PhP " << price << " each.\n"
        << "Final bill, including tax, is PhP " << bill
        << endl;
```

Function call

User Defined Function (cont.)

```
    system("pause > 0");
    return 0;
}

double totalCost(int numberParameter, double priceParameter)
{
    const double TAXRATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal*TAXRATE);
}
```

Function head

Function Body

Function Definition

```
Enter the number of items purchased: 12
Enter the price per item PhP 100
12 items at PhP 100.00 each.
Final bill, including tax, is PhP 1260.00
```

User defined Function (cont.)

- The identifiers *numberParameter* and *priceParameter* are called **formal parameters**, or **parameters** for short.

- A **function definition** describes how the function computes the value it returns.

User defined Function (cont.)

Example Program:
Source Code:

```
#include <iostream>
#include <cmath>
using namespace std;
int rnd(double number);
//Assumes number >= 0
//Returns number rounded to the nearest integer
int main()
{
   double doubleValue;
   char ans;
   do{
      cout << "Enter a double value: ";
      cin >> doubleValue;
      cout << "Rounded that number is " << round(doubleValue) << endl;
      cout << "Again? (y/n): ";
      cin >> ans;
   }while(ans=='y' || ans=='Y');
   cout << "End of Testing.\n";
   system("pause > 0");
   return 0;
}
int rnd(double number){
    return static_cast<int>(floor(number+0.5));
}
```

User Defined Function (cont.)

- Output

```
Enter a double value: 10.5
Rounded that number is 11
Again? (y/n): y
Enter a double value: 7.4
Rounded that number is 7
Again? (y/n): n
```

User Defined Function

Preconditions and Post conditions Functions Comment

- Pre-condition functions comment

States what is assumed to be true when the function is called. The function should not be used and cannot be expected to perform correctly unless the precondition holds.

- Post-condition function comment

Describes the effect of the function call; that is, the post-condition tells what will be true after the function is executed in a situation in which the precondition holds.

Example

```
void showInterest(double balance, double rate);
//Precondition: balance is a nonnegative savings acount.
//rate is the interest rate expressed as a percentage, such as 5 for 5%.
//Postcondition: The amount of interest on the given balance
//at the given rate is shown on the screen


double celsius(double fahrenheit);
//Precondition: farenheit is a temperature in degrees Fahrenheit.
//Returns the equivalent temperature expressed in degrees Celsius
```

Example

```
#include <iostream>
using namespace std;
void iceCreamDivision(int number, double totalWeight);
//Outputs instructions for dividing totalWeight ounces of ice cream among
//number customers. If number is 0, only an error message is output
int main()
{
    int number;
    double totalWeight;
    cout << "Enter the number of customers: ";
    cin >> number;
    cout << "Enter weight of ice cream to divide (in ounces): ";
    cin >> totalWeight;
    iceCreamDivision(number, totalWeight);
    system("pause > 0");
    return 0;
}

void iceCreamDivision(int number, double totalWeight)
{
    double portion;
    if(number == 0)
    {
        cout << "Cannot divide among zero customers.\n";
        return; //if number is 0, then the function execution ends here.
    }
    portion = totalWeight/number;
    cout << "Each one receives "
         << portion << " ounces of ice cream." << endl;
}
```

Output

- Case 1

```
Enter the number of customers: 10
Enter weight of ice cream to divide (in ounces): 3
Each one receives 0.3 ounces of ice cream.
```

- Case 2

```
Enter the number of customers: 0
Enter weight of ice cream to divide (in ounces): 5
Cannot divide among zero customers.
```

Recursion

A function definition that includes a call to itself is said to be recursive.

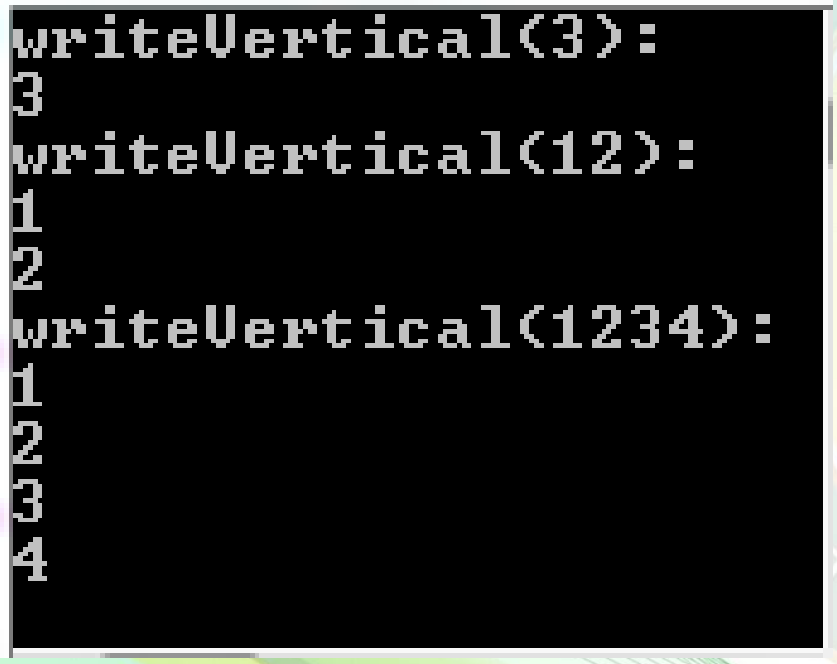
```
writeVertical(3)
    if (3<10)
        print 3
writeVertical(12)   // n=12
    else
        writeVertical(12/10) // n=1
            if (1<10)
                print 1   next line
        print 12%10    ====   2
writeVertical(1234)   // n=1234
    else
        writeVertical(1234/10)  // n = 123
            else
                writeVertical(123/10)  // n=12
                    else
                        writeVertical(12/10) // n=1
                            if (1<10)
                                print 1
                        print (12%10)   ===== 2
                print (123%10)   ====== 3
        print (1234%10)    =======   4
```

Example Program (Recursive void function)

```
//Program to demonstrate the recursive function writeVertical
#include <iostream>
using namespace std;

void writeVertical(int n);
//Precondition: n >= 0.
//Postcondition: THe number n is written to the screen vertically,
//with each digit on a seperate line.

int main()
{
   cout << "writeVertical(3): " << endl;
   writeVertical(3);
   cout << "writeVertical(12): " << endl;
   writeVertical(12);
   cout << "writeVertical(1234): " << endl;
   writeVertical(1234);
   system("pause > 0");
   return 0;
}

void writeVertical(int n)
{
   if(n < 10)
      cout << n << endl;
   else
   {
      writeVertical(n/10);
      cout << (n%10) << endl;
   }
}
```

```
writeVertical(3):
3
writeVertical(12):
1
2
writeVertical(1234):
1
2
3
4
```

Output


writeVertical(3):
3
writeVertical(12):
1
2
writeVertical(1234):
1
2
3
4

Recursive function that return a value

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

int power(int x, int n);
//Precondition: n >= 0.
//Return x to the power n.

int main()
{
    for(int n=0; n<4; n++)
        cout << "3 to the power " << n
             << " is " << power(3,n) << endl;
    system("pause > 0");
    return 0;
}

int power(int x, int n)
{
    if(n<0)
    {
        cout << "Illegal argument to power.\n";
        exit(1);
    }
    if(n>0)
        return(power(x, n-1)*x);
    else // n==0
        return(1);
}
```

3 to the power 0 is 1
3 to the power 1 is 3
3 to the power 2 is 9
3 to the power 3 is 27

Output

```
3 to the power 0 is 1
3 to the power 1 is 3
3 to the power 2 is 9
3 to the power 3 is 27
```

n = 0
return (1)
n = 1 x = 3
return (power(3, 0) * 3)
return (1)
n=2 x=3
return (power(3,1) * 3)
return (power(3,0) * 3)
return(1)
n=3 x=3
return(power(3,2)*3)
return(power(3,1)*3)
return(power(3,0)*3)
return(1)
n=4 out of for loop

Overloading

Giving two (or more) function definitions for the same function name is called **overloading** the function name.

The function definitions must have different numbers of formal parameters or some formal parameters of different types. When there is a function call, the compiler uses the function definition whose number of formal parameters and types of formal parameters match the arguments in the function call.

You cannot overload a function name by giving two definitions that differ only in the type of the value returned.

Overloading

Example 1

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

//Function Declaration
double ave(double n1, double n2);
double ave(double n1, double n2, double n3);

//Main Function
int main()
{
    double a,b,c;
    cout<< "Enter two numbers: ";
    cin >> a >> b;
    cout<< "The average is "
        << ave(a,b);//function ave that has two arguments
    cout << "\n\nEnter three numbers: ";
    cin >> a >> b >> c;
    cout<< "The average is "
        << ave(a,b,c);//function ave that has three arguments
    system("pause > 0");
    return 0;
}
```

Overloading (cont.)

```
//Function Definition
double ave(double n1, double n2)
{
    return ((n1+n2)/2.0);
}

double ave(double n1, double n2, double n3)
{
    return ((n1+n2+n3)/3.0);
}
```

Output

```
Enter two numbers: 10 20
The average is 15

Enter three numbers: 5 10 15
The average is 10
```

Overloading

Example 2

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

//Function declaration overloading
double area(double l, double w);
double area(double r);

//Main function
int main()
{
     double length,width,radius;
     cout << "Enter the radius of a circle: ";
     cin >> radius;
     cout << "Enter the length and width of a rectangle: ";
     cin >> length >> width;

     //Set the output value to two decimal places
     cout.setf(ios::fixed);
     cout.setf(ios::showpoint);
     cout.precision(2);

     cout << "\nArea (sq. units)";
     cout << "\nCircle: " << area(radius);
     cout << "\nRectangle: " << area(length, width);
```

Overloading (cont.)

```
    system("pause > 0");
    return 0;
}


//Function definition
//Rectangle
double area(double l, double w)
{
    return (l*w);
}
//Circle
double area(double r)
{
    const double PI = 3.14159;
    return  (PI * r * r);
}
```

Output

```
Enter the radius of a circle: 80
Enter the length and width of a rectangle: 50 11.5

Area (sq. units)
Circle: 20106.18
Rectangle: 575.00
```

Default Arguments

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

//Function declaration with w and h having a default argument value of 1
void showVolume(int l, int w=1, int h=1);

//Main function
int main()
{
    showVolume(4,6,2);
    showVolume(4,6);
    showVolume(4);
    system("pause > 0");
    return 0;
}

//Function definition
void showVolume(int l, int w, int h)
{
    cout << "\nVolume is "
        << (l*w*h)
        << " cu. units";
}
```

Output

```
Volume is 48 cu. units
Volume is 24 cu. units
Volume is 4 cu. units
```

Array in Functions

- You can use both array indexed variables and entire arrays as arguments to functions.
- An indexed variable can be an argument to a function in exactly the same way that any variable of the array base type can be an argument.
- The following are legal

```
double I, n, a[10];
myFunction(a[0]);
myFunction(a[3]);
```

Array in Functions

- An indexed variable can be call-by-value argument or a call-by-reference argument.
- A function can have a formal parameter for an entire array so that when the function is called, the argument that is plugged in for this formal parameter is an entire array.
- A formal parameter for an entire array is neither a call-by-value parameter nor a call-by-reference parameter. (array parameter)

Array in Functions

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

//Function declaration
void fillUp(int a[], int size);

//Main function
int main()
{
    int score[5], numberOfScores=5;
    fillUp(score, numberOfScores);
    cout << "\nThe scores are\n";
    for(int i=0;i<numberOfScores;i++)
        cout << score[i] << " ";
    system("pause > 0");
    return 0;
}

//Function definition
void fillUp(int a[], int size)
{
    cout << "Enter " << size << " numbers:\n";
    for(int i=0;i<size;i++)
        cin >> a[i];
}
```

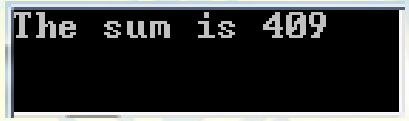
Output

```
Enter 5 numbers:
90 88 76 89 80

The scores are
90 88 76 89 80
```

Array in Functions

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

//Function declaration
int sum(int s[]);

//Main function
int main()
{
     int score[5]={80,88,70,81,90};
     cout << "The sum is "
          << sum(score);
     system("pause > 0");
     return 0;
}

//Function definition
int sum(int s[])
{
     int sum(0);
     for(int i=0;i<5;i++)
          sum+=s[i];
     return sum;
}
```

Output



Call-by-Reference Parameters

- To make a formal parameter a call-by-reference parameter, append the ampersand sign, &, to its type name

Call-by-Reference Parameters

```
#include <iostream>
using namespace std;
void swapValues(int& a, int& b);
int main()
{
    int n1, n2;
    cout << "Enter two integers value: ";
    cin >> n1 >> n2;
    cout << "n1 = " << n1 << endl;
    cout << "n2 = " << n2 << endl;
    cout << "\nCalling the function swapValue()\n";
    swapValues(n1,n2);
    cout << "n1 = " << n1 << endl;
    cout << "n2 = " << n2 << endl;
    system("pause>0");
    return 0;
}
void swapValues(int& a, int& b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

Output:

```
Enter two integers value: 10 20
n1 = 10
n2 = 20

Calling the function swapValue()
n1 = 20
```