Functions

http://net.pku.edu.cn/~course/cs101/2008

Hongfei Yan
School of EECS, Peking University
11/12/2008
Outline

• function definition and declaration
• Scope of variables
• Arguments passed
• Default values in parameters, overloaded functions, and function templates
• Recursivity
Function definition

- **type** name ( parameter1, parameter2, ...) { statements }
- **where:**
  - type is the data type specifier of the data returned by the function.
  - name is the identifier by which it will be possible to call the function.
  - parameters (as many as needed):
    - Each parameter consists of a data type specifier followed by an identifier,
    - and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called.
    - The different parameters are separated by commas.
  - statements is the function's body.
    - It is a block of statements surrounded by braces { }. 

#include <iostream>
using namespace std;

int addition (int a, int b) {
    int r;
    r = a + b;
    return (r);
}

int main () {
    int z;
    z = addition (5, 3);
    cout << "The result is " << z;
}

The result is 8
At the point at which the function is called from within main,
  - the control is lost by main and passed to function addition.

The value of both arguments passed in the call (5 and 3)
  - are copied to the local variables int a and int b within the function.
finalizes function addition

```c
int addition (int a, int b)
{
    z = addition ( 5 , 3 );
}
```

• The content of z becomes the value of evaluating the function call.
• **returns the control back** to the function that called it in the first place (in this case, main).
• At this moment the program follows its regular course from **the same point at which it was interrupted** by the call to addition.
#include <iostream>
using namespace std;

int subtraction (int a, int b)
{
    int r;
    r=a-b;
    return (r);
}

int main ()
{
    int x=5, y=3, z;
    z = subtraction (7,2);
    cout << "The first result is " << z << "\n";
    cout << "The second result is " << subtraction (7,2) << "\n";
    cout << "The third result is " << subtraction (x,y) << "\n";
    z= 4 + subtraction (x,y);
    cout << "The fourth result is " << z << "\n";
}
Functions with no type (the use of void)

#include <iostream>
using namespace std;

void printmessage ()
{
    cout << "I'm a function!";
}

int main ()
{
    printmessage ();
}
Declaring a prototype of the function before it is used

- type name (argument_type1, argument_type2, ...);
- The parameter enumeration does not need to include the identifiers, but only the type specifiers.
- The inclusion of a name for each parameter as in the function definition is optional in the prototype declaration.

```
int protofunction (int first, int second);
int protofunction (int, int);
```
```cpp
#include <iostream>
using namespace std;

void odd (int a);
void even (int a);

int main ()
{
    int i;
do {
        cout << "Type a number (0 to exit): ";cin >> i;
        odd (i);
    } while (i!=0);
}

void odd (int a)
{
    if ((a%2)!=0) cout << "Number is odd.\n";
    else even (a);
}

void even (int a)
{
    if ((a%2)==0) cout << "Number is even.\n";else odd (a);
}
```

Type a number (0 to exit): 9
Number is odd.
Type a number (0 to exit): 6
Number is even.
Type a number (0 to exit): 1030
Number is even.
Type a number (0 to exit): 0
Number is even.
#include <iostream>
#include <string>
using namespace std;

const char table[]={"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F");

char change(string w,int l)
{
    int r=0;
    for (int i=0; i<l;i++)
        r = r*2 + w[i] - '0';
    return table[r];
}

int main()
{
    int n;
    cin >> n;
    while (n--) {
        string num;
        cin >> num;
        int len=num.size();

        string subnum,result;
        result.clear();
        while (len>=4) {
            subnum = num.substr(len-4,4); // Generate substring
            result = change(subnum,4)+result;
            len -= 4;
        }
        if (len) {
            subnum = num.substr(0,len); 
            result = change(subnum,len)+result;
        }
        cout << result <<endl;
    }
}
Outline

- function definition and declaration
- **Scope of variables**
- Arguments passed
- Default values in parameters, overloaded functions, and function templates
- Recursivity
Scope of variables

- The scope of variables declared within a function or any other inner block **cannot** be used outside of them.
Arguments passed

• By value
  – what we have passed to the function were copies of their values
  – but never the variables themselves.

• By reference
  – we are somehow passing the variable itself to the function and
  – any modification that we do to the local variables will have an effect in their counterpart variables passed as arguments in the call to the function.
passing parameters by reference

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

void duplicate (int& a, int& b, int& c)
{
    a <<= 1;
    b <<= 1;
    c <<= 1;
}

int main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout << "x=" << x << " , y=" << y << " , z=" << z;
}
```

The type of each parameter was followed by an ampersand sign (&).

The output is:

```
x=2, y=6, z=14
```
#include <iostream>
using namespace std;

void prevnext (int x, int& prev, int& next)
{
    prev = x-1;
    next = x+1;
}

int main ()
{
    int x=100, y, z;
    prevnext (x, y, z);
    cout << "Previous=" << y << ", Next=" << z;
}
Outline

• Functions
  – function definition and declaration
  – Scope of variables
  – Arguments passed
  – Default values in parameters, overloaded functions, and function templates
  – Recursivity
#include <iostream>
using namespace std;

int divide (int a, int b=2) 
{
    int r;
    r=a/b;
    return (r);
}

int main ()
{
    cout << divide (12);
    cout << divide (20,4);
    cout << endl;
}

• If a value for that parameter is not passed when the function is called,
  ➢ the default value is used,
  ➢ but if a value is specified this default value is ignored and the passed value is used instead.
#include <iostream>
using namespace std;

int operate (int a, int b)
{
    return (a*b);
}

float operate (float a, float b)
{
    return (a/b);
}

int main ()
{
    int x=5,y=2;
    float n=5.0,m=2.0;
    cout << operate (x,y);
    cout << "\n";
    cout << operate (n,m);
    cout << "\n";
}
Function templates

#include <iostream>
using namespace std;

template <class T>
T operate (T a, T b)
{
    return (a/b);
}

int main ()
{
    int x=5,y=2;
    float n=5.0,m=2.0;
    cout << operate (x,y);
    cout << "\n";
    cout << operate (n,m);
    cout << "\n";
}
Outline

• function definition and declaration
• Scope of variables
• Arguments passed
• Default values in parameters, overloaded functions, and function templates
• Recursivity
#include <iostream>
using namespace std;

long long factorial (long long a) {
    if (a > 1) {
        return (a * factorial (a-1));
    } else {
        return (1);
    }
}

int main () {
    long long number;
    //cout << sizeof(long long) << endl;
    cout << "Please type a number: ";
    cin >> number;
    cout << number << "! = " << factorial (number);
}
references

• Functions: