Programming in C/C++

Patrick Ho
peiqistar@gmail.com

(Class 4)

Function

- A function is a named block of code that performs a task to control caller
- A function is often executed (called) several times
- A function has a name,
- A function’s inputs are known as its arguments
- A function has a return value

- General format

```
    type name(type1 arg1, type2 arg2, ...)
    {
          /* code */
     }
```

Function Example (1)

- A function to calculate average of 2 double numbers

```
double findaverage(double a, double b)
{
        double average;
        average = (a+b)/2;
        return(average);
}   // call this; double val=findaverage(10., 20.);
```

- A function can be *void* type, which has return without value

```
void printInteger(int i)
{
        printf(“integer=%d\n”, i);
        return;
}   // call this; printInteger(10);
```

Function Example (2)

- A function argument can be an array using [ ]

```
double findaverage(int size, double list[])
{
        double sum=0.0;
        for (int i=0; i<size; i++)
                sum += list[i];
        return(sum/size);
}
```

- A function argument can be pointer using * (or **)

```
double findaverage(int size, double *list)
{
        // same code as last array example
}
```

Function Call

- main() function is the first function, which is called by your program when it's started to run

- some functions need include file to call them
  - *scanf(), printf() are functions that we already called*

- A function needs declared if it's called before function codes
  - *To* ***declare*** *a function prototype simply state the type, the name and list of the arguments. Stop with ;*
    e.g. double findaverage(double a, double b) ;

- A function can be called inside a function
- *Function can use reference variable, which can change value inside function, with & sign (not often in USACO)*

Problem Solving

Calculate the sum of the sum of 2 adjacent number square from 5 list of integer

Input: 1 2 3 4 5

Output: 84

1. Solving:

```
/*
 * lesson6.cpp
 *
 *  Created on: Nov. 3, 2011
 *      Author: PatrickHo
 */

#include <stdio.h>

unsigned long long SumOf2Int(int a, int b); // declare

int main()
{
    int iInteger[5];

    for (int i=0; i<5; i++)
        scanf("%d", &iInteger[i]);

    unsigned long long sum = 0;

    for (int i=0; i<4; i++)
        sum += SumOf2Int(iInteger[i], iInteger[i+1]);

    printf("%llu\n",sum);
    return 0;
}

// function of SumOf2Int
unsigned long long SumOf2Int(int a, int b)
{
    unsigned long long sum;

    unsigned long long ia = a;
    unsigned long long ib = b;
    sum = ia*ia + ib*ib;
    //printf("sum=%llu\n",sum);
    return sum;
}
```

General File Input/Output Functions

- General File Open Function: fopen

  FILE *fopen(const char *filename, const char *mode);
  filename:
  absolute path or relative current directory(folder)
  mode:
  r - open for reading
  w - open for writing (file need not exist)
  a - open for appending (file need not exist)

- Usage Model:
  - define a FILE pointer
  - call fopen
  - call fscanf or fprintf for input or output
  - call fclose

Functions Review (used in homework)

scanf, fscanf, sscanf: input data to variable in format

printf, fprintf, sprintf: output data from variable in format

getchar, fgetc: input one character

gets,fgets: input whole line string(char array)

isdigit,isupper,islower: check char case

tolower,toupper: char case transfer

strlen: get the length of char array string

strcmp: compare 2 char array string same or not

strcpy: copy one char array contents to other

strdup: duplicate a char array with new memory & contents

memset: reset a block value (for 0/-1only)

Recursive Function

- It is the function that has a call to itself
- It uses memory stack to copy all local variables, be careful using in large data cases
- It needs a return to terminate dead loop

- It makes a complex problem to sub simple problem (divide and conquer)

- To use recursive function
    a. find recursive formula
    b. find terminate condition

Recursive Samples

- Typical problem:
  factorial of a given number n
- Design
  - manually list small number case
    0! = 1
    1! = 1 = 1*0!
    2! = 2*1 = 2*1*0! = 2*1!
    3! = 3*2*1 = 3*2!
  - derive main problem
    fact(n) = n   *   fact(n–1)

- Coding
- Test

Recursive vs Iteration

- Recursive is easy and efficient in coding
- Iteration use less memory

e.g.

```
long long fact(int n)
{
    long long ret = 1;
    while(n>0)
    {
        ret = ret * n;
        n--;
    }
    return ret;
}
```

Problem Solving

Ladder Walk

One ladder has n(n<=20) steps. One walk up can go either one step a time, or two steps a time. Calculate how many different ways to go to top.

Solution A:

```
a.  Manual list
1 step ladder: 1 way (1/time)
2 steps ladder: 2 ways
(1: 1/time; or 1: 2/time)
3 steps ladder:
(1: 1/time; or 1:
    1/time+2/time; or
 1: 2/time + 1/time;)
…
N steps ladder: f(N-1) + f(N-
    2)
```

Sorting

- an algorithm that puts a list of elements in a certain order

- efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly

- popular sorting algorithms

Quick Sort:	O(nlogn) <call big O, growth rate>
Bubble Sort:	O(n^2)
Merge Sort:	O(nlogn)
Heap Sort:	O(nlogn)

STL sort Function

- it’s included in <algorithm>
- Short code and easy to use (no cast needed in compare function as in qsort)
- Usage:

```
template <class RandomAccessIterator>
        void sort ( RandomAccessIterator first, RandomAccessIterator last );
template <class RandomAccessIterator, class Compare>
        void sort ( RandomAccessIterator first, RandomAccessIterator last, Compare comp );
```

- Compare function is bool type and use reference &

```
bool myComp(const double &a, const double &b) {
      if (a>b) return true;     // sort descending
      else return false;
}
```

- Example:
  - int array
    `sort(arr,arr+5); // ascending`
  - double array
    `sort(darr, darr+5, myComp);`
  - sort vector
    `sort(a.begin(), a.end()); // use < operator`

Class in C/C++

- A structured (record) type that combines a set of different types objects into a single object, and functions

- Basic usage is enough for contest

- A class is defined by

```
class class_name  {
scope:                          // just public
    type object_name1;
    type object_name2;
     ...
    type function_name1();
    type function_name2(parameters)
    {
            ….
     }
 };
```

Access Class Object Member

• Use **.** sign to access a none-pointer class variable object member
• Use -> (2 operators together) to access a pointer class variable object member

•Example

```
class student
{
public:
    string cName;
    int iId;
    int iMathScore;
};
```

```
…
student studentA;   // declare
studentA.cName = “Tom”;
studentA.iId = 1;
studentA.iMathScore = 100;
…

student classAStudent[100];   // declare
int iTopScore = classAStudent[5].iMathScore;
if (classAStudent[5].iId == 1)
    printf(“Name=%s\n”, classAStudent[5].cName.c_str()

strudent *pOneStudent = classAStudent + 5; // use pointer
 printf(“Name:%10s ID:%3d\n”,
            pOneStudent->cName.c_str(),
            pOneStudent->iId);
```

Class in C/C++

- Example:
  - •define

```
class student  {
public:
  string sName;
     int  iMathScore;
     int  iArtScore;
     int GetAllScore() { return iMathScore + iArtScore;}
};
```

  - •declare

```
student A;
```

  - •set

```
A.sName= “Patrick” ;
```

  - •call function

```
printf( “total score of A is %d\n” , A.GetAllScore());
```

Operator Overloading C/C++

- Redefine or overload the function of most built-in operators
  - change the behavior of +, -, *, /, +, =, <, > etc.
  - in contest, this can be used as default compare function for any class data type (e.g. in sort, priority_queue, etc)

- Example:

```
class myMan
{
public:
        string name;
        int age;

        // change < operator
        bool operator< (const myMan& a) const
        {
              return age > a.age;  // make sort from old to young
        }
};
```

Problem (Use class)

Sort people list based on age from young to old

Input: data.in file:1st is the total people number N (N<5000)

2nd to N+1 line has person first name(string length < 20) and his age(integer)

e.g.

```
3
Tim 45
Jim 20
Patrick 50
```

Output: list list string from young to old

e.g.

```
Jim Tim Patrick
```

Structures in C/C++ (option)

- A structured (record) type that combines a set of different types objects into a single object.

- A struct is defined by

```
struct struct_tag  {
     type object_name1;
     type object_name2;
      ...
 }
```

  (Note: struct_tag is optional)

- Example

```
struct student  {
       char cName[64];
       unsigned int iId;
       int iMathScore;
}
```

Special Data Type of Struct

- Using typedef to make a struct as a special data type named by programmer

- Advantages:
  this new special named data type can be used same as simple data type(int, char, etc) everywhere

- Syntax:

```
typedef struct struct_tag  {
    type object_name1;
    type object_name2;
     ...
} new_data_type_name;
```

Struct_tag is optional

- **Example**

```
typedef struct  {
        string cName;
        unsigned int iId;
        int iMathScore;
} MyStudent;
```

Define Variable of Struct

- struct data type can be used to define variable, variable array or pointer varibles

- Example

```
MyStudent studentA, classAStudent[100];
MyStudent *pOneStudent = 0;
MyStudent AAA = { “Patrick” , 10, 100}; // with init value
```

Access struct Object Member

- Use **.** sign to access a none-pointer struct variable object member
- Use -> (2 operators together) to access a pointer struct variable object member

- Example

```
studentA.cName = “Tom”;
studentA.iId = 1;
studentA.iMathScore = 100;

int iTopScore = classAStudent[5].iMathScore;
if (classAStudent[5].iId == 1)
    printf( “Name=%s\n”, classAStudent[5].cName.c_str());

 pOneStudent = classAStudent + 5; // use pointer
 printf( “Name:%10s ID:%3d\n”, pOneStudent->cName.c_str(),
                pOneStudent->iId);
```

Problem (Use struct)

Sort people list based on age from young to old

Input: 1st is the total people number N (N<5000)
2nd to N+1 line has person first name(string length < 20) and his age(integer)
e.g.
3
Tim 45
Jim 20
Patrick 50

Output: list list string from young to old
e.g.
Jim Tim Patrick

Global vs Local Variable

- Example:

```
    …
  bool bVisit[100][100];  // init with false (0)
  int iTotle;
  …
  void BT(int m, int &it) {
     int iT = iTotal+m;
     it++;
     if (iT == 0) return;
     BT(iT, it);
}
…
int  main(int argc, char ** argv) {
    …
    if (argc == 3) {
        int im=100, it=iTotal+1;
        BT(im, it);
        cout << im << it << endl;
   }
}
```

Global vs Local Variable

- Global Variable:
  - Declared outside of function
  - Initialized
  - Used in all functions after it
- Local Variable:
  - Declared inside of function
  - Valid in scope only {…}
  - Declare it as close as to using place
  - Function argument pass the value of caller value, not caller variable(except using reference &, or pointer *)

Formula of Summation

- $1 + 2 + 3 + \ldots + n = n(n+1)/2$
- $1 + 3 + 5 + \ldots + 2n-1 = n^2$
- $2 + 4 + 6 + \ldots + 2n = n(n+1)$
- $1^2 + 2^2 + 3^2 + \ldots + n^2 = n(n+1)(2n + 1)/6$
- $1^3 + 2^3 + 3^3 + \ldots + n^3 = (n(n+1)/2)^2$

Note: if not sure, use 1 to 3 number and calculate result manually.