MAHIDOL UNIVERSITY

Wisdom of the Land

[SCPY204]

Computer Programing

for Physicists

Class 03: 1 Feb 2018

Content: function and recursion, array

Instructor: Puwis Amatyakul

JAN
31
2018
น่าน
Cr. Nattapong Kamyan

Today's Goals

Part I: Review

Part II: Homework Q&A

Part III: Function and recursion

Part IV: Array

Review

1. Steps in Programming

Problem analysis / Planning and design / Coding / Testing & debugging / Documentation

2. Programming in C

- ✓ C – Basic Syntax : semicolon, comment, whitespace
- ✓ C – Data Types : char, int, float, double, char, void
- ✓ C – Variable : definition, declaration
- ✓ C – Operators : arithmetic, relational, logical
- ✓ C – Decision Making (if … else)
- ✓ C – Loop (while, for)

Homework

Question and Answer

Today's Goals

Part I: Review

Part II: Homework Q&A

Part III: Function (and recursion)

Part IV: Array

C - FUNCTION

- ✓ A **function** is a group of statements that together perform a task.
- ✓ Every **C program** has at least one **function**, which is main(), and all the most trivial programs can define additional **functions**.
- ✓ You can divide up your code into separate **functions**.

Function

```
return_type function_name(  parameter list  ) {
       body of the function
}
```

C Basic: Function

1. Defining a function

```
return_type function_name(  parameter list  ) {
      body of the function
}
```

• **Return Type** – A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

• **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.

• **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

• **Function Body** – The function body contains a collection of statements that define what the function does.

C Basic: Function

Example

```
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

C Basic: Function

2. Calling a function

To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the **required parameters along with the function** name, and **if the function returns a value, then you can store the returned value**.

Example

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

C Basic: Function

Exercise 1

Remember a **factorial** problem in the previous homework?

DO: Make a function named **factorial()** and try calling it.

L03_ex_1.c

C Basic: Function

3. Function Arguments

S.N.	Call Type & Description
1	**Call by value** This method copies the actual value of an argument into the formal parameter of the function. **In this case, changes made to the parameter inside the function have no effect on the argument.**
2	**Call by reference** This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. **This means that changes made to the parameter affect the argument.**

C Basic: Function

Example

Call by value

```
void swap1(int x, int y) {

   int temp;

   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put temp into y */

   return;
}
```

Call by reference

```
void swap2(int *x, int *y) {

   int temp;
   temp = *x;     /* save the value at address x */
   *x = *y;       /* put y into x */
   *y = temp;     /* put temp into y */

   return;
}
```

C Basic: Function

Example

```c
#include <stdio.h>

/* function declaration */
void swap1(int x, int y);
void swap2(int *x, int *y);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   swap1(a, b);

   printf("After swap1, value of a : %d\n", a );
   printf("After swap1, value of b : %d\n", b );

   swap2(&a, &b);

   printf("After swap2, value of a : %d\n", a );
   printf("After swap2, value of b : %d\n", b );


   return 0;
}
```

```c
void swap2(int x, int y) {

   int temp;

   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put temp into y */

   return;
}
```

```c
void swap2(int *x, int *y) {

   int temp;
   temp = *x;     /* save the value at address x */
   *x = *y;       /* put y into x */
   *y = temp;     /* put temp into y */

   return;
}
```

C Basic: Function Recursion

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
   recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

C Basic: Function Recursion

Example: factorial in recursion form

```
#include <stdio.h>

int factorial(unsigned int i) {

   if(i <= 1) {
      return 1;
   }
   return i * factorial(i - 1);
}

int  main() {
   int i = 15;
   printf("Factorial of %d is %d\n", i, factorial(i));
   return 0;
}
```

C Basic: Function Recursion

Exercise 2

Remember Fibonacci series problem in the previous homework?

DO: Make a recursive function named **fibonacci()** and try calling it for the first 10 terms.

L03_ex_2.c

Today's Goals

Part I: Review

Part II: Homework Q&A

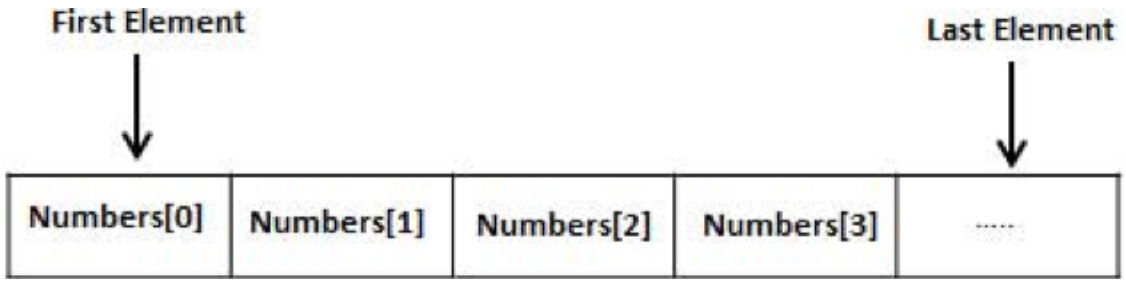
Part III: Function (and recursion)

Part IV: Array

C Basic: Array

Arrays is a kind of data structure that can **store a fixed-size sequential collection of elements of the same type**.

An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.



All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

C Basic: Array

1. Declaring Arrays

```
type arrayName [ arraySize ];
```

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type.

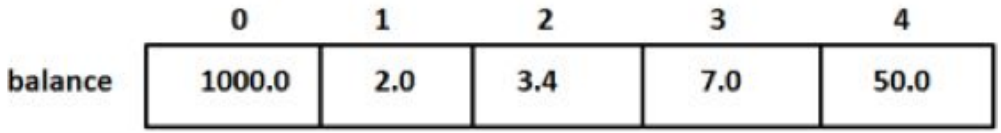
```
Ex: double balance[10];
```

2. Initializing Arrays

```
Way I:   double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Way II:  double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Way III: balance[4] = 50.0; /* so on */
```



C Basic: Array

3. Accessing Array Elements

```
double salary;
salary = balance[9];
```

Example

```c
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

C Basic: Array

S.N.	Concept & Description
1	**Multi-dimensional arrays** C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
2	**Passing arrays to functions** You can pass to the function a pointer to an array by specifying the array's name without an index.
3	**Return array from a function** C allows a function to return an array.

Multidimensional arrays

C programming language allows multidimensional arrays.

```
type name[size1][size2]...[sizeN];
```

Ex: `int threedim[5][10][4];`

Example: Two-dimensional array

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

C Basic: Array

Example: Two-dimensional array

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Example: Two-dimensional array

Initializing Two-Dimensional Arrays

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

or

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

```
int val = a[2][3];
```

C Basic: Array

Example: Two-dimensional array

```
#include <stdio.h>

int main () {

   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ ) {

      for ( j = 0; j < 2; j++ ) {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }

   return 0;
}
```

Passing Arrays as Function Arguments

If you want to pass an array as an argument in a function, you would have to declare a formal parameter in one of following three ways and all three declaration methods.

```
void myFunction(int *param) {
   .
   .
   .
}
```

```
void myFunction(int param[10]) {
   .
   .
   .
}
```

```
void myFunction(int param[]) {
   .
   .
   .
}
```

C Basic: Array

Passing Arrays as Function Argument

Example

```c
#include <stdio.h>

/* function declaration */
double getAverage(int arr[], int size);

int main () {

   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value */
   printf( "Average value is: %f ", avg );

   return 0;
}
```

```c
double getAverage(int arr[], int size) {

   int i;
   double avg;
   double sum = 0;

   for (i = 0; i < size; ++i) {
      sum += arr[i];
   }

   avg = sum / size;

   return avg;
}
```

C Basic: Array

Return array from function

Example

```c
#include <stdio.h>

/* function to generate and return random numbers */
int * getRandom( ) {

   static int  r[10];
   int i;

   /* set the seed */
   srand( (unsigned)time( NULL ) );

   for ( i = 0; i < 10; ++i) {
      r[i] = rand();
      printf( "r[%d] = %d\n", i, r[i]);
   }

   return r;
}

/* main function to call above defined function */
int main () {

   /* a pointer to an int */
   int *p;
   int i;

   p = getRandom();

   for ( i = 0; i < 10; i++ ) {
      printf( "*(p + %d) : %d\n", i, *(p + i));
   }

   return 0;
}
```

C Basic: Array

Exercise 3

Create the grading system for a school teacher.

- Teacher will input each student score from keyboard.
- The grading criteria is simple:

  >=80 gets A, >= 70 gets B, >= 60 gets C, >= 50 gets D, else F

- The result will be printed as:

```
Student 1: score = 85.5, gets A
Student 1: score = 65.5, gets C
…
```

for 10 students.

L03_ex_4.c

C Basic: Array

Exercise 4 and a homework

Solve **Ax** = **b** using Gaussian elimination

for 3 x 3 system

L03_hw_1.c

JAN
31
2018
น่าน
Cr. Nattapong Kamyan