Advance C / C++

FRA 142 Computer Programming for Robotics and Automation Engineering II

Suriya Natsupakpong, PhD

Institute of Field Robotics (FIBO)

King Mongkut's University of Technology Thonburi (KMUTT)

Cooking

A recipe is also a program, and everyone with some cooking experience can agree on the following:

1. It is usually easier to follow a recipe than to create one.
2. There are good recipes and there are bad recipes.
3. Some recipes are easy to follow and some are not easy to follow.
4. Some recipes produce reliable results and some do not.
5. You must have some knowledge of how to use cooking tools to follow a recipe to completion.
6. To create good new recipes, you must have a lot of knowledge and a good understanding of cooking.

Function

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

- Return Type: A function may return a value or no returning a value (void).
- Function Name: A name of the function.
- Parameter List: A placeholder to pass a value into function.
  - Call by Value
  - Call by Pointer
  - Call by Reference
- Function Body: A collection of statement define what the function does.

Example Function

```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;
}
```

```c
/* 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;
}
```

Function : 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.

```
/* function definition to swap the values */
void swap(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;
}
```

```
#include <stdio.h>

/* function declaration */
void swap(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 );

   /* calling a function to swap the values */
   swap(a, b);

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

   return 0;
}
```

Function : Call by Pointer

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.

```
/* function definition to swap the values */
void swap(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;
}
```

```
#include <stdio.h>

/* function declaration */
void swap(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 );

   /* calling a function to swap the values.
    * &a indicates pointer to a ie. address of variable a and
    * &b indicates pointer to b ie. address of variable b.  */
   swap(&a, &b);

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

   return 0;
}
```

Function : Call by Reference

This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

```
// function definition to swap the values.
void swap(int &x, int &y)
{
   int temp;

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

   return;
}
```

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

// function declaration
void swap(int &x, int &y);

int main ()
{
   // local variable declaration:
   int a = 100;
   int b = 200;

   cout <<"Before swap, value of a :" << a << endl;
   cout <<"Before swap, value of b :" << b << endl;

   /* calling a function to swap the values using variable
reference.*/
   swap(a, b);

   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;

   return 0;
}
```

Variable Scope Rules

- Inside a function or a block which is called **local** variables.

  Local variables that are declared inside a function or block. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

- Outside of all functions which is called **global** variables.

  Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

- In the definition of function parameters which is called **formal** parameters.

  Function parameters, formal parameters, are treated as local variables with-in that function and they will take preference over the global variables.

C++ Basic Syntax

- Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.
- Class - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- Methods - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- Instant Variables - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

C and C++ Program Structure

```c
#include <stdio.h>

/* main() is where program execution begins. */

int main()
{
   printf("Hello World"); /* prints Hello World */
   return 0;
}
```

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

// main() is where program execution begins.

int main()
{
   cout <<"Hello World";// prints Hello World
   return 0;
}
```

Rectangle in C

```
#include <stdio.h>

int main()
{
   double length;
   double width;
   double area;
   double perimeter;
   printf("Program to compute and output the perimeter and ");
   printf("area of a rectangle.\n");
   printf("Please enter length :");
   scanf("%f", &length);
   printf("Please enter width :");
   scanf("%f", &width);
   perimeter = 2 * (length + width);
   area = length * width;
   printf("Length = %f\n", length);
   printf("Width = %f\n", width);
   printf("Perimeter = %f\n", perimeter);
   printf("Area = %f\n", area);
   return 0;
}
```

Rectangle in C++

```
#include <iostream>
using namespace std;
int main()
{
   double length;
   double width;
   double area;
   double perimeter;
   cout << "Program to compute and output the perimeter and "
        << "area of a rectangle." << endl;
   cout << "Please enter length :";
   cin >> length;
   cout << "Please enter width :";
   cin >> width;
   perimeter = 2 * (length + width);
   area = length * width;
   cout << "Length = " << length << endl;
   cout << "Width =  " << width << endl;
   cout << "Perimeter = " << perimeter << endl;
   cout << "Area = " << area << endl;
   return 0;
}
```

Variable declarations

Input statements.

Assignment statement.

Output statements.

Preprocessor Directives

```
#include <headerFileName>
```

Standard C++ Header File Name	ANSI/ISO Standard C++ Header File Name
assert.h	cassert
ctype.h	cctype
float.h	cfloat
fstream.h	fstream
iomanip.h	iomanip
iostream.h	iostream
limits.h	climits
math.h	cmath
stdlib.h	cstdlib
string.h	cstring

Order of Precedence

When more than one arithmetic operator is used in an expression, C++ uses the operator precedence rules to evaluate the expression.

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the inner most pair is evaluated first.
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+, -	Addition, Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Example: 3 * 7 – 6 + 2 * 5 / 4 + 6

Order of Precedence

Including the arithmetic, relational, and logical operators

Operators	Precedence
!, +, – (unary operators)	first
*, /, %	second
+, –	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Example: 5 + 3 <= 9 && 2 > 3

Mix Expressions

```
// This program illustrates how mixed expressions are evaluated.
#include <iostream>
using namespace std;
int main()
{
    cout << "3 / 2 + 5.5 = " << 3 / 2 + 5.5 << endl;
    cout << "15.6 / 2 + 5 = " << 15.6 / 2 + 5 << endl;
    cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
    cout << "4 * 3 + 7 / 5 - 25.5 = "
         << 4 * 3 + 7 / 5 - 25.5
         << endl;
    return 0;
}
```

```
Sample Run:
3 / 2 + 5.5 = 6.5
15.6 / 2 + 5 = 12.8
4 + 5 / 2.0 = 6.5
4 * 3 + 7 / 5 - 25.5 = -12.5
```

C++ Input : cin >>

```
cin >> variable >> variable ...;
```

```
double payRate, hoursWorked;
cin >> payRate >> hoursWorked;
```

is the same as

```
cin >> payRate;
cin >> hoursWorked;
```

and the input is

```
15.50 48.30
```

or:

```
15.50   48.30
```

or:

```
15.50
48.30
```

the preceding input statement would store 15.50 in payRate and 48.30 in hoursWorked.

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch, ch1, ch2;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> z >> ch >> a;`	`36.78B34`	`z = 36.78, ch = 'B', a = 34`
2	`cin >> z >> ch >> a;`	`36.78` `B34`	`z = 36.78, ch = 'B', a = 34`
3	`cin >> a >> b >> z;`	`11 34`	`a = 11, b = 34,` computer waits for the next number
4	`cin >> a >> z;`	`78.49`	`a = 78, z = 0.49`
5	`cin >> ch >> a;`	`256`	`ch = '2', a = 56`
6	`cin >> a >> ch;`	`256`	`a = 256,` computer waits for the input value for `ch`
7	`cin >> ch1 >> ch2;`	`A B`	`ch1 = 'A', ch2 = 'B'`

Functions in cin

get Function

```
cin.get(varChar);
```

The get function inputs the very next character, including whitespace characters, from the input stream and stores it in the memory location indicated by its argument.

ignore Function

```
cin.ignore(intExp, chExp);
```

The ignore function ignors the first intExp characters or all characters until the character chExp is found, whichever comes first.

putback Function

```
istreamVar.putback(ch);
```

The putback function puts the last character extracted from the input stream by the get function back into the input stream.

peek Function

```
ch = istreamVar.peek();
```

The peek function returns the next character from the input stream but does not remove the character from that stream.

Clear Function

```
istreamVar.clear();
```

The clear function restore the input stream to a working state.

```
//Functions peek and putback
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout << "Line 1: Enter a string: ";
    cin.get(ch);
    cout << endl;
    cout << "Line 4: After first cin.get(ch); "
         << "ch = " << ch << endl;
    cin.get(ch);
    cout << "Line 6: After second cin.get(ch); "
         << "ch = " << ch << endl;
    cin.putback(ch);
    cin.get(ch);
    cout << "Line 9: After putback and then "
         << "cin.get(ch); ch = " << ch << endl;
    ch = cin.peek();
    cout << "Line 11: After cin.peek(); ch = "
         << ch << endl;
    cin.get(ch);
    cout << "Line 13: After cin.get(ch); ch = "
         << ch << endl;
    return 0;
}
```

```
Sample Run: In this sample run, the user input is in white.
Line 1: Enter a string: abcd
Line 4: After first cin.get(ch); ch = a
Line 6: After second cin.get(ch); ch = b
Line 9: After putback and then cin.get(ch); ch = b
Line 11: After cin.peek(); ch = c
Line 13: After cin.get(ch); ch = c
```

C++ Output : cout <<

```
cout << expression or manipulator << expression or manipulator ...;
```

endl : the insertion point is move to the beginning of the next line.

\n	Newline
\t	Tab
\b	Backspace
\r	Return
\\	Backslash
\’	Single quotation
\”	Double quotation

```
How to print this message :
The tab character is represented as '\t'
```

cout and Output Format

```
#include <iomanip>
```

Manipulation Functions

```
cout << setprecision(n);

cout << fixed;
cout.unsetf(ios::fixed);
cout << scientific;
cout.unsetf(ios::scientific);

cout << showpoint;

cout << setw(n);

cout << setfill(ch);

cout << left;
cout.unsetf(ios::left);
cout << right;
```

```
//Example: left justification
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int x = 15;
    float y = 76.34;

    cout << left;
    cout << "12345678901234567890" << endl;
    cout << setw(5) << x << setw(7) << y
         << setw(8) << "Warm" << endl;
    cout << setfill('*');
    cout << setw(5) << x << setw(7) << y << setw(8) << "Warm" << endl;
    cout << setw(5) << x << setw(7) << setfill('#') << y << setw(8) << "Warm" << endl;
    cout << setw(5) << setfill('@') << x
         << setw(7) << setfill('#') << y
         << setw(8) << setfill('^') << "Warm"
         << endl;
    cout << right;
    cout << setfill(' ');
    cout << setw(5) << x << setw(7) << y << setw(8) << "Warm" << endl;
    return 0;
}
```

```
Sample Run:
12345678901234567890
15   76.34  Warm
15***76.34**Warm****
15***76.34##Warm####
15@@@76.34##Warm^^^^
   15  76.34    Warm
```

File Input / Output

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char ch;
   FILE *source, *target;
   //Open the files
   source = fopen("prog.dat", "r");

   if( source == NULL ) {
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }

   target = fopen(("prog.out", "w");

   while( ( ch = fgetc(source) ) != EOF ) {
      fputc(ch, target);
   }

   //Close files
   fclose(source);
   fclose(target);

   return 0;
}
```

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

int main()
{
    //Declare file stream variables such as the following
    ifstream inData;
    ofstream outData;

    //Open the files
    inData.open("prog.dat"); //open the input file
    if (!inData) {
        cout << "Cannot open input file." << endl;
        return 1;
    }
    outData.open("prog.out"); //open the output file

    //Code for data manipulation
    while(!inData.eof()) {
        // do something...
    }

    //Close files
    inData.close();
    outData.close();

    return 0;
}
```

Why array?

- Find minimum of 20 numbers :

123 23 12 32 43 11 343 237 675 31

526 98 79 90 25 15 67 876 423 123

```
int num1, num2, num3, num4, num5, num6, num7, num8, num9, num10,
    num11, num12, num13, num14, num15, num16, num17, num18, num19, num20;
```

```
int num[20];
```

Arrays

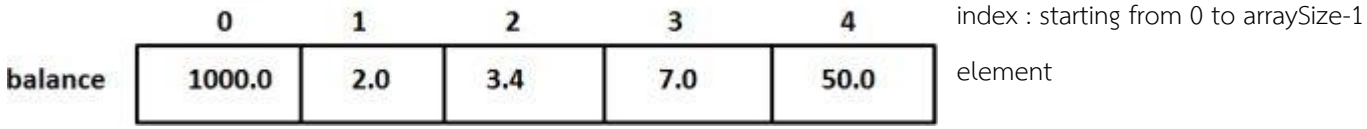
- Declaring Arrays

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

```
double balance[10];
```

- Initializing Arrays

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



- Accessing Array Elements

```
double val = balance[3];
balnace[1] = 3.5;
```

Two-dimensional Arrays

	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]

- Declaring Two-dimensional Arrays

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

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

- 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 */
};
```

- Accessing Two-dimensional Array Elements

```
int val = a[2][3];
a[1][2] = 3;
```

Passing Arrays as Function Arguments

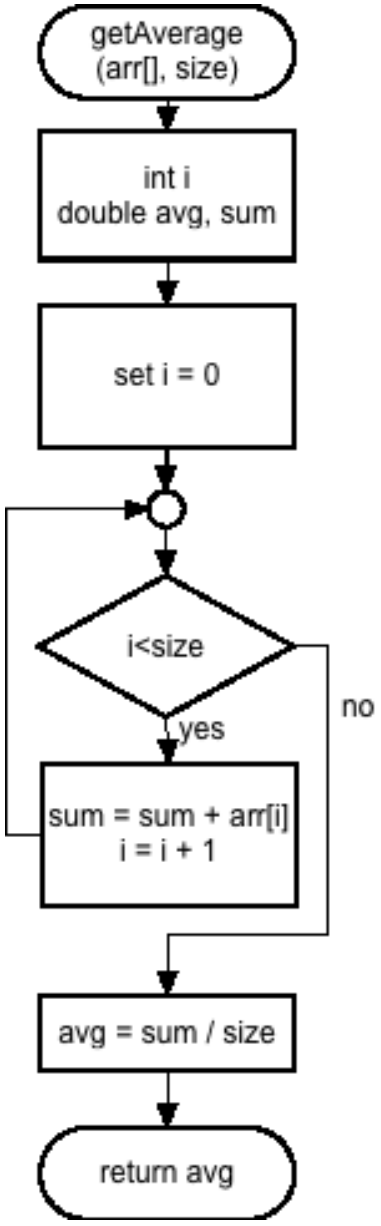
```
void myFunction(int param[10])
```

```
void myFunction(int param[])
```

```
void myFunction(int *param)
```

```
double getAverage(int arr[], int size)
{
  int    i;
  double avg, sum;

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



Task 3-1

จงเขียนโปรแกรมรับจำนวนทศนิยม 7 ค่า แล้วเก็บใน array หนึ่งมิติที่มีขนาด 7 ตัว
แล้วส่งค่าไปยังฟังก์ชั่นหาค่าสูงสุด (getMax) ค่าต่ำสุด (getMin) และค่าเฉลี่ย (Mean)
ตัวอย่างผลการรัน

Enter 7 floating numbers (separated by spacebar): 1.2 1.1 1.5 1.4 1.7 1.6 1.3

The minimum number is 1.1

The maximum number is 1.7

The average is 1.4

What are Pointers?

A **pointer** is a variable whose value is the address of another variable

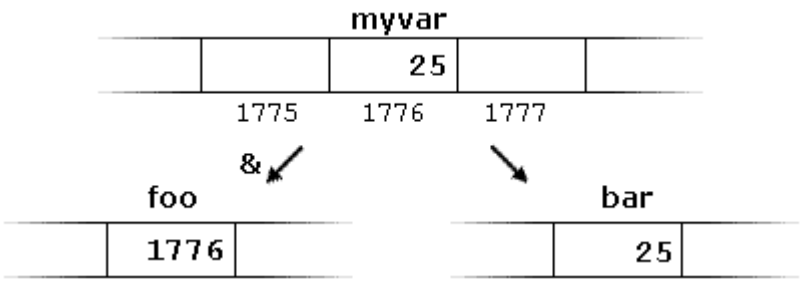
```
type *var-name;
```

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Pointer Operators

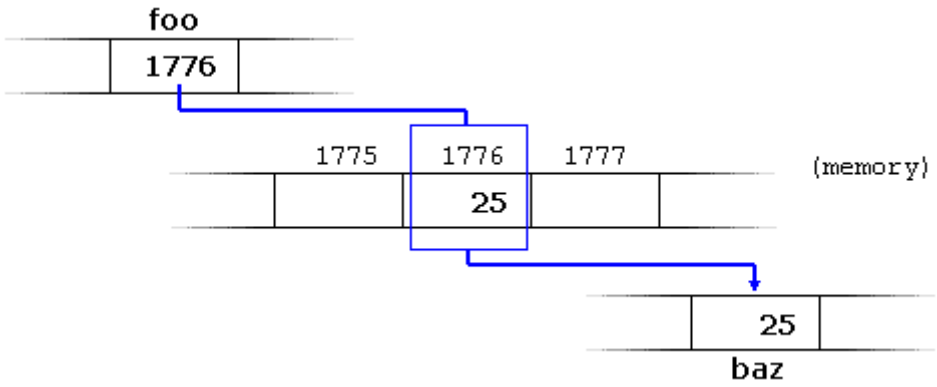
- Address-of operator (&)

```
myvar = 25;
foo = &myvar;
bar = myvar;
```



- Dereference operator (*)

```
baz = *foo;
```




```
baz = foo;   // baz equal to foo (1776)
baz = *foo;  // baz equal to value pointed to by foo (25)
```

How to use Pointers?

A **pointer** is a variable whose value is the address of another variable

```
#include <stdio.h>

int main ()
{
   int  var = 20;   /* actual variable declaration */
   int  *ip = NULL; /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

```
Address of var variable: 28ff18
Address stored in ip variable: 28ff18
Value of *ip variable: 20
```

Pointer Example

```
int  a = 5;
int  *ptr1 = NULL;
int  *ptr2 = NULL;
```

Address	Contents
0x8130	0x00000005
0x8134	0x00000000
0x8138	0x00000000

```
ptr1 = &a;
```

Address	Contents
0x8130	0x00000005
0x8134	0x00008130

```
*ptr1 = 8;
```

Address	Contents
0x8130	0x00000008
0x8134	0x00008130

```
ptr2 = ptr1;
```

Address	Contents
0x8130	0x00000008
0x8134	0x00008130
0x8138	0x00008130

```
What values does this program print?

int a = 5, b = 10;
int *p1, *p2;
p1 = &a;
p2 = &b;
*p1 = 10;
p1 = p2;
*p1 = 20;
printf("a = %d\n", a);
printf("b = %d\n", b);


int c[5] = {1,2,3,4,5};
int *p3;
int dave;
p3 = &c[1];
dave = *(p3+2);
printf("*p3 = %d\n", *p3);
printf("dave = %d\n", dave);
```

c/c++ array index starts at 0

Difference in Reference Variable and Pointer Variable

- References are generally implemented using pointers. A reference is same object, just with a different name and reference must refer to an object. Since references can’t be NULL, they are safer to use.
- A pointer can be re-assigned while reference cannot, and must be assigned at initialization only.
- Pointer can be assigned NULL directly, whereas reference cannot.
- Pointers can iterate over an array, we can use ++ to go to the next item that a pointer is pointing to.
- A pointer is a variable that holds a memory address. A reference has the same memory address as the item it references.
- A pointer to a class/struct uses ‘->'(arrow operator) to access it’s members whereas a reference uses a ‘.'(dot operator)
- A pointer needs to be dereferenced with * to access the memory location it points to, whereas a reference can be used directly.
- References are usually preferred over pointers whenever we don’t need “reseating”.
- **Use references when you can, and pointers when you have to**.

http://www.geeksforgeeks.org/passing-by-pointer-vs-passing-by-reference-in-c/

Difference in Reference Variable and Pointer Variable

```
// Differences between pointer and reference.
#include <iostream>
using namespace std;

struct demo {
    int a;
};

int main() {
    int x = 5, y = 6;
    demo d;

    int *p;
    p = &x;
    p = &y;       // 1. Pointer reintialization allowed
    int &r = x;
    // &r = y;    // 1. Compile Error
    r = y;        // 1. x value becomes 6

    p = NULL;
    // &r = NULL; // 2. Compile Error

    p++;          // 3. Points to next memory location
    r++;          // 3. x values becomes 7

    cout << &p << " " << &x << endl; // 4. Different address
    cout << &r << " " << &x << endl; // 4. Same address

    demo *q = &d;
    demo &qq = d;

    q->a = 8;
    // q.a = 8;    // 5. Compile Error
    qq.a = 8;
    // qq->a = 8;  // 5. Compile Error

    cout << p << endl;  // 6. Prints the address
    cout << r << endl;  // 6. Print the value of x
    return 0;
}
```

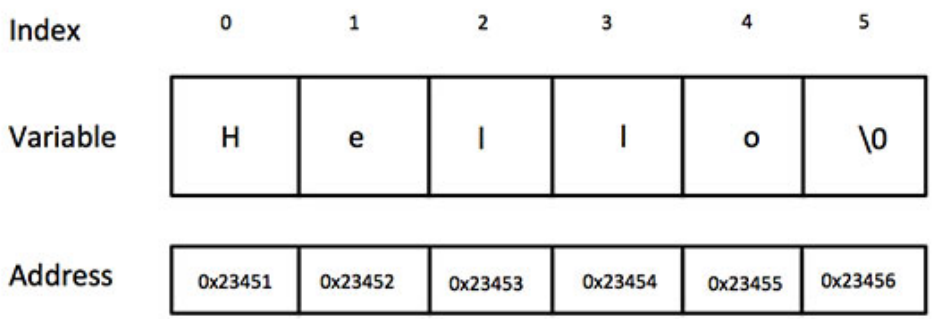
http://www.geeksforgeeks.org/passing-by-pointer-vs-passing-by-reference-in-c/

What are Strings?

The string in C programming language is actually a one-dimensional array of characters which is terminated by a **null** character '\0'.

```
char *var-name-ptr;
```

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
char greeting[] = "Hello";
```



Task 3-2

ให้ประกาศตัวแปร name ที่เป็น Array of Pointer to Char Array ที่เก็บข้อมูลดังต่อไปนี้

```
{"Bachelor in Robotics and Automation Engineering",
 "FRAB",
 "Institute of Field Robotics",
 "FIBO",
 "King Mongkut's University of Technology Thonburi",
 "KMUTT",
 "FB306"}
```

จากนั้นพิมพ์ข้อมูลบนหน้าจอ และตอบคำถามต่อไปนี้ว่ามีผลลัพธ์เป็นเท่าไร

```
name[0],  name[0]+12, name[1], (*name), (*name+2)+5, *(name+5)
```

Useful String Functions

#include <string.h>

char *strcpy(char *s1, const char *s2)

Copies string **s2** into array **s1**. The value of **s1** is returned.

char *strncpy(char *s1, const char *s2, size_t n)

Copies at most **n** characters of string **s2** into array **s1**. The value of **s1** is returned.

char *strcat(char *s1, const char *s2)

Appends string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.

char *strncat(char *s1, const char *s2, size_t n)

Appends at most **n** characters of string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.

Useful String Conversion Functions

#include <stdlib.h>

double atof(const char *nPtr)

Converts the string **nPtr** to **double**.

int atoi(const char *nPtr)

Converts the string **nPtr** to **int**.

long atol(const char *nPtr)

Converts the string **nPtr** to long **int**.

String in C and C++

```
C-strings  (#include <cstring>)
==============================

Declaring a C-string variable
-----------------------------
char str[10];

Initializing a C-string variable
--------------------------------
char str1[11] = "Call home!";
char str2[] = "Send money!";
char str3[] = {'O', 'K', '\0'};
Last line above has same effect as:
char str3[] = "OK";


Assigning to a C-string variable
--------------------------------
Can't do it, i.e., can't do this:
char str[10];
str = "Hello!";
```

```
C++ strings  (#include <string>)
================================

Declaring a C++ string object
-----------------------------
string str;

Initializing a C++ string object
--------------------------------
string str1("Call home!");
string str2 = "Send money!";
string str3("OK");


string str4(10, 'x');

Assigning to a C++ string object
--------------------------------
string str;
str = "Hello";
str = otherString;
```

http://cs.stmarys.ca/~porter/csc/ref/c_cpp_strings.html

String in C and C++

```
C-strings  (#include <cstring>)
==============================

Concatenating two C-strings
---------------------------
strcat(str1, str2);
strcpy(str, strcat(str1, str2));

Copying a C-string variable
---------------------------
char str[20];
strcpy(str, "Hello!");
strcpy(str, otherString);

Accessing a single character
----------------------------
str[index]
```

```
C++ strings  (#include <string>)
================================

Concatenating two C++ string objects
------------------------------------
str1 += str2;
str = str1 + str2;

Copying a C++ string object
---------------------------
string str;
str = "Hello";
str = otherString;

Accessing a single character
----------------------------
str[index]
str.at(index)
str(index, count)
```

http://cs.stmarys.ca/~porter/csc/ref/c_cpp_strings.html

String in C and C++

```
C-strings  (#include <cstring>)
==============================

Comparing two C-strings
-----------------------
if (strcmp(str1, str2) < 0)
    cout << "str1 comes 1st.";
if (strcmp(str1, str2) == 0)
    cout << "Equal strings.";
if (strcmp(str1, str2) > 0)
    cout << "str2 comes 1st.";

Finding the length of a C-string
--------------------------------
strlen(str)

Output of a C-string variable
-----------------------------
cout << str;
cout << setw(width) << str;
```

```
C++ strings  (#include <string>)
================================

Comparing two C++ string objects
--------------------------------
if (str1 < str2)
    cout << "str1 comes 1st.";
if (str1 == str2)
    cout << "Equal strings.";
if (str1 > str2)
    cout << "str2 comes 1st.";

Finding the length of a C++ string object
-----------------------------------------
str.length()

Output of a C++ string object
-----------------------------
cout << str;
cout << setw(width) << str;
```

http://cs.stmarys.ca/~porter/csc/ref/c_cpp_strings.html

Array of Pointers to Functions

```c
#include <stdio.h>

void fn1(int);
void fn2(int);
void fn3(int);

int main() {
   void(*f[3])(int) = { fn1, fn2,fn3 };
   int choice;
   printf("Enter a number between 0 and 2, 3 to end : ");
   scanf("%d", &choice);
   while(choice >= 0 && choice < 3) {
     (*f[choice])(choice);
     printf("Enter a number between 0 and 2, 3 to end : ");
     scanf("%d", &choice);
   }
   printf("Program execution completed.\n");
   return 0;
}
```

```c
void fn1(int a) {
   printf("You entered %d so fn1 was called\n\n", a);
}
void fn2(int b) {
   printf("You entered %d so fn2 was called\n\n", b);
}
void fn3(int b) {
   printf("You entered %d so fn3 was called\n\n", b);
}
```

Random Number Generator

```
#include <iostream>
#include <cstdlib>
#include <iomanip>
#include <ctime>

using namespace std;
int main()
{
    srand(time(NULL));
    cout << fixed << showpoint << setprecision(5);
    cout << "The value of RAND_MAX: " << RAND_MAX << endl;
    cout << "A random number: " << rand() << endl;
    cout << "A random number between 0 and 9: "
         << rand() % 10 << endl;
    cout << "A random number between 0 and 1: "
         << static_cast<double> (rand())
         / static_cast<double>(RAND_MAX)
         << endl;
    return 0;
}
```

Task 3-3 : Roman numbers

Roman numbers. Write a program that converts a positive integer into the Roman number system. The Roman number system has digits

I	1
V	5
X	10
L	50
C	100
D	500
M	1,000

Numbers are formed according to the following rules.

(1) Only numbers up to 3,999 are represented.

(2) As in the decimal system, the thousands, hundreds, tens, and ones are expressed separately.

(3) The numbers 1 to 9 are expressed as

I	1
II	2
III	3
IV	4
V	5
VI	6
VII	7
VIII	8
IX	9

As you can see, an I preceding a V or X is subtracted from the value, and you can never have more than three I's in a row.

(4) Tens and hundreds are done the same way, except that the let ters X, L, C and C, D, M are used instead of I, V, X, respectively.

Your program should take an input, such as 1978, and convert it to Roman numerals, MCMLXXVIII.