

Problem Solving With C++

Mohammad A. Rajabi

Dept. of Geomatics Eng.

University of Tehran

Cell: 0912 132 5823

Email: marajabi@ut.ac.ir

<http://www.marajabi.com>

Pointers

- Most of C++'s power is derived from pointers
- They allow C++ to support dynamic memory allocation

Assumption:

- characters are one byte in length
- integers are four bytes long
- floats are four bytes long
- doubles are eight bytes long

Pointers

- Pointer is a variable that contains a **memory address**.
- If **x** contains the address of **y**, then **x** is said to **point to y**
- **General Form of Pointer Declaration:**

```
type *var_name;
```

e.g.

```
int *p;
```

```
float *x;
```

```
double *s;
```

Pointer Operators

- There are two special operators that are used with pointers: `*` and `&`
- The `&` is a unary operator that returns the memory address of its operand.

```
int balance = 350;
```

```
int *balptr;
```

```
balptr = &balance;
```

- This address is the location of the variable in memory, it has nothing to do with the value of balance.

Pointer Operators

- The second operator `*` is the complement of `&`. It is a unary operator that returns the **value of variable located at address specified by its operand.**

```
int balance = 350;
```


```
int *balptr;
```

```
balptr = &balance;
```

```
int value;
```

```
value = *balptr;    //what does value contain?
```

Pointer Operators

&  address of

*  value at address

Pointer Operators

```
int main() {  
    int balance;  
    int *balptr;  
    int value;  
    balance = 3200;  
    balptr = &balance;  
    value = *balptr;  
    cout << "balance is: " << value << '\n';  
    cout << "Memory address where balance is stored is: ”  
        <<balptr<<endl;  
    return 0;  
}
```

Base Type

```
value = *balptr;
```

- The compiler transfers the proper number of bytes according to base type.

```
int *p;
```

```
double f;
```

```
// ...
```

```
p = &f; // ERROR
```

Assigning values through a pointer

```
int *p;
```

```
int x;
```

```
p=&x;
```

```
//Assign a value to the location pointed to by p
```

```
*p = 101;
```

```
cout<<x; //what is value of x?
```

Assigning values through a pointer

(*p)++; //increment value to the location pointed to by p

```
int main() {  
    int *p, num;  
    p = &num;  
    *p = 454;  
    cout << num << ' ';  
    ++ (*p);  
    cout << num << ' ';  
    (*p) - -;  
    cout << num << '\n';  
    return 0;  
} //Output?
```

Pointer Arithmetic

- There are only four arithmetic operators that can be used on pointers: $++$, $--$, $+$, $-$
- Let $p1$ be an integer pointer which contains the address 5000
- $p1++$; // now $p1$ will be 5004
- Each time $p1$ is incremented, it shall point to the next integer.
- $p1--$; will cause $p1$ to be 4996 if initially it was 5000.

Pointer Arithmetic

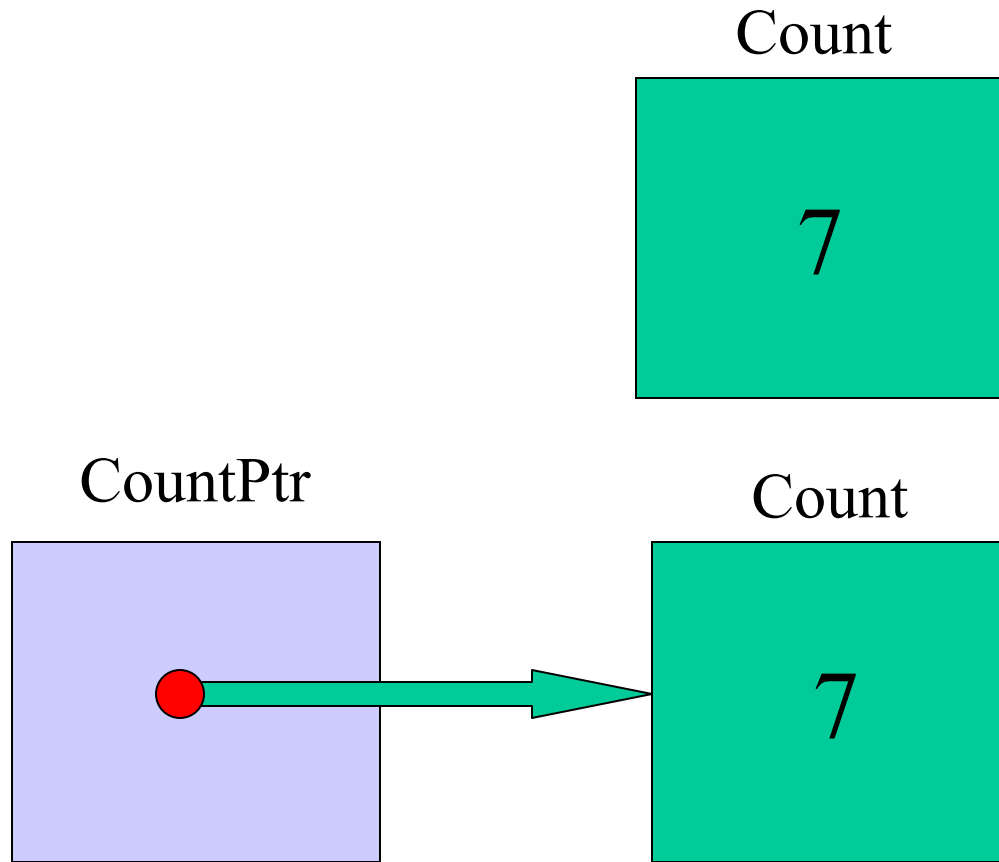
- Let `p1` be an **char** pointer which contains the address 5000
- `p1++;` // now `p1` will be 5001
- Each time `p1` is incremented, it shall point to the **next character**.
- `p1--;` will cause `p1` to be 4999 if initially it was 5000.
- Pointer of type other than `char` shall increase or decrease by length of base type.

Pointer Arithmetic

- You cannot add two pointers
- You can subtract two pointers (if they are of same base type).
- Other than addition or subtraction of **a pointer and an integer**, or the **subtraction of two pointers**, no other arithmetic operations can be performed on pointers.
- You cannot add or subtract float or double values.

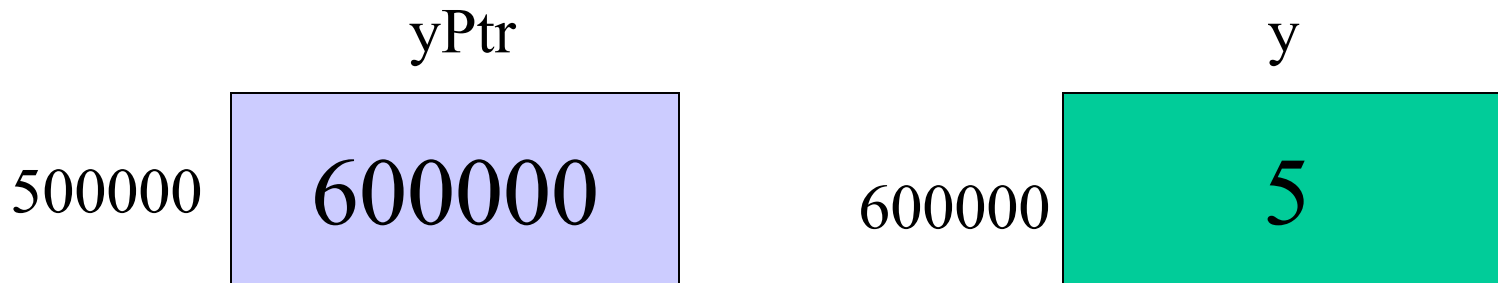
Directly and indirectly referencing a variable

- Directly and indirectly referencing a variable



Directly and indirectly referencing a variable

- **Pointers**



Pointers (ex-1)

```
main() {  
    int a;    // a is an integer  
    int *aPtr; // aPtr is a pointer to an integer  
    a = 7;  
    aPtr = &a; // aPtr set to address of a  
    cout << "The address of a is " << &a << endl  
        << "The value of aPtr is " << aPtr << endl << endl;  
    cout << "The value of a is " << a << endl  
        << "The value of *aPtr is " << *aPtr << endl <<  
        endl;  
    return 0;  
}
```

Pointers (ex-2)

```
int main()
{
    int *p;
    int x;
    p=&x;
    *p = 123;
    cout<<x<<endl;
    cout<<*p<<endl;
    cout<<p<<endl;
    cout<<&x<<endl;

    return 0;
}
```

Pointer Arithmetic (Example: pointer-add)

```
int main() {  
    int *i, a;  
    double *f, b;  
    int x;  
    a=10;  
    b=20;  
    i = &a;  
    f = &b;  
    for(x=0; x<10; x++)  
        cout << i+x << "    " << f+x << '\n';  
        cout <<*( i+x) << "    " <<*( f+x) << '\n';  
        cout << *i+x << "    " << *f+x << '\n';  
    return 0;  
}
```

Pointers and Arrays

- In C++, there is a close relationship between pointers and arrays.
- In fact arrays and pointers are frequently interchangeable!

Pointers and Arrays

```
int main() {  
    int *intPtr, i[10]={0,1,2,3,4,5,6,7,8,9};  
    char *charPtr;  
    char c[]="We are testing char pointers";  
    intPtr = i;  
    charPtr = c;  
    cout << i <<"    " << c << endl;  
    cout << intPtr << "    " << charPtr << '\n';  
    cout << *intPtr <<"    " << i[0] << '\n';  
    return 0;  
}
```

Indexing a Pointer

```
int main() {  
    char str[20] = "hello tom";  
    char *p;  
    int i;  
    p = str;  
    for(i=0; p[i]; i++) p[i] = toupper(p[i]);  
    cout << p << endl; // display the string  
    return 0;  
} //example
```

Are Pointers and Arrays Interchangeable?

```
int num[10];
```

```
int i;
```

```
for(i=0; i<10; i++) {
```

```
    *num = i; // this is OK
```

```
    num++; // ERROR -- cannot modify num
```

```
}
```

- **num is an array of integers**
- **array name without an index generates a pointer to the beginning of array**
- **num is a **constant** that points to the beginning of an array**

Are Pointers and Arrays Interchangeable?

```
int num[10];
```

```
int i;
```

```
for(i=0; i<10; i++) {
```

```
    *num = i; // this is OK
```

```
    num++; // ERROR -- cannot modify num
```

```
}
```

- ***(num+3) = 100; // This is OK because num is not changed**

String Constants

- When the compiler encounters a string constant, it stores it in the program string table and generates a **pointer** to the string.

```
int main() {  
    char *s;  
    s = "Pointers are fun to use.\n";  
    cout << s;  
    return 0;  
}
```

Tokenizing Example

- This program scans the input string, copying characters from the string into another array, called token, until a space is encountered. It prints the token and repeats the process until null at end of string is encountered.
- e.g “This is a test:

This

is

a

test

Tokenizing Example Using Arrays

```
int main() {  
    char str[80], token[80];  
    int i, j;  
    cout << "Enter a sentence: ";  
    gets(str);  
    // Read a token at a time from the string.  
    for(i=0; ; i++) {  
        /* Read characters until either a space or the null terminator is encountered. */  
        for(j=0; str[i]!=' ' && str[i]; j++, i++)  
            token[j] = str[i];  
        token[j] = '\0'; // null terminate the token  
        cout << token << '\n';  
        if(!str[i]) break;  
    }  
    return 0;  
}
```

Tokenizing Example Using Pointers

```
int main() {  
    char str[80], token[80];  
    char *p, *q;  
    cout << "Enter a sentence: ";  
    gets(str);  
    p = str;
```

Tokenizing Example Using Pointers

// Read a token at a time from the string.

while(*p) {

q = token; // set q pointing to start of token

/* Read characters until either a space or the null terminator is encountered. */

while(*p!=' ' && *p) {

***q = *p;**

q++; p++;

}

if(*p) p++; // advance past the space

***q = '\0'; // null terminate the token**

cout << token << '\n';

}

return 0;