261102
Computer Programming

Lecture 16: Pointers I

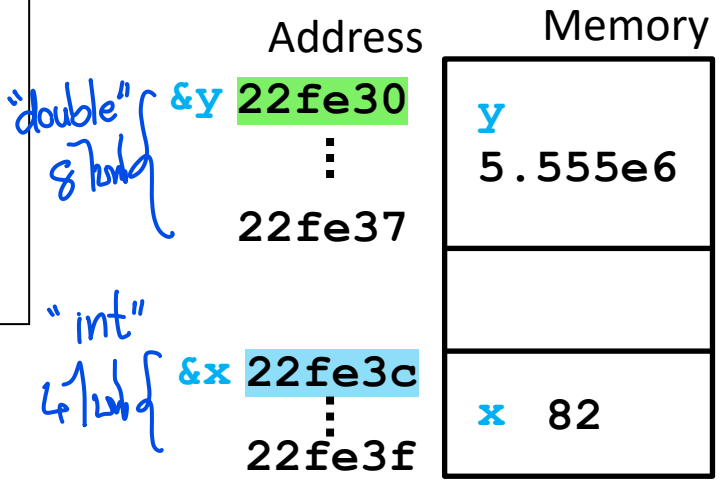
Memory Address

- **&** (address operator)
  - Returns memory address of its operand
  - &x = address of variable x

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 82;
7      double y = 55.55e5;
8      cout << "x = " << x << "\n";
9      cout << "&x = " << &x << "\n";
10     cout << "y = " << y << "\n";
11     cout << "&y = " << &y << "\n";
12     return 0;
13 }
```

Output

```
x = 82
&x = 0x22fe3c
y = 5.555e+006
&y = 0x22fe30
```



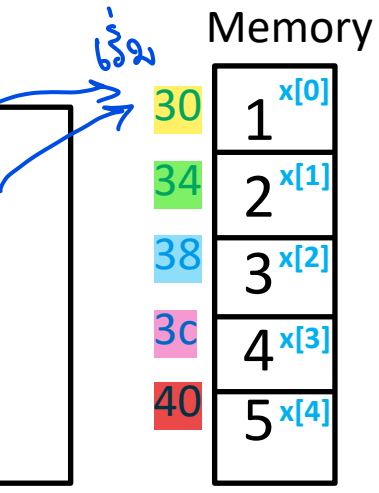
Memory Address

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

int main()
{
    int x[] = {1,2,3,4,5};
    cout << "&x = " << &x << "\n";
    cout << "&x[0] = " << &x[0] << "\n";
    cout << "&x[1] = " << &x[1] << "\n";
    cout << "&x[2] = " << &x[2] << "\n";
    cout << "&x[3] = " << &x[3] << "\n";
    cout << "&x[4] = " << &x[4] << "\n";
    return 0;
}
```

Output

```
&x = 0x22fe30
&x[0] = 0x22fe30
&x[1] = 0x22fe34
&x[2] = 0x22fe38
&x[3] = 0x22fe3c
&x[4] = 0x22fe40
```

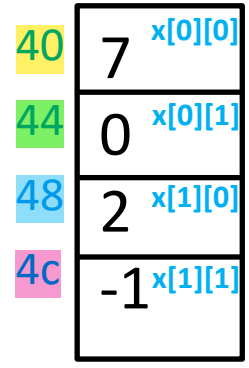



```cpp
#include <iostream>
using namespace std;

int main()
{
    int x[2][2] = {{7,0},{2,-1}};
    cout << "&x = " << &x << "\n";
    cout << "&x[0] = " << &x[0] << "\n";
    cout << "&x[1] = " << &x[1] << "\n";
    cout << "&x[0][0] = " << &x[0][0] << "\n";
    cout << "&x[0][1] = " << &x[0][1] << "\n";
    cout << "&x[1][0] = " << &x[1][0] << "\n";
    cout << "&x[1][1] = " << &x[1][1] << "\n";
    return 0;
}
```

Output

```
&x = 0x22fe40
&x[0] = 0x22fe40
&x[1] = 0x22fe48
&x[0][0] = 0x22fe40
&x[0][1] = 0x22fe44
&x[1][0] = 0x22fe48
&x[1][1] = 0x22fe4c
```

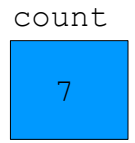


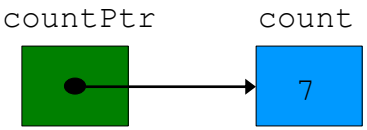
Pointer

"เก็บเลขaddress" "*"

↳ &□ , 0 , null

- Pointer variables
  - Contain memory addresses as values
  - Normally, variable contains specific value (direct reference)
  - Pointers contain address of variable that has specific value (indirect reference)





- Indirection
  - Referencing value through pointer
- Pointer declarations
  - * indicates variable is pointer

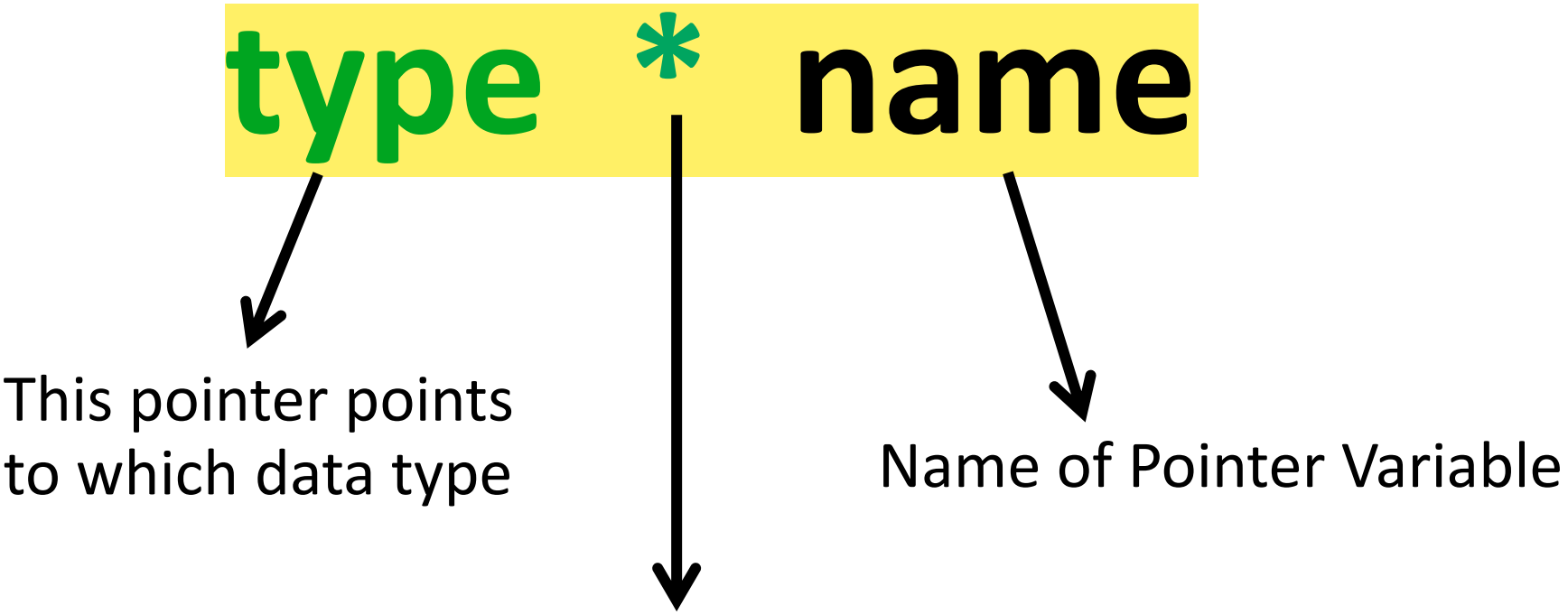
```
int *myPtr;
```

declares pointer to int, pointer of type int *

  - Can declare pointers to any data type
  - Multiple pointers require multiple asterisks

```
int *myPtr1, *myPtr2;
```

Pointer Declarations



Pointer

กำหนดค่า pointer

- Pointer assignment
  - Assign value of pointer to 0, **NULL**, or address

```
int x = 69;
int *myPtr1, *myPtr2;
myPtr1 = 0;
myPtr2 = &x;
```

  - Can not be assigned by normal number such as **int, long int, double** (except 0)

ยกเว้น

```
int *myPtr1 = 0x69;
```

error: invalid conversion from 'int' to 'int*'

  - 0 or **NULL** points to nothing

(พ้อยพอย่างเปล่า)

- Pointer initialization

```
int x = 69;
int *myPtr1 = 0, *myPtr2 = &x;
```

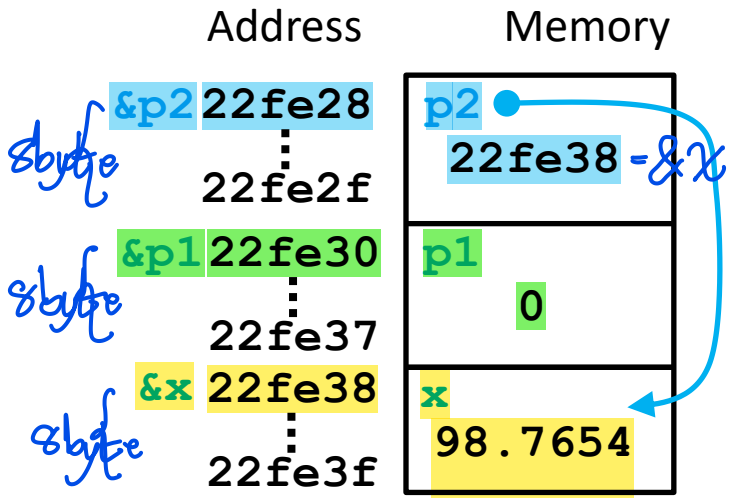
1 2

Pointer

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      double x = 98.7654;
7      int *p1 = NULL;
8      double *p2 = &x;
9
10     cout << "x = " << x << "\n";
11     cout << "&x = " << &x << "\n";
12     cout << "p1 = " << p1 << "\n";
13     cout << "p2 = " << p2 << "\n";
14     cout << "&p1 = " << &p1 << "\n";
15     cout << "&p2 = " << &p2 << "\n";
16     cout << "size of p1 = " << sizeof(p1) << "\n";
17     cout << "size of p2 = " << sizeof(p2) << "\n";
18
19     return 0;
20 }
```

Output

```
x = 98.7654
&x = 0x22fe38
p1 = 0
p2 = 0x22fe38
&p1 = 0x22fe30
&p2 = 0x22fe28
size of p1 = 8
size of p2 = 8
```



Pointer Assignment

- Pointer can be assigned to another pointer if both of same type
- If not same type, cast operator must be used
- Exception: pointer to `void` (type `void *`)
  - Generic pointer, represents any type
  - No casting needed to convert pointer to `void` pointer
  - `void` pointers cannot be dereferenced

Pointer Assignment

```
int x = 55;
int *p1 = x;
```

[Error] invalid conversion from 'int' to 'int*' [-fpermissive]

```
int x = 55;
double *p = &x;
```

[Error] cannot convert 'int*' to 'double*' in initialization

```
int x = 55;
double y = 55.55;
void *p = &x;
cout << "&x =" << &x << "\n";
cout << "p =" << p << "\n";

p = &y;
cout << "&y =" << &y << "\n";
cout << "p =" << p << "\n« ;
```

Output

```
&x =0x22fe34
p =0x22fe34
&y =0x22fe28
p =0x22fe28
```

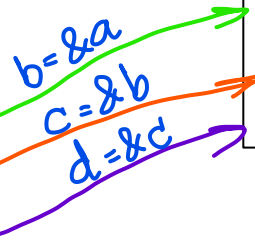
Pointers to Pointer

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

int main()
{
    int a = 98;
    int *b = &a;
    int **c = &b;
    int ***d = &c;

    cout << "a = " << a << "\t\t &a = " << &a << "\n";
    cout << "b = " << b << "\t &b = " << &b << "\n";
    cout << "c = " << c << "\t &c = " << &c << "\n";
    cout << "d = " << d << "\t &d = " << &d << "\n";

    return 0;
}
```

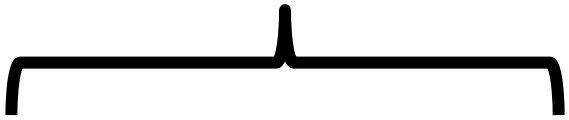


Output

```
a = 98          &a = 0x22fe3c
b = 0x22fe3c    &b = 0x22fe30
c = 0x22fe30    &c = 0x22fe28
d = 0x22fe28    &d = 0x22fe20
```

Pointers to Pointer Declarations

This pointer points to
pointer of this type



type * (* name)

Name of Pointer Variable

This variable type is pointer

Dereferencing Operator

"ค่าที่พอยเตอชี้ชี้ไป คือค่าอะไร"

- * (indirection/dereferencing operator)
  - Returns synonym for object its pointer operand points to
    - The value stored at the address
  - ***yPtr** returns **y** (because **yPtr** points to **y**)
  - dereferenced pointer is lvalue

```
*yptr = 9;      // assigns 9 to y
```

- * and & are inverses of each other

Dereferencing Operator

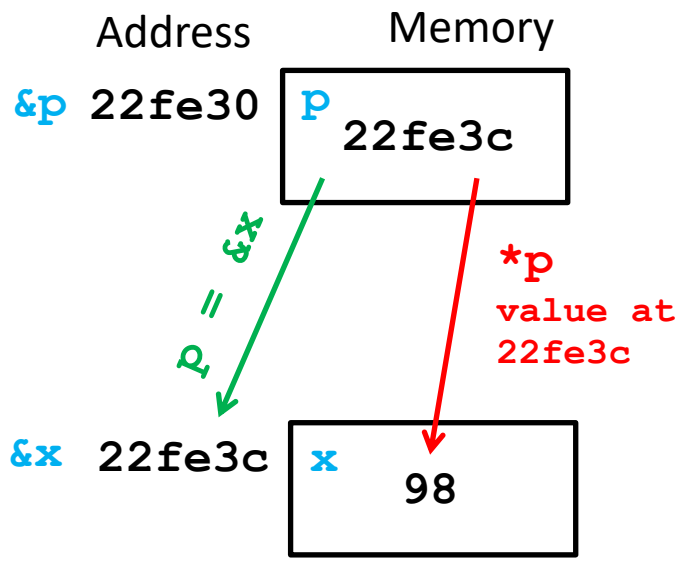
```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 98;
7      int *p;
8      p = &x;
9
10     cout << "x = " << x << "\n";
11     cout << "&x = " << &x << "\n";
12     cout << "p = " << p << "\n";
13     cout << "&p = " << &p << "\n";
14     cout << "*p = " << *p << "\n";
15
16     return 0;
17 }
```

Output

```
x = 98
&x = 0x22fe3c
p = 0x22fe3c
&p = 0x22fe30
*p = 98
```

`int * p` = p is a pointer to data type int
`&x` = address of x
`p` = p store an address of x (p points to x)
`*p` = value at address stored in p
(value that p points to)

If `p = &x` then `*p = x`



Dereferencing Operator

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 98;
7      int *p = &x;
8
9      cout << "*&p = " << *&p << "\n";
10     cout << "&*p = " << &*p << "\n";
11
12     return 0;
13 }
```

หักล้างกัน - เป็นอินเวอร์ส

***&p = p**

&p = 22fe30

*&p = value at 22fe30 = 22fe3c

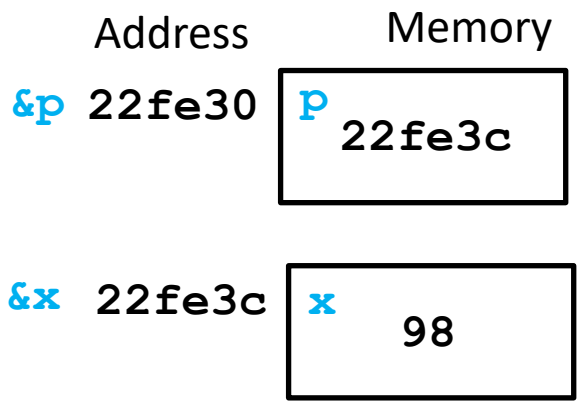
&*p = p

*p = x

&*p = &x = 22fe3c

Output

```
*&p = 0x22fe3c
&*p = 0x22fe3c
```



Dereferencing Operator

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int a = 98;
7       int *b = &a; b=&a
8       int **c = &b; c=&b
9       int ***d = &c; d=&c
10
11      cout << "a = " << a << "\t\t &a = " << &a << "\n";
12      cout << "b = " << b << "\t &b = " << &b << "\t *b = " << *b << "\n";
13      cout << "c = " << c << "\t &c = " << &c << "\t *c = " << *c << "\n";
14      cout << "d = " << d << "\t &d = " << &d << "\t *d = " << *d << "\n";
15
16      return 0;
17  }
```

Output

```
a = 98          &a = 0x22fe3c
b = 0x22fe3c    &b = 0x22fe30    *b = 98
c = 0x22fe30    &c = 0x22fe28    *c = 0x22fe3c
d = 0x22fe28    &d = 0x22fe20    *d = 0x22fe30
```

b=&a

c=&b

d=&c

b = 0x22fe3c = &a

c = 0x22fe30 = &b

d = 0x22fe28 = &c

*b = 98 = a

*c = 0x22fe3c = b

*d = 0x22fe30 = c

*(&c) = c

Dereferencing Operator

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 12;
7      int y = 13;
8      int *p = &x;
9      *(&x) = x
10     *p = 4;
11     p = &y;
12     *p = (*p)*2+x;
13     *(&y) = y → y = y*2+x
14     cout << "x = " << x << "\n";
15     cout << "y = " << y << "\n";
16
17     return 0;
18 }
```

p = &x

x = 4

Output

```
x = 4
y = 30
```

y = y*2+x

Reference Variable

ชื่อตัวอื่นของตัวแปร

type & new_name = target_name;

- A reference variable is a variable which "refers to" another named or unnamed variable.
- References must be initialized at the point of instantiation.
- A reference variable can not refer to nothing (NULL).
- Once a reference is set to refer to a particular variable, you cannot, during its lifetime, "rebind" it to refer to a different variable.

Reference Variable

```
1  #include <iostream>
2  using namespace std;
3
4
5  int main()
6  {
7      int x = 17;
8      int &y = x;
9
10     cout << "x = " << x << " y = " << y;
11
12     x = 55;
13     cout << "\nx = " << x << " y = " << y;
14
15     y = 0;
16     cout << "\nx = " << x << " y = " << y;
17 }
```

"y เป็นอีกชื่อหนึ่งของx"
y กับx - ชี้ address เดียวกัน
y - x

Output

```
x = 17 y = 17
x = 55 y = 55
x = 0 y = 0
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 17;
7      int &y = x;
8      int *p = &y;
9
10     cout << "&x = " << &x << "\n";
11     cout << "&y = " << &y << "\n";
12     cout << "*p = " << *p << "\n";
13
14     return 0;
15 }
```

y - x
p = &y = &x
*(&y) = y = x

Output

```
&x = 0x22fe2c
&y = 0x22fe2c
*p = 17
```

x and y refer to the same location of the memory
(one variable with two names)

Calling Function by Reference

- 3 ways to pass arguments to function
  - Pass-by-value
  - Pass-by-reference with reference [&] arguments
  - Pass-by-reference with pointer arguments
- `return` can return one value from function
- Arguments passed to function using reference arguments
  - Modify original values of arguments
  - More than one value "returned"

Calling Function by reference arguments

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

void swap(int &,int &);

int main()
{
    int a = 9, b = 6;
    cout << "a = " << a << " &a = " << &a << "\n";
    cout << "b = " << b << " &b = " << &b << "\n";
    swap(a,b);
    cout << "a = " << a << " &a = " << &a << "\n";
    cout << "b = " << b << " &b = " << &b << "\n";

}

void swap(int &x, int &y){
    cout << "x = " << x << " &x = " << &x << "\n";
    cout << "y = " << y << " &y = " << &y << "\n";

    int temp = x;
    x = y;
    y = temp;
}
```

```
a = 9  &a = 0x22fe3c
b = 6  &b = 0x22fe38
x = 9  &x = 0x22fe3c
y = 6  &y = 0x22fe38
a = 6  &a = 0x22fe3c
b = 9  &b = 0x22fe38
```

Output

Calling Function by pointer arguments

- Pass-by-reference with pointer arguments
  - Simulate pass-by-reference
    - Use pointers and indirection operator
  - Pass address of argument using **&** operator
  - Arrays not passed with **&** because array name already pointer
  - ***** operator used as alias/nickname for variable inside of function

Calling Function by pointer arguments

```
a = 9 &a = 0x22fe3c
b = 6 &b = 0x22fe38
x = 0x22fe3c &x = 0x22fe10 *x = 9
y = 0x22fe38 &y = 0x22fe18 *y = 6
a = 6 &a = 0x22fe3c
b = 9 &b = 0x22fe38
```

Output

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int *,int *);
5
6  int main()
7  {
8      int a = 9, b = 6;
9      cout << "a = " << a << " &a = " << &a << "\n";
10     cout << "b = " << b << " &b = " << &b << "\n";
11     swap(&a,&b);
12     cout << "a = " << a << " &a = " << &a << "\n";
13     cout << "b = " << b << " &b = " << &b << "\n";
14
15 }
16
17 void swap(int *x, int *y){
18     cout << "x = " << x << " &x = " << &x << " *x = " << *x << "\n";
19     cout << "y = " << y << " &y = " << &y << " *y = " << *y << "\n";
20
21     int temp = *x;
22     *x = *y;
23     *y = temp;
24 }
```

[int] *x=&a → x=&a y=&b

Calling Function (Summary)

	Prototype	Definition	Calling

By value	`int func(int);`	`int func(int x){`
		`  x = 2*x;`
		`  return x;`
		`}`
			`int x = 0;`
			`func(x);`

By reference (reference arguments)	`void func(int &);`	`void func(int &x){`
		`  x = 2*x;`
		`}`
			`int x = 0;`
			`func(x);`

By reference (pointer arguments)	`void func(int *);`	`void func(int *x){`
		`  *x = 2*(*x);`
		`}`
			`int x = 0;`
			`func(&x);`
			--- 	
			`int *p = &x;`
			`func(p);`

By reference (array arguments)	`void func(int []);`	`void func(int x[]){`
		`  x[0] = 2*x[0];`
		`}`
			`int x[] = {0,0,0};`
			`func(x);`

Accessing the value of a reference works the same way as accessing a normal value -- no asterisk needed.

& and *

	Declaration	Operator
&	`int & x;` → x = □ Used for declaring reference variable	`&x;` Return address of variable x
*	`int * x;` → x = □ Used for declaring pointer variable	`*x;` *x = *(&x) = x Return value that pointer x points to

Using const with Pointers

"ไม่สามารถเปลี่ยนค่าได้"

- `const` pointers
  - Always point to same memory location
  - Default for array name
  - Must be initialized when declared

- Four ways to pass pointer to function
  - Nonconstant pointer to nonconstant data
    - Highest amount of access
  - Nonconstant pointer to constant data
  - Constant pointer to nonconstant data
  - Constant pointer to constant data
    - Least amount of access

Using const with Pointers

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 6;
7      const int y = 9;
8
9      int * const p1 = &x;
10     *p1 = 0;
11     cout << "x = " << x << "\n";
12     // p1 = 0;
13
14     const int *p2 = &y;
15     p2 = 0;
16     cout << "p2 = " << p2 << "\n";
17     // *p2 = 0;
18
19     const int * const p3 = &y;
20     // p3 = 0;
21     // *p3 = 0;
22
23     return 0;
24 }
```

x เปลี่ยนค่าได้ แต่ p1 เปลี่ยนค่าไม่ได้

p2 เปลี่ยนได้ แต่ y เปลี่ยนไม่ได้

เปลี่ยนไม่ได้ทั้ง 2 ค่า

Output

```
x = 0
p2 = 0
```

1) Constant pointer that point to non-constant data
(Data can be modified but pointer can not be changed)

2) Non-constant pointer that point to constant data
(Data can not be modified but pointer can be changed)

3) Constant pointer that point to constant data
(Both data and pointer can not be modified)

Using const with Pointers

1) ไม่สามารถเปลี่ยนค่าที่ x ชี้ไปได้ ; ย้อยเต้อเปลี่ยนได้
2) เปลี่ยนค่าย้อยเต้อไม่ได้ ; เปลี่ยนค่าที่ย้อยเต้อชี้ไปได้
3) เปลี่ยนค่าย้อยเต้อและค่าที่ชี้ไปไม่ได้

```
1   #include <iostream>
2   using namespace std;
3
4   void myFunc(const int *x, int * const y, const int * const z){
5       int temp = -999;
6
7       // *x = 0; << Illegal
8       x = &temp; //OK
9
10      // y = &temp; << Illegal
11      *y = *x+2*(*z)+999;
12
13      // z = &temp; << Illegal
14      // *z = 0; << Illegal
15  }
16
17  int main()
18  {
19      int a = 1, b = 5, c = 10;
20      myFunc(&a,&b,&c);
21      cout << "a = " << a << "\n";
22      cout << "b = " << b << "\n";
23      cout << "c = " << c << "\n";
24
25      return 0;
26  }
```

Output

```
a = 1
b = 20
c = 10
```

Example of standard function that uses pointers

#include

<algorithm>

function template

std::**sort** (เรียงลำดับ)

Function prototype

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

Sort elements in range

Sorts the elements in the range [first,last) into ascending order.

Meaning of each parameter

Parameters

`first, last`

Random-access iterators to the initial and final positions of the sequence to be sorted. The range used is [first,last), which contains all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.

Return value

none

•http://www.cplusplus.com/reference/algorithm/sort/

Example of standard function that uses pointers

function template

<algorithm>

std:sort เรียงลำดับ

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

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

int main(){
     int data[] = {7,8,2,1,2,4,6,9,0,1};
     cout << "Before: ";
     for(int i = 0; i<10; i++) cout << data[i] << " ";
     sort(&data[0],&data[10]);
     cout << "\nAfter: ";
     for(int i = 0; i<10; i++) cout << data[i] << " ";
       return 0;
}
```

data[0] data[9] [first, last)

Output

```
Before: 7 8 2 1 2 4 6 9 0 1
After: 0 1 1 2 2 4 6 7 8 9
```

•http://www.cplusplus.com/reference/algorithm/sort/

Example of standard function that uses pointers

Function prototype

function

strtof (C++11)

<cstdlib>

```
float strtof (const char* str, char** endptr);
```

Convert string to float

Parses the C-string *str* interpreting its content as a floating point number (according to the current locale) and returns its value as a `float`. If *endptr* is not a *null pointer*, the function also sets the value of *endptr* to point to the first character after the number.

The function first discards as many whitespace characters (as in `isspace`) as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax resembling that of floating point literals (see below), and interprets them as a numerical value. A pointer to the rest of the string after the last valid character is stored in the object pointed by *endptr*.

A valid floating point number for `strtof` using the `"C"` locale is formed by an optional sign character (`+` or `-`), followed by one of:

- A sequence of digits, optionally containing a decimal-point character (`.`), optionally followed by an exponent part (an `e` or `E` character followed by an optional sign and a sequence of digits).
- A `0x` or `0X` prefix, then a sequence of hexadecimal digits (as in `isxdigit`) optionally containing a period which separates the whole and fractional number parts. Optionally followed by a power of 2 exponent (a `p` or `P` character followed by an optional sign and a sequence of hexadecimal digits).
- `INF` or `INFINITY` (ignoring case).
- `NAN` or `NAN`*sequence* (ignoring case), where *sequence* is a sequence of characters, where each character is either an alphanumeric character (as in `isalnum`) or the underscore character (`_`).

If the first sequence of non-whitespace characters in *str* does not form a valid floating-point number as just described, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed and the function returns `0.0F`.

str: char *

Pointer to the first element of string

(String is passed by reference and can't be modified inside function but the pointer can be modified.)

endptr: char **

Pointer to the pointer variable that point to the end of string

(We want to return pointer that point to the end of string from this function so this pointer is passed by reference by using another pointer to point it.)

Meaning of each parameter

Parameters

`str`
C-string beginning with the representation of a floating-point number.

`endptr`
Reference to an already allocated object of type `char*`, whose value is set by the function to the next character in *str* after the numerical value.
This parameter can also be a *null pointer*, in which case it is not used.

http://www.cplusplus.com/reference/cstdlib/strtof/