261102
Computer Programming

Lecture 16: Pointers I

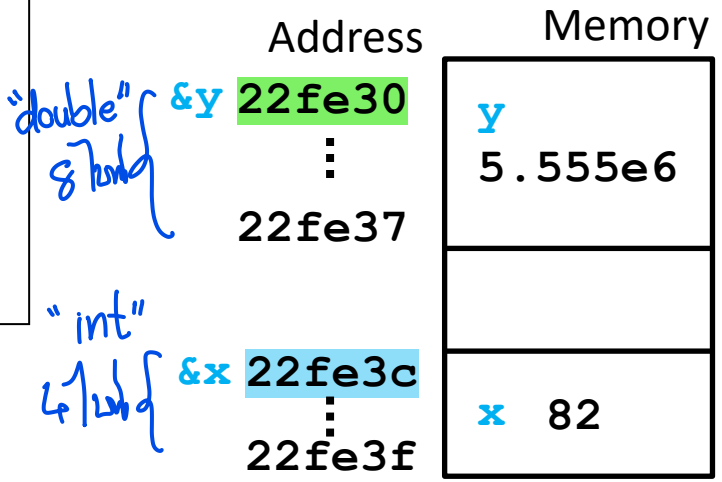
Memory Address

- **&** (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
```

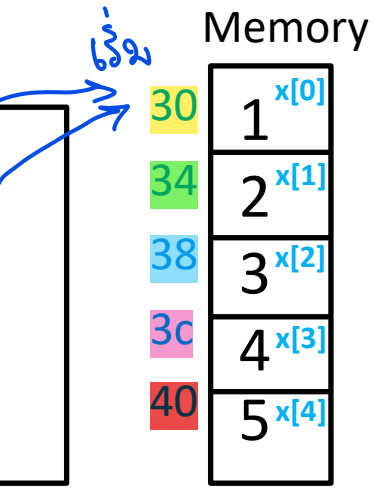


Memory Address

```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

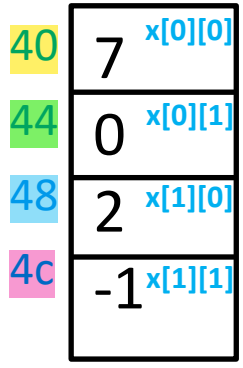
```
&x = 0x22fe30
&x[0] = 0x22fe30
&x[1] = 0x22fe34
&x[2] = 0x22fe38
&x[3] = 0x22fe3c
&x[4] = 0x22fe40
```




```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
```

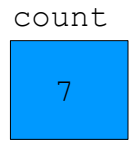


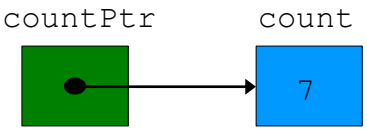
Pointer

"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

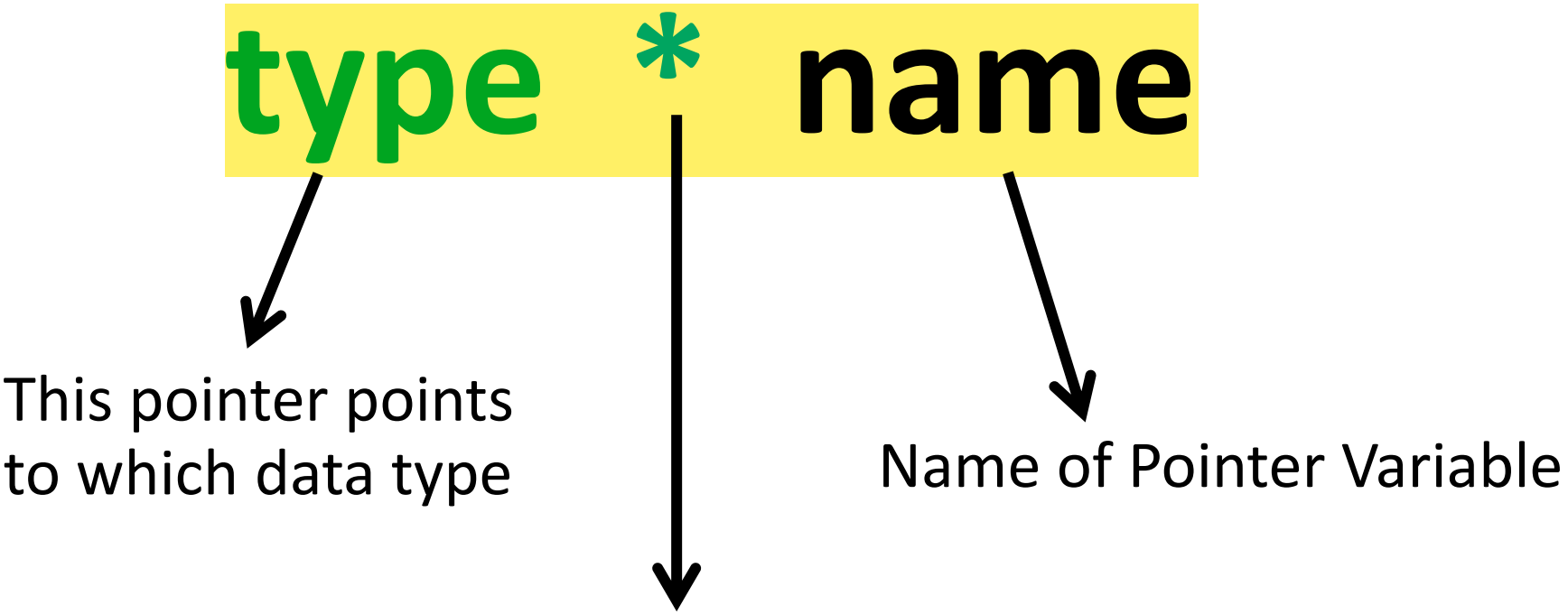
```
int *myPtr;
```

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;
```

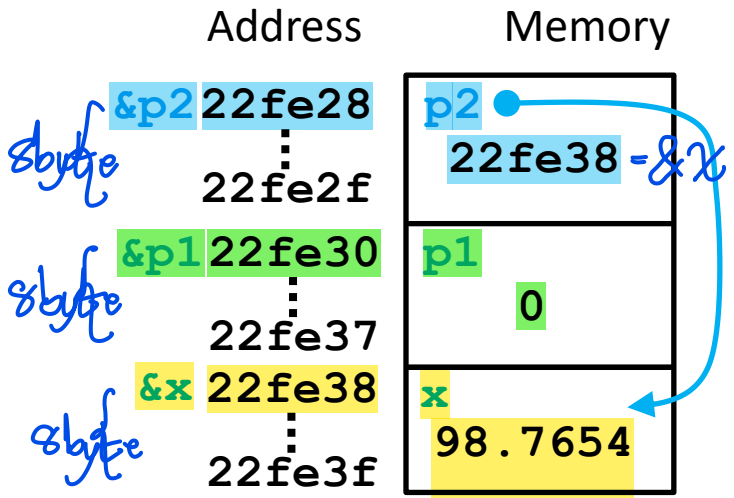
1 2

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

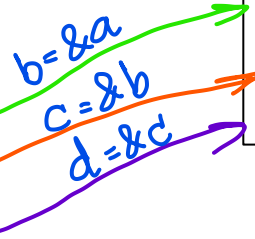
```
int x = 55;
double y = 55.55;
void *p = &x;
cout << "&x =" << &x << "\n";
cout << "p =" << p << "\n";

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;
}
```

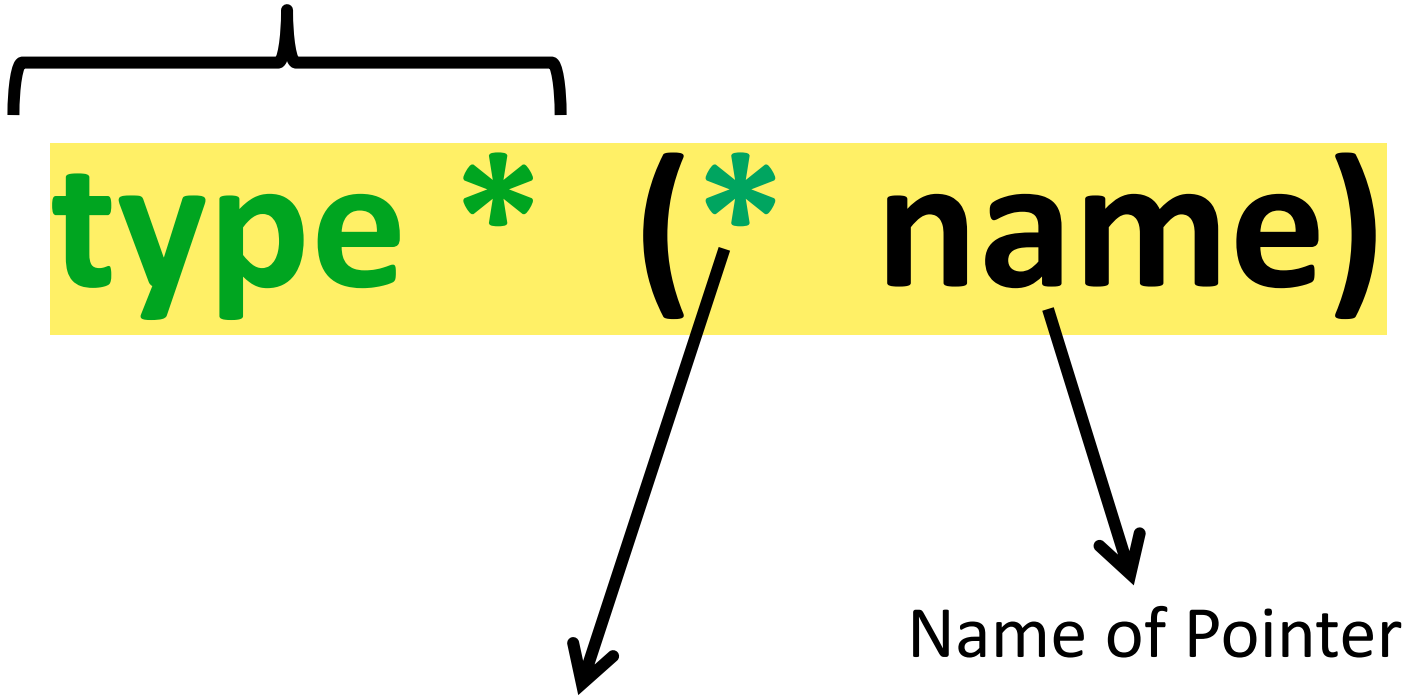


Output

```
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



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

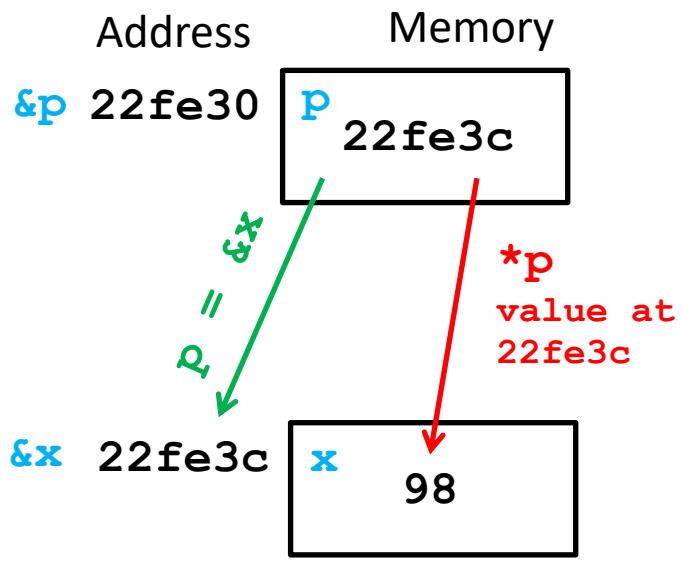
```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x = 98;
7      int *p;
8      p = &x;
9
10     cout << "x = " << x << "\n";
11     cout << "&x = " << &x << "\n";
12     cout << "p = " << p << "\n";
13     cout << "&p = " << &p << "\n";
14     cout << "*p = " << *p 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

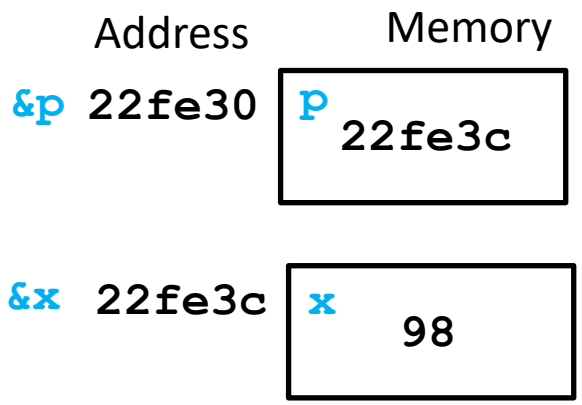
&*p = p

*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;
8       int **c = &b;
9       int ***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
```

Reference Variable

ชื่อตัวที่อ้างถึง ของตัวแปร

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 }
```

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 }
```

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

```
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 } [int]&x=a; x=a  y=b
16
17 void swap(int &x, int &y){
18     cout << "x = " << x << " &x = " << &x << "\n";
19     cout << "y = " << y << " &y = " << &y << "\n";
20
21     int temp = x;
22     x = y;
23     y = temp;
24 }
```

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

x=a, y=b, swap, a, b

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

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

int main()
{
    int x = 6;
    const int y = 9;

    int * const p1 = &x;
    *p1 = 0;
    cout << "x = " << x << "\n";
    // p1 = 0;

    const int *p2 = &y;
    p2 = 0;
    cout << "p2 = " << p2 << "\n";
    // *p2 = 0;

    const int * const p3 = &y;
    // p3 = 0;
    // *p3 = 0;

    return 0;
}
```

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/

261102
Computer Programming

Lecture 17: Pointers II

Pointer to Array

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int x[] = {55,43,-12};
7       int *p = x;
8       // or    int *p = &x[0];
9
10      cout << "&x = " << &x << "\n";
11      cout << "&x[0] = " << &x[0] << "\n";
12      cout << "&x[1] = " << &x[1] << "\n";
13      cout << "&x[2] = " << &x[2] << "\n";
14      cout << "p = " << p << "\n";
15      cout << "p+1 = " << p+1 << "\n";
16      cout << "p+2 = " << p+2 << "\n";
17      return 0;
18  }
```

x[5] = { }
cout << x ;
Output : address x , address x[0]

Output

```
&x = 0x22fe20
&x[0] = 0x22fe20
&x[1] = 0x22fe24
&x[2] = 0x22fe28
p = 0x22fe20
p+1 = 0x22fe24
p+2 = 0x22fe28
```

+4 +8

- Array name already pointer
- Array name **x** is a pointer to **&x[0]**
- Return data type of **&x** is **int (*) [3]**
- Return data type of **&x[0]** is **int ***

Pointer to Array

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       double x[] = {55,43,-12};
7       double *p = x;
8       // or    double *p = &x[0];
9
10      cout << "&x = " << &x << "\n";
11      cout << "&x[0] = " << &x[0] << "\n";
12      cout << "&x[1] = " << &x[1] << "\n";
13      cout << "&x[2] = " << &x[2] << "\n";
14      cout << "p = " << p << "\n";
15      cout << "p+1 = " << p+1 << "\n";
16      cout << "p+2 = " << p+2 << "\n";
17      return 0;
18  }
```

Output

```
&x = 0x22fe20
&x[0] = 0x22fe20
&x[1] = 0x22fe28
&x[2] = 0x22fe30
p = 0x22fe20
p+1 = 0x22fe28
p+2 = 0x22fe30
```

`p` is pointer to `double`

Now address shift (+1) = 8 bytes

Pointer to Array

"ฟังชันต์ชี้ไปที่ array"

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x[] = {55,43,-12};
7      int (*p) [3] = &x;
8
9      cout << "p = " << p << "\n";
10     cout << "p+1 = " << p+1 << "\n";
11     cout << "p+2 = " << p+2 << "\n";
12     return 0;
13 }
```

Output

```
p = 0x22fe20
p+1 = 0x22fe2c
p+2 = 0x22fe38
```

+12, +24

p is pointer to 3-element int array

Now address shift (+1) = 12 bytes

(4*3)

cout << char *

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

Output

```
&x = 0x22fe40
&x[0] = Strong
&x[1] = trong
&x[2] = rong
&x[3] = ong
&x[4] = ng
&x[5] = g
```

ปกติ cout ค่า pointer ของ type อื่น จะแสดง address อันนี้แม่ง Indy

char array is a special case that have different behavior on `cout << char *`

cout << char *

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

Output

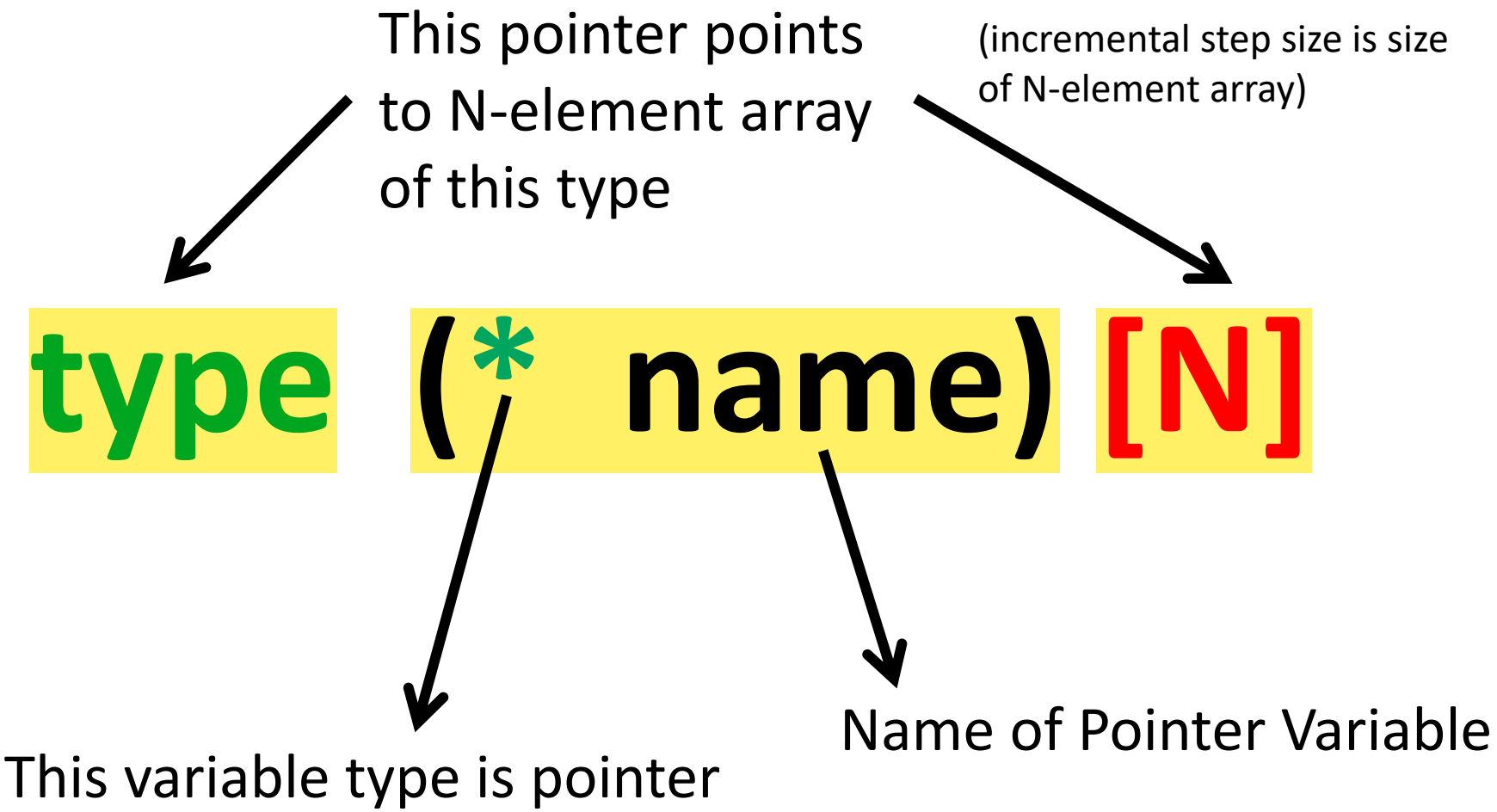
```
&x = 0x22fe40
&x[0] = 0x22fe40
&x[1] = 0x22fe41
&x[2] = 0x22fe42
&x[3] = 0x22fe43
&x[4] = 0x22fe44
&x[5] = 0x22fe45
```

ถ้าอยากจะโชว์เป็น Address

Using type casting to other pointer types such as **int *** (pointer to int), **void *** (void pointer)

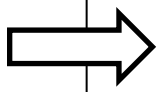
- **&x** returns **char (*) [7]** (pointer to char array of 7-element)
- **&x[0]** returns **char (*)** (pointer to char)

Pointers to Array Declarations

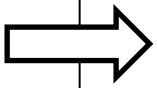


Pointers to Array Declarations

```
int x[] = {1,2,3};

int * a = x;
int * b = &x[0];

int (*c) [3] = &x;
```



Point to first element (int)



Point to whole array (int[3])

Pointer Arithmetic

- Increment/decrement pointer (++ or −−)
- Add/subtract an integer to/from a pointer (+ or += , − or −=)
- Pointers may be subtracted from each other
- Pointer arithmetic meaningless unless performed on pointer to array

Pointer Arithmetic

```
int x = 55;
double y = 55.55;
int *p = &x;
cout << "&x =" << &x << "\n";
cout << "p =" << p << "\n";
cout << "p+1 =" << p+1 << "\n";

double *q = &y;
cout << "&y =" << &y << "\n";
cout << "q =" << q << "\n";
cout << "q+1 =" << q+1 << "\n";
```

Output

```
&x =0x22fe2c
p =0x22fe2c +4
p+1 =0x22fe30
&y =0x22fe20
q =0x22fe20 +8
q+1 =0x22fe28
```

Pointer Arithmetic

```
int x = 55;
double y = 55.55;
void *p = &x;
cout << "&x =" << &x << "\n";
cout << "p =" << p << "\n";
cout << "p+1 =" << p+1 << "\n";

p = &y;
cout << "&y =" << &y << "\n";
cout << "p =" << p << "\n";
cout << "p+1 =" << p+1 << "\n";
```

Output

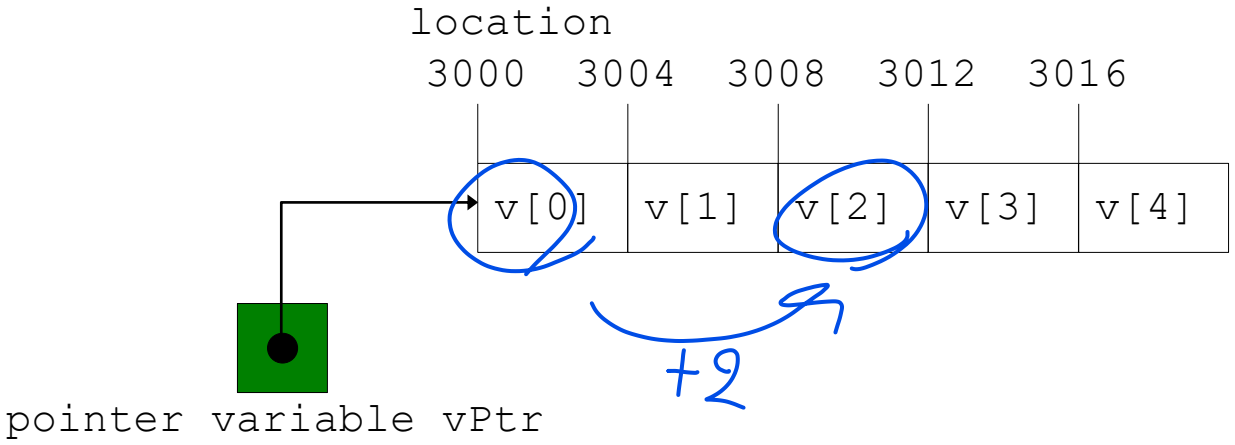
```
&x =0x22fe34
p =0x22fe34 +1
p+1 =0x22fe35
&y =0x22fe28
p =0x22fe28 +1
p+1 =0x22fe29
```

Pointer Arithmetic

- 5 element **`int`** array on a machine using 4 byte **`int`**s
  - **`vPtr`** // points to first element **`v[0]`**

    `vPtr = 3000`
  - **`vPtr += 2;`** // sets **`vPtr`** to **`3008`**

    `vPtr` points to **`v[ 2 ]`**



Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements between two addresses

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x[] = {55,43,-12};
7      int *p1 = &x[0];
8      int *p2 = &x[2];
9
10     cout << "p1 = " << p1 << "\n";
11     cout << "p2 = " << p2 << "\n";
12     cout << "p2-p1 = " << p2-p1 << "\n";
13     return 0;
14 }
```

Output

```
p1 = 0x22fe20
p2 = 0x22fe28
p2-p1 = 2
```

จำนวน element ที่ห่างกัน

Pointer Comparison

- Use equality and relational operators
- Comparisons meaningless unless pointers point to members of same array
- Compare addresses stored in pointers
- Example: could show that one pointer points to higher numbered element of array than other pointer
- Common use to determine whether pointer is 0 (does not point to anything: NULL)

Pointer Comparison

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int x[] = {1,2,3};
7       int *p1 = &x[0];
8       int *p2 = &x[2];
9
10      if(p2 > p1) cout << "p2 is point to the element after p1.\n";
11      else if(p2 < p1) cout << "p2 is point to the element before p1.\n";
12      else cout << "p2 is point to the same element at p1.\n";
13
14      if(p2 == 0) cout << "p2 is a NULL pointer.\n";
15      else cout << "p2 is not a NULL pointer.\n";
16
17      return 0;
18  }
```

Output

```
p2 is point to the element after p1.
p2 is not a NULL pointer.
```

Dereferencing Operator

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int x[] = {55,43,-12};
7       int *p = x;
8
9       cout << "p = " << p << "\n";
10      cout << "*p = " << *p << "\n";
11      cout << "*(p+1) = " << *(p+1)<< "\n";
12      cout << "*(p+2) = " << *(p+2)<< "\n";
13      cout << "*p+1 = " << *p+1 << "\n";
14      return 0;
15  }
```

Output

```
p = 0x22fe20
*p = 55
*(p+1) = 43
*(p+2) = -12
*p+1 = 56
```

*p	equivalent to	x[0]
*(p+1)	equivalent to	x[1]
*(p+2)	equivalent to	x[2]

Array and Pointer Names

(ใช้แทนกันได้)

- Arrays and pointers closely related
  - Array name like constant pointer
  - Pointers can do array subscripting operations
- Accessing array elements with pointers
  - Element `b[n]` can be accessed by `*(bPtr + n)`
    - Called pointer/offset notation
  - Addresses
    - `&b[3]` same as `bPtr + 3`
  - Value : Array name can be treated as pointer
    - `b[3]` same as `*(b + 3)`
  - Pointers can be subscripted (pointer/subscript notation)
    - `bPtr[3]` same as `b[3]`

Array and Pointer Names

- A pointer variable can be used to access the elements of an array of the same type.

```
int gradeList[8] = {92,85,75,88,79,54,34,96};
int *myGrades = gradeList;

cout << gradeList[0] << " ";
cout << gradeList[2] << " ";

cout << *myGrades << " ";
cout << *(myGrades + 2) << " ";

cout << myGrades[3] << " ";
cout << *gradeList << " ";
cout << *(gradeList + 2);
```

Output

```
92 75 92 75 88 92 75
```

Array-style

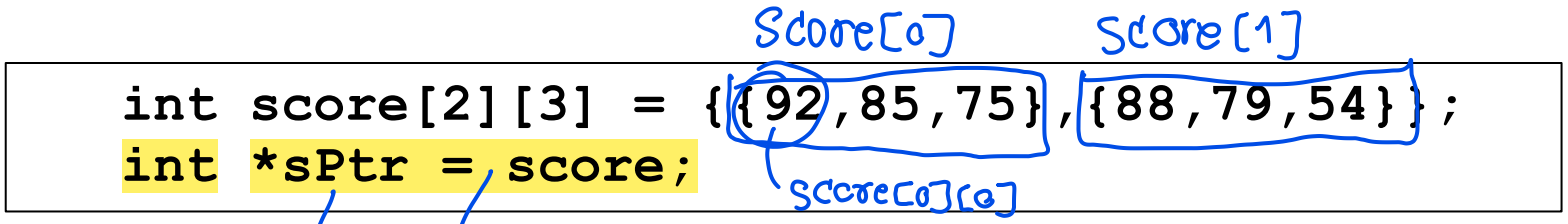
Pointer-style

Use pointer name as array name

Use array name as pointer name

Note that the array name **`gradeList`** acts like the pointer variable **`myGrades`**.

Pointers to 2D Arrays




```
int score[2][3] = {{92,85,75},{88,79,54}};
int *sPtr = score;
```

[Error] cannot convert 'int (*)[3]' to 'int*' in initialization

`int *sPtr = score;` is equivalent to `int *sPtr = &score[0];`

`score[0]` is int array with 3 elements = {92,85,75}

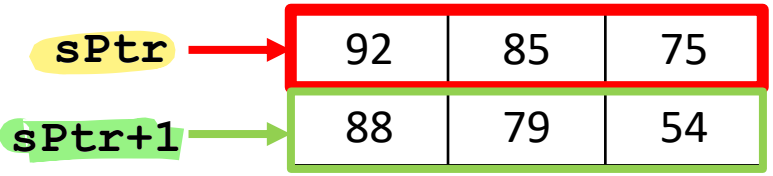
`&score[0]` returns `int (*) [3]` not `int *`

```
int score[2][3] = {{92,85,75},{88,79,54}};
int (*sPtr) [3] = score;
```

In this case, sPtr++ will move the address by 4*3 bytes

Pointers to 2D Arrays

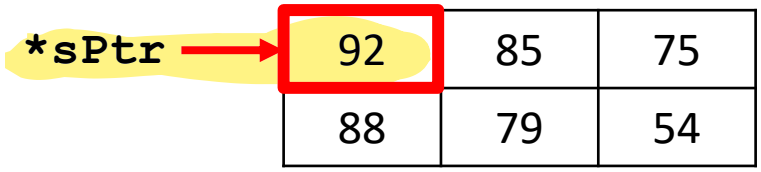
```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int score[2][3] = {{92,85,75},{88,79,54}};
7       int (*sPtr) [3] = score;
8
9       cout << "sPtr = " << sPtr << "\n";
10      cout << "sPtr+1 = " << sPtr+1 << "\n";
11      cout << "*sPtr = " << *sPtr << "\n";
12      cout << "**sPtr = " << **sPtr << "\n";
13      cout << "*(sPtr+1) = " << *(sPtr+1) << "\n";
14      cout << "**(sPtr+1) = " << **(sPtr+1) << "\n";
15
16      return 0;
17  }
```



sPtr = &score[0]

*sPtr = *&score[0]
= score[0]
= &score[0][0]

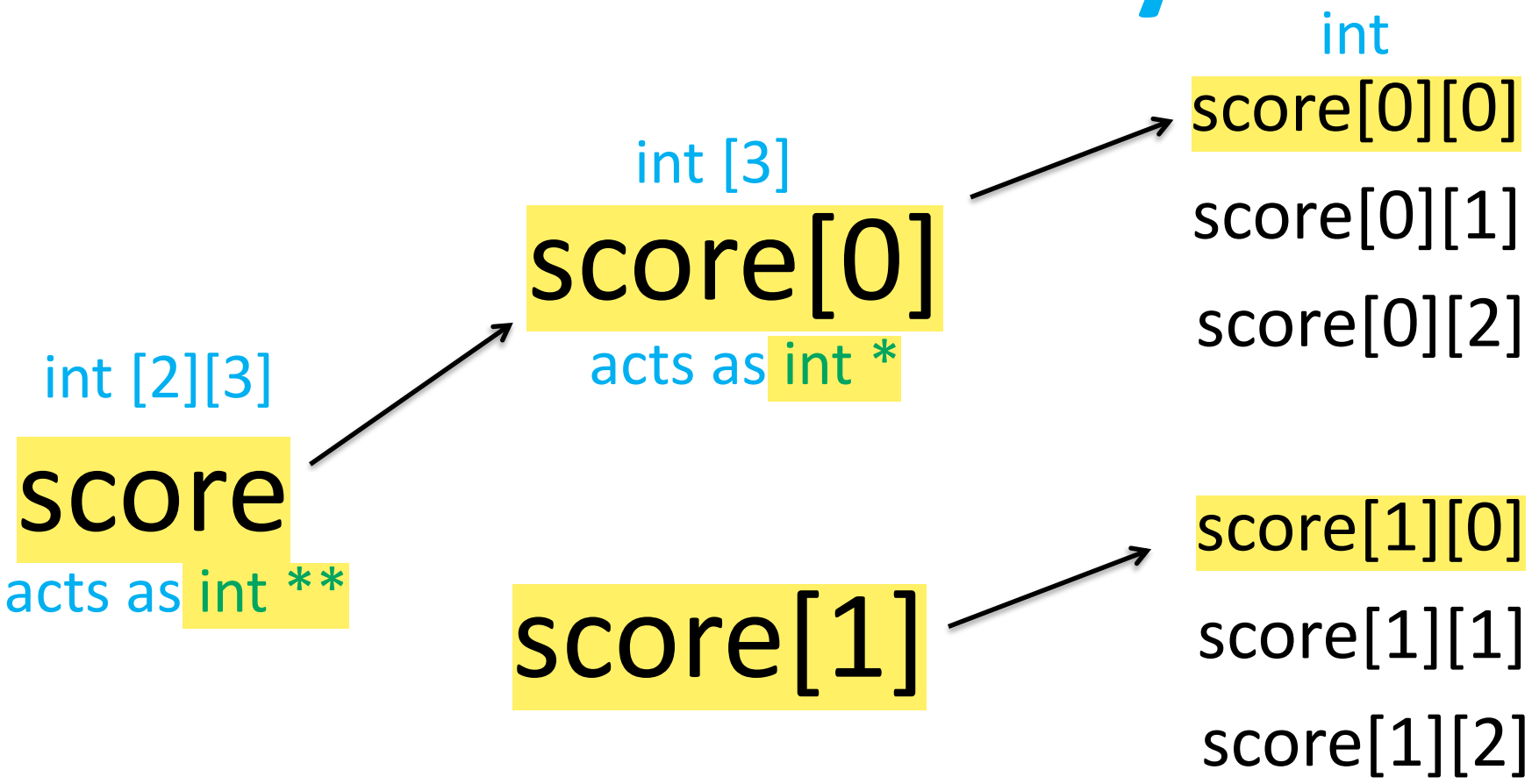
int*



Output

```
sPtr = 0x22fe20
sPtr+1 = 0x22fe2c
*sPtr = 0x22fe20
**sPtr = 92
*(sPtr+1) = 0x22fe2c
**(sPtr+1) = 88
```

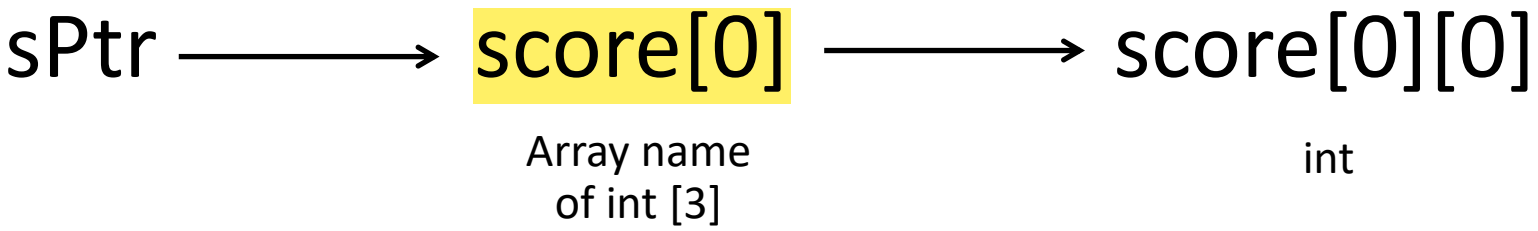
Pointers to 2D Arrays



Array name acts like a pointer

Pointers to 2D Arrays

sPtr = score; or sPtr = &score[0];



score[0] is array name that acts as pointer to score[0][0]

Value of score[0] becomes address of score[0][0]

*sPtr = address of score[0][0]

Pointers to 2D Arrays

sPtr ⟶ score[0] ⟶ score[0][0]

Array name of int [3]

int

**sPtr = *score[0] = value of score[0][0]

score[0] is array name that acts as pointer to score[0][0]

---

sPtr+1 ⟶ score[1] ⟶ score[1][0]

Array name of int [3]

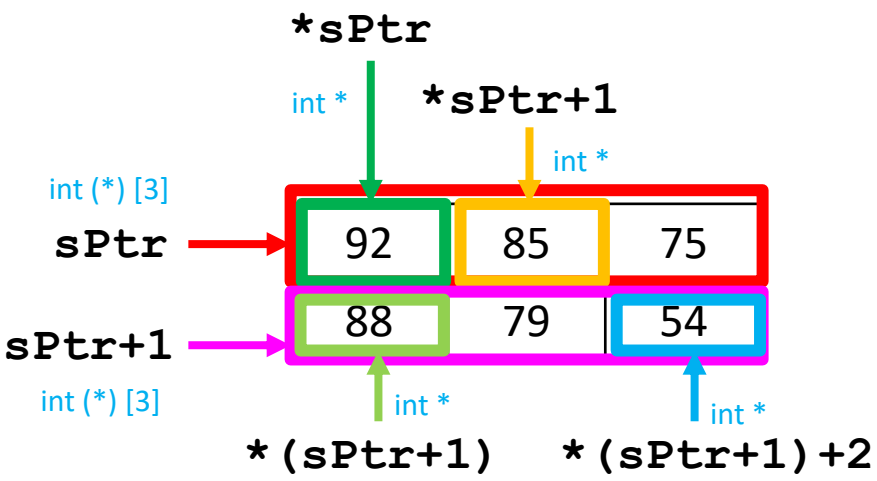
int

(Act as pointer stores address of score[1][0])

Pointers to 2D Arrays

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int score[2][3] = {{92,85,75},{88,79,54}};
7       int (*sPtr) [3] = score;
8
9       cout << "**sPtr = " << **sPtr << "\n"; //score[0][0]
10      cout << "**(sPtr+1) = " << **(sPtr+1) << "\n"; //score[1][0]
11      cout << "*(*sPtr+1) = " << *(*sPtr+1) << "\n"; //score[0][1]
12      cout << "*(*(sPtr+1)+2) = " << *(*(sPtr+1)+2) << "\n";  //score[1][2]
13
14      return 0;
15  }
```

sPtr is int (*) [3]
*sPtr is int *
**sPtr is int



Output

```
**sPtr = 92
**(sPtr+1) = 88
*(*sPtr+1) = 85
*(*(sPtr+1)+2) = 54
```

Pointers to 2D Arrays

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int score[2][3] = {{92,85,75},{88,79,54}};
7      int (*sPtr) [3] = score;
8
9      cout << "sPtr[0][0] = " << sPtr[0][0] << "\n"; //score[0][0]
10     cout << "*sPtr[1] = " << *sPtr[1] << "\n"; //score[1][0]
11     cout << "(*sPtr)[1] = " << (*sPtr)[1] << "\n"; //score[0][1]
12
13     cout << "**score = " <<  **score << "\n";  //score[0][0]
14     cout << "**(score+1) = " <<  **(score+1) << "\n";  //score[1][0]
15     cout << "*(*score+1) = " <<  *(*score+1) << "\n";  //score[0][1]
16
17     return 0;
18 }
```

Use pointer name as array name

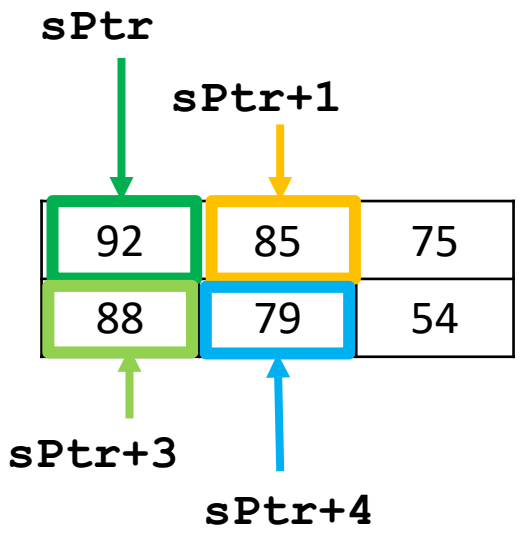
Use array name as pointer name

Output

```
sPtr[0][0] = 92
*sPtr[1] = 88
(*sPtr)[1] = 85
**score = 92
**(score+1) = 88
*(*score+1) = 85
```

Pointers to 2D Arrays

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int score[2][3] = {{92,85,75},{88,79,54}};
7       int *sPtr = score[0];
8       // or int *sPtr = &score[0][0];
9
10      cout << "sPtr = " << sPtr << "\n";
11      cout << "sPtr+1 = " << sPtr+1 << "\n";
12      cout << "*sPtr = " << *sPtr << "\n";
13      cout << "*(sPtr+1) = " << *(sPtr+1) << "\n";
14      cout << "*(sPtr+4) = " << *(sPtr+4) << "\n";
15
16
17      return 0;
18  }
```



Output

```
sPtr = 0x22fe20
sPtr+1 = 0x22fe24
*sPtr = 92
*(sPtr+1) = 85
*(sPtr+4) = 79
```

Array values are stored in consecutive addresses.

Arrays of Pointers

Array that contains N pointers

```
type *name[N]
```

Pointer that points to array

```
type (* name) [N]
```

Arrays of Pointers

```
int x[] = {1,2,3};

int * a = x;
int * b = &x[0];

int (*c) [3] = &x;

int * d [3];
int * (e[3]);
```

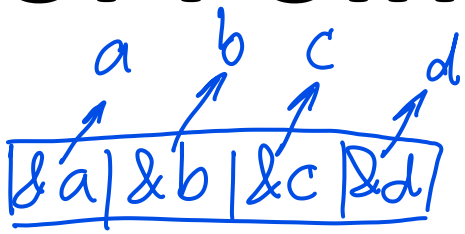
Point to first element (int)

Point to whole array (int[3])

[int* int* int*]

Array of three pointers
(point to int)

Arrays of Pointers



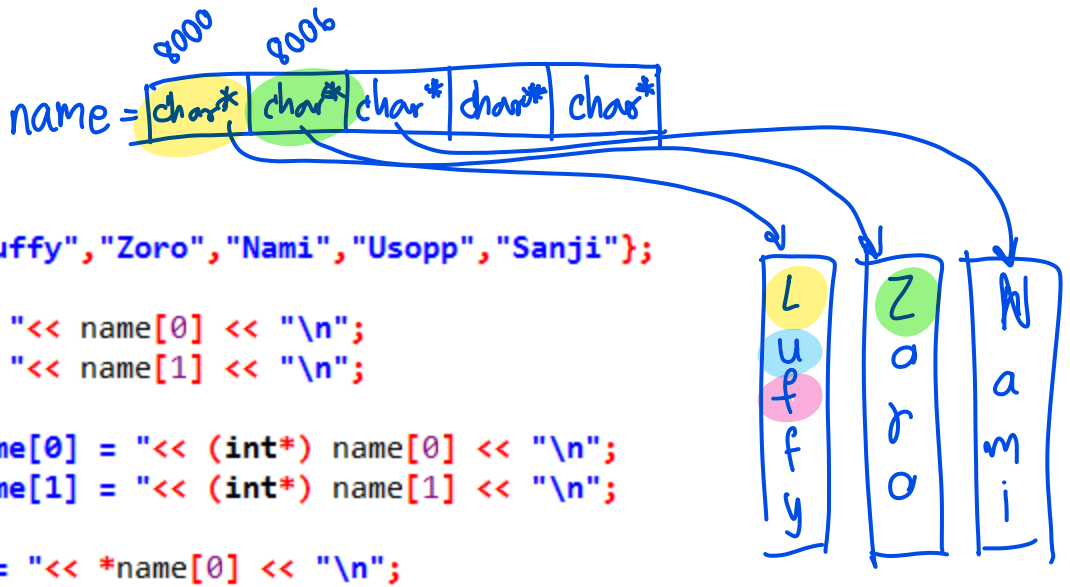

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main()
7  {
8      int a = 50, b = 100, c = 500, d = 1000;
9      int *sPtr[4];
10     sPtr[0] = &a; sPtr[1] = &b; sPtr[2] = &c; sPtr[3] = &d;
11
12     cout << "sPtr[0] = " << sPtr[0] << "\n";
13     cout << "sPtr[1] = " << sPtr[1] << "\n";
14     cout << "sPtr[2] = " << sPtr[2] << "\n";
15     cout << "sPtr[3] = " << sPtr[3] << "\n";
16
17     int temp,idx1,idx2;
18     srand(time(0));
19     for(int i = 0; i < 69; i++){
20         idx1 = rand()%4;
21         idx2 = rand()%4;
22         temp = *sPtr[idx1];
23         *sPtr[idx1] = *sPtr[idx2];
24         *sPtr[idx2] = temp;
25     }
26
27     cout << "a = " << a << "\n";
28     cout << "b = " << b << "\n";
29     cout << "c = " << c << "\n";
30     cout << "d = " << d << "\n";
31
32     return 0;
33 }
```

Output

```
sPtr[0] = 0x22fe2c
sPtr[1] = 0x22fe28
sPtr[2] = 0x22fe24
sPtr[3] = 0x22fe20
a = 1000
b = 500
c = 100
d = 50
```

Arrays of Pointers

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       char *name[] = {"Luffy","Zoro","Nami","Usopp","Sanji"};
7
8       cout << "name[0] = "<< name[0] << "\n";
9       cout << "name[1] = "<< name[1] << "\n";
10
11      cout << "(int*) name[0] = "<< (int*) name[0] << "\n";
12      cout << "(int*) name[1] = "<< (int*) name[1] << "\n";
13
14      cout << "*name[0] = "<< *name[0] << "\n";
15      cout << "*name[1] = "<< *name[1] << "\n";
16
17      cout << "*(name[0]+1) = "<< *(name[0]+1) << "\n";
18      cout << "*(name[0]+2) = "<< *(name[0]+2) << "\n";
19
20
21      return 0;
22  }
```



Output

```
name[0] = Luffy
name[1] = Zoro
(int*) name[0] = 0x488000
(int*) name[1] = 0x488006
*name[0] = L
*name[1] = Z
*(name[0]+1) = u
*(name[0]+2) = f
```

Similar to

```
string name[] = {"Luffy","Zoro","Nami","Usopp","Sanji"};
cout << "name[0] = "<< name[0] << "\n";
cout << "name[1] = "<< name[1] << "\n";
// Deferencing operator * can not be used for C++ string object
```

Example 17-A: Destiny Draw

```
1   #include <iostream>
2   #include <cstdlib>
3   #include <ctime>
4   using namespace std;
5
6   void deckShuffle(int *);
7   void showDrawSequence(int [][13]);
8   void draw(int [][13], int *);
9   char suit[] = "\3\4\5\6";
10  char *face[] = {"A","2","3","4","5","6","7","8","9","10","J","Q","K"};
11
12  int main()
13  {
14      int deck[4][13] = {};
15      int count = 1;
16      srand(time(0));
17      deckShuffle(&deck[0][0]);
18      showDrawSequence(deck);
19
20      for(int i = 0; i < 3;i++){
21          draw(deck,&count);
22      }
23
24      return 0;
25  }
```

Array of char

Array of char *

```
------------ Draw Sequence ------------
    A  2  3  4  5  6  7  8  9 10  J  Q  K
♥  22 35 20  6 36 27 41 21 32 15 43 31 52
♦  16 18 26 50 37  4 24 30 33 23 47 44 11
♣  39 34  7 13 45 14  8 49 12 40  5 25  1
♠   2 51 10 29 48 19 46 17 28 42  9 38  3
---------------------------------------
You got K♣
You got A♠
You got K♠
```

Example 17-A: Destiny Draw

Pass by pointer (point to `deck[0][0]`)

```
27  void deckShuffle(int *dptr){
28      int row,col;
29      for(int i = 1; i <= 52; i++){
30          do{
31              row = rand()%4;
32              col = rand()%13;
33          }while(*(dptr+col+13*row) != 0);
34          *(dptr+col+13*row) = i;
35      }
36  }
```

If `deck[row][col]` is equal to 0
set `deck[row][col] = i`

```
17      deckShuffle(&deck[0][0]);
```

Function calling (pass address of `deck[0][0]`)
pointer `dptr = &deck[0][0]`

dptr ⟶ `deck[0][0]`
dptr+1 ⟶ `deck[0][1]`
dptr+2 ⟶ `deck[0][2]`

dptr+12 ⟶ `deck[0][12]`
dptr+13 ⟶ `deck[1][0]`
dptr+13+1 ⟶ `dect[1][1]`

Example 17-A: Destiny Draw

```
38  void showDrawSequence(int d[][13]){
39      cout << "------------ Draw Sequence ------------\n";
40
41      cout << "    ";
42      for(int j = 0; j < 13; j++){
43          if(face[j] == "9") cout << face[j] << " ";
44          else cout << face[j] << "  ";
45      }
46      cout << "\n";
47
48      for(int i = 0;i < 4; i++){
49          cout << suit[i] << "  ";
50          for(int j = 0; j < 13; j++){
51              if(d[i][j] <= 9) cout << " ";
52              cout << d[i][j] << " ";
53          }
54          cout << "\n";
55      }
56      cout << "---------------------------------------\n";
57  }
```

Function call

```
18      showDrawSequence(deck);
```

```
------------ Draw Sequence ------------
    A  2  3  4  5  6  7  8  9 10  J  Q  K
♥  22 35 20  6 36 27 41 21 32 15 43 31 52
♦  16 18 26 50 37  4 24 30 33 23 47 44 11
♣  39 34  7 13 45 14  8 49 12 40  5 25  1
♠   2 51 10 29 48 19 46 17 28 42  9 38  3
---------------------------------------
You got K♣
You got A♠
You got K♠
```

Example 17-A: Destiny Draw

Pass by array (`deck`)

Pass by pointer (point to `count`)

```
59  void draw(int d[][13], int *c){
60      for(int i = 0;i < 4; i++){
61          for(int j = 0; j < 13; j++){
62              if(d[i][j] == *c){
63                  cout << "You got " << face[j] << suit[i] << "\n";
64                  (*c)++;
65                  return;
66              }
67          }
68      }
69  }
```

`char *` `char`

261102
Computer Programming

Lecture 18: Dynamic Memory Allocation

Types of Program Data

- *Static Data*: Memory allocation exists throughout execution of program
- *Automatic Data*: Automatically created at function entry, resides in activation frame of the function, and is destroyed when returning from function
- *Dynamic Data*: Explicitly allocated and deallocated during program execution by C++ instructions written by programmer

Allocation of Memory

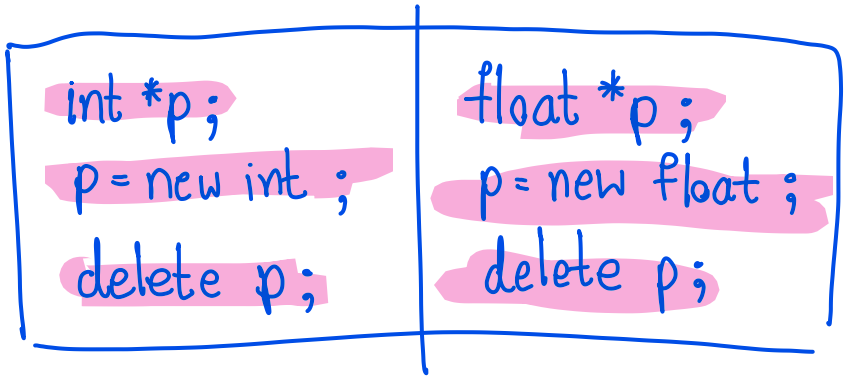
- *Static Allocation*: Allocation of memory space at compile time. — รัน program
- *Dynamic Allocation*: Allocation of memory space at run time. "Heep memory"

Dynamic Memory Allocation

- Dynamic allocation is useful when
  - arrays need to be created whose extent is not known until run time
  - complex structures of unknown size and/or shape need to be constructed as the program runs
  - objects need to be created and the constructor arguments are not known until run time

Dynamic Memory Allocation

- Pointers need to be used for dynamic allocation of memory
- Use the operator **`new`** to dynamically allocate space
- Use the operator **`delete`** to later free this space



The `new` operator

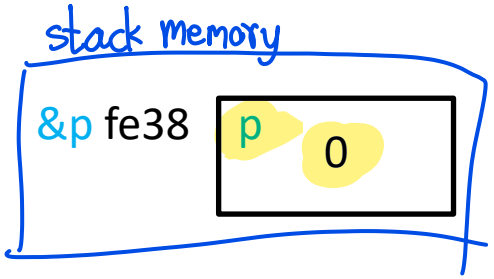
- If memory is available, the **`new`** operator allocates memory space for the requested object/array, and returns a pointer to (address of) the memory allocated.
- If sufficient memory is NOT available, the **`new`** operator returns **`NULL`**.
- The dynamically allocated object/array exists until the **`delete`** operator destroys it.

The `delete` operator

- The **`delete`** operator deallocates the object or array currently pointed to by the pointer which was previously allocated at run-time by the **`new`** operator.
- If the value of the pointer is **`NULL`** there is no effect.

Dynamic Memory Allocation

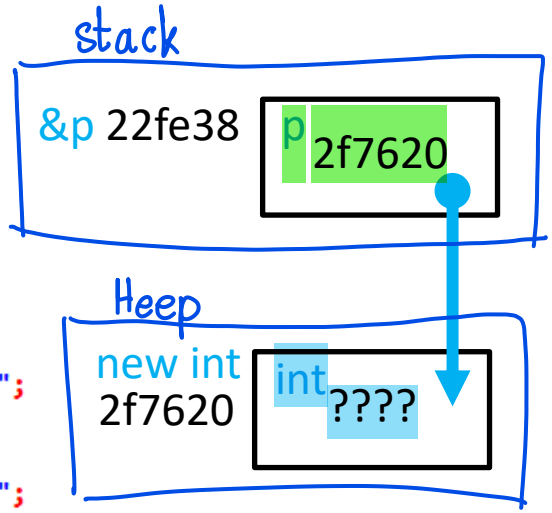
```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5
6      int *p = NULL;
7      cout << "p = " << p << "\t\t&p = " << &p << "\n";
8
9      p = new int;
10     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
11
12     *p = 12;
13     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
14
15     delete p;
16     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
17
18     p = NULL;
19     cout << "p = " << p << "\t\t&p = " << &p ;
20
21     return 0;
22 }
```




```
p = 0                &p = 0x22fe38
p = 0x2f7620         &p = 0x22fe38      *p = 3111712
p = 0x2f7620         &p = 0x22fe38      *p = 12
p = 0x2f7620         &p = 0x22fe38      *p = 3111712
p = 0                &p = 0x22fe38
```

Dynamic Memory Allocation

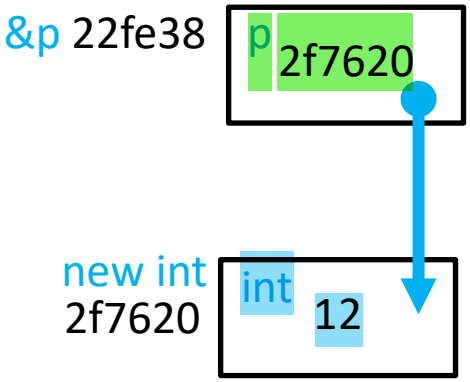
```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5
6      int *p = NULL;
7      cout << "p = " << p << "\t\t&p = " << &p << "\n";
8
9      p = new int;
10     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
11
12     *p = 12;
13     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
14
15     delete p;
16     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
17
18     p = NULL;
19     cout << "p = " << p << "\t\t&p = " << &p ;
20
21     return 0;
22 }
```




```
p = 0               &p = 0x22fe38
p = 0x2f7620        &p = 0x22fe38    *p = 3111712
p = 0x2f7620        &p = 0x22fe38    *p = 12
p = 0x2f7620        &p = 0x22fe38    *p = 3111712
p = 0               &p = 0x22fe38
```

Dynamic Memory Allocation

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5
6      int *p = NULL;
7      cout << "p = " << p << "\t\t&p = " << &p << "\n";
8
9      p = new int;
10     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
11
12     *p = 12;
13     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
14
15     delete p;
16     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
17
18     p = NULL;
19     cout << "p = " << p << "\t\t&p = " << &p ;
20
21     return 0;
22 }
```



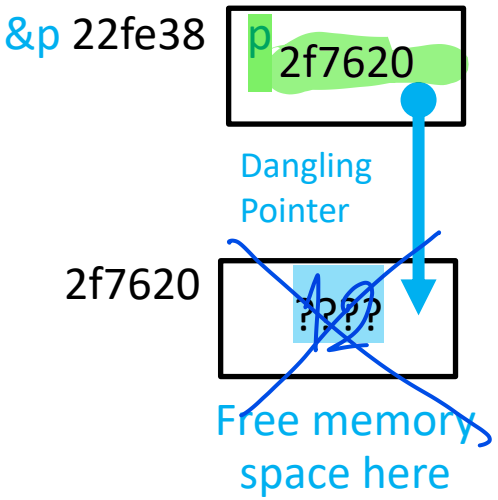

```
p = 0               &p = 0x22fe38
p = 0x2f7620        &p = 0x22fe38    *p = 3111712
p = 0x2f7620        &p = 0x22fe38    *p = 12
p = 0x2f7620        &p = 0x22fe38    *p = 3111712
p = 0               &p = 0x22fe38
```

Dynamic Memory Allocation

```
1   #include<iostream>
2   using namespace std;
3
4   int main(){
5
6       int *p = NULL;
7       cout << "p = " << p << "\t\t&p = " << &p << "\n";
8
9       p = new int;
10      cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
11
12      *p = 12;
13      cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
14
15      delete p;
16      cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
17
18      p = NULL;
19      cout << "p = " << p << "\t\t&p = " << &p ;
20
21      return 0;
22  }
```

เคลียร์ Heep memory

ค่าขยะ

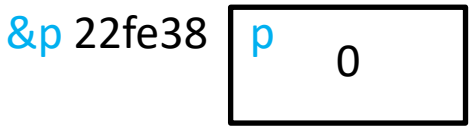


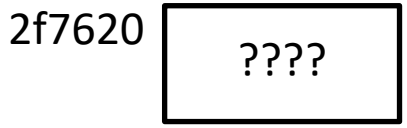

```
p = 0               &p = 0x22fe38
p = 0x2f7620        &p = 0x22fe38    *p = 3111712
p = 0x2f7620        &p = 0x22fe38    *p = 12
p = 0x2f7620        &p = 0x22fe38    *p = 3111712
p = 0               &p = 0x22fe38
```

ค่าขยะ

Dynamic Memory Allocation

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5
6      int *p = NULL;
7      cout << "p = " << p << "\t\t&p = " << &p << "\n";
8
9      p = new int;
10     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
11
12     *p = 12;
13     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
14
15     delete p;
16     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
17
18     p = NULL;
19     cout << "p = " << p << "\t\t&p = " << &p ;
20
21     return 0;
22 }
```






```
p = 0              &p = 0x22fe38
p = 0x2f7620       &p = 0x22fe38    *p = 3111712
p = 0x2f7620       &p = 0x22fe38    *p = 12
p = 0x2f7620       &p = 0x22fe38    *p = 3111712
p = 0              &p = 0x22fe38
```

Dynamic Memory Allocation

```
1   #include<iostream>
2   using namespace std;
3
4   int main(){
5
6       int *p = new int;
7       *p = 12;
8       cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
9
10      int *q = new int;
11      *q = 25;
12      cout << "q = " << q << "\t&q = " << &q << "\t*q = " << *q << "\n";
13
14      delete p,q;
15
16      return 0;
17  }
```

หรือ → `delete p;` `delete q;`

Output

```
p = 0x577620    &p = 0x22fe38    *p = 12
q = 0x577640    &q = 0x22fe30    *q = 25
```

Dynamic Memory Allocation

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5
6     int *p = new int;
7     *p = 12;
8     cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";
9
10    delete p;
11
12    int *q = new int;
13    *q = 25;
14    cout << "q = " << q << "\t&q = " << &q << "\t*q = " << *q << "\n";
15    cout << "*p = " << *p << "\n";
16
17    delete q;
18
19    return 0;
20 }
```

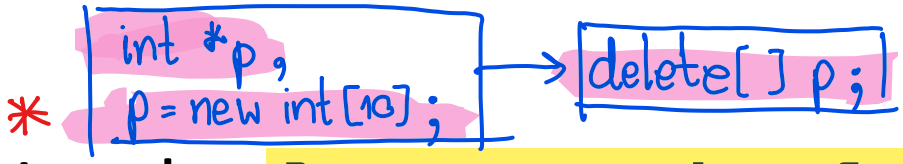
เข้าถึง *p (สิ่งที่ p ชี้ ใน heep)

Output

```
p = 0x2f7620      &p = 0x22fe38     *p = 12
q = 0x2f7620      &q = 0x22fe30     *q = 25
*p = 25
```

← ค่าขยะ ของเอ๊ยได้ 25

Dynamic allocation of Arrays



- Use the **`[array_size]`** on the **`new`** statement to create an array of objects instead of a single instance.
- On the **`delete`** statement use **`[]`** to indicate that an array of objects is to be deallocated.
- **`std::vector`** is the dynamic arrays in C++.

[ไม่ควรทำ]

Array of Runtime Bound

```
Input number of your data: 3
Input your data [1]: 1.2
Input your data [2]: 3.6
Input your data [3]: 6.9
Your data = 1.2 3.6 6.9
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int N;
7      cout << "Input number of your data: ";
8      cin >> N;
9      float c[N];
10     for(int i=0;i<N;i++){
11         cout << "Input your data [" << i+1 << "]: ";
12         cin >> c[i];
13     }
14
15     cout << "Your data = ";
16     for(int i=0;i<N;i++) cout << c[i] << " ";
17
18     return 0;
19 }
```

Array of runtime bound

May be illegal in some complier where array size must be specified with constant variable (**const**)

const int N = 2;

(automatic storage)

Arrays of Runtime Bound = Arrays of Variable Length

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3875.pdf

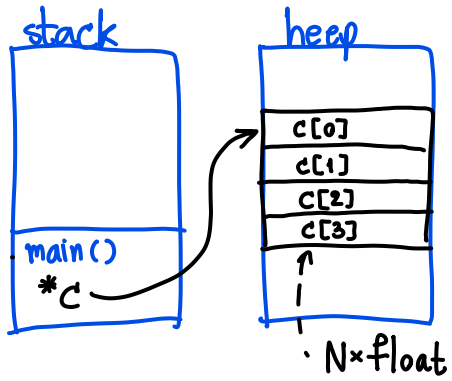
Dynamic allocation of Arrays

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int N;
7       cout << "Input number of your data: ";
8       cin >> N;
9       float *c = new float[N]; //float c[N];
10      for(int i=0;i<N;i++){
11          cout << "Input your data [" << i+1 << "]: ";
12          cin >> c[i];
13      }
14
15      cout << "Your data = ";
16      for(int i=0;i<N;i++) cout << c[i] << " ";
17
18      delete [] c; //Don't forget to free memory
19
20      return 0;
21  }
```

จอง heap ขนาด float × N

c[i] ≡ *(c+i)

```
Input number of your data: 3
Input your data [1]: 1.2
Input your data [2]: 3.6
Input your data [3]: 6.9
Your data = 1.2 3.6 6.9
```



Pointer name can used as array name

Dynamic allocation of 2D-Arrays

- A two dimensional array is really an array of arrays (rows).
- To dynamically declare a two dimensional array of `int` type, you need to declare a pointer to a pointer as:

```
int **matrix;
```

```
int **matrix;
matrix = new int *[r];
for(i=0;i<r;i++)
    matrix[i] = new int[c];
```

Dynamic allocation of 2D-Arrays

- To allocate space for the 2D array with `r` rows and `c` columns:
  - You first allocate the array of pointers which will point to the arrays (rows)

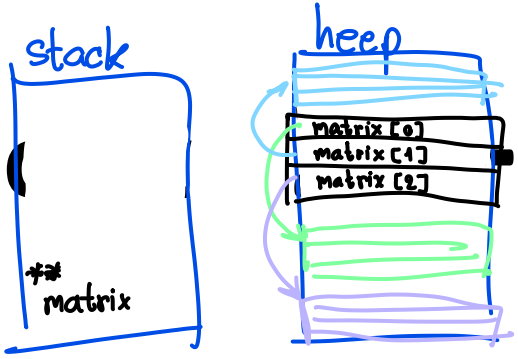
```
matrix = new int*[r];
```

  - This creates space for `r` addresses; each being a pointer to an `int`.
- Then you need to allocate the space for the 1D arrays themselves, each with a size of `c`

```
for(i=0; i<r; i++)
    matrix[i] = new int[c];
```

*(matrix+i)

Dynamic allocation of 2D Arrays



- The elements of the array **`matrix`** now can be accessed by the **`matrix[i][j]`** notation
- Keep in mind, the entire array is not in contiguous space (unlike a static 2D array) array ไม่มีความต่อเนื่องในหน่วยความจำ
- The elements of each row are in contiguous space, but the rows themselves are not.
  - **`matrix[i][j+1]`** is after **`matrix[i][j]`** in memory, but **`matrix[i][0]`** may be before or after **`matrix[i+1][0]`** in memory

(ไม่ควรทำ)

Array of Runtime Bound

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int N,M;
7      cout << "Input size of your matrix: ";
8      cin >> N >> M;
9      float c[N][M];
10     for(int i=0;i<N;i++){
11         cout << "Input row [" << i+1 << "]: ";
12         for(int j=0;j<M;j++){
13         cin >> c[i][j];
14         }
15     }
16
17     return 0;
18 }
```

เก็บในstackทั้งหมด

Array of arrays of runtime bound

May be illegal in some complier where array size must be specified with constant variable (**const**)

```
Input size of your matrix: 2 4
Input row [1]: 1 2 5 8
Input row [2]: 4 9 9 8
```

Dynamic allocation of 2D-Arrays

```
Input size of your matrix: 2 4
Input row [1]: 1 2 5 8
Input row [2]: 4 9 9 8
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int N,M;
7      cout << "Input size of your matrix: ";
8      cin >> N >> M;
9
10     float **c = new float *[N];
11     for(int i = 0; i<N; i++) c[i] = new float[M];
12
13
14     for(int i=0;i<N;i++){
15         cout << "Input row [" << i+1 << "]: ";
16         for(int j=0;j<M;j++){
17         cin >> c[i][j];
18         }
19     }
20
21     for(int i=0; i<N; i++) delete [] c[i];
22     delete [] c;
23
24     return 0;
25 }
```

c เก็บใน stack
สิ่งที่ c ชี้ อยู่ใน heep

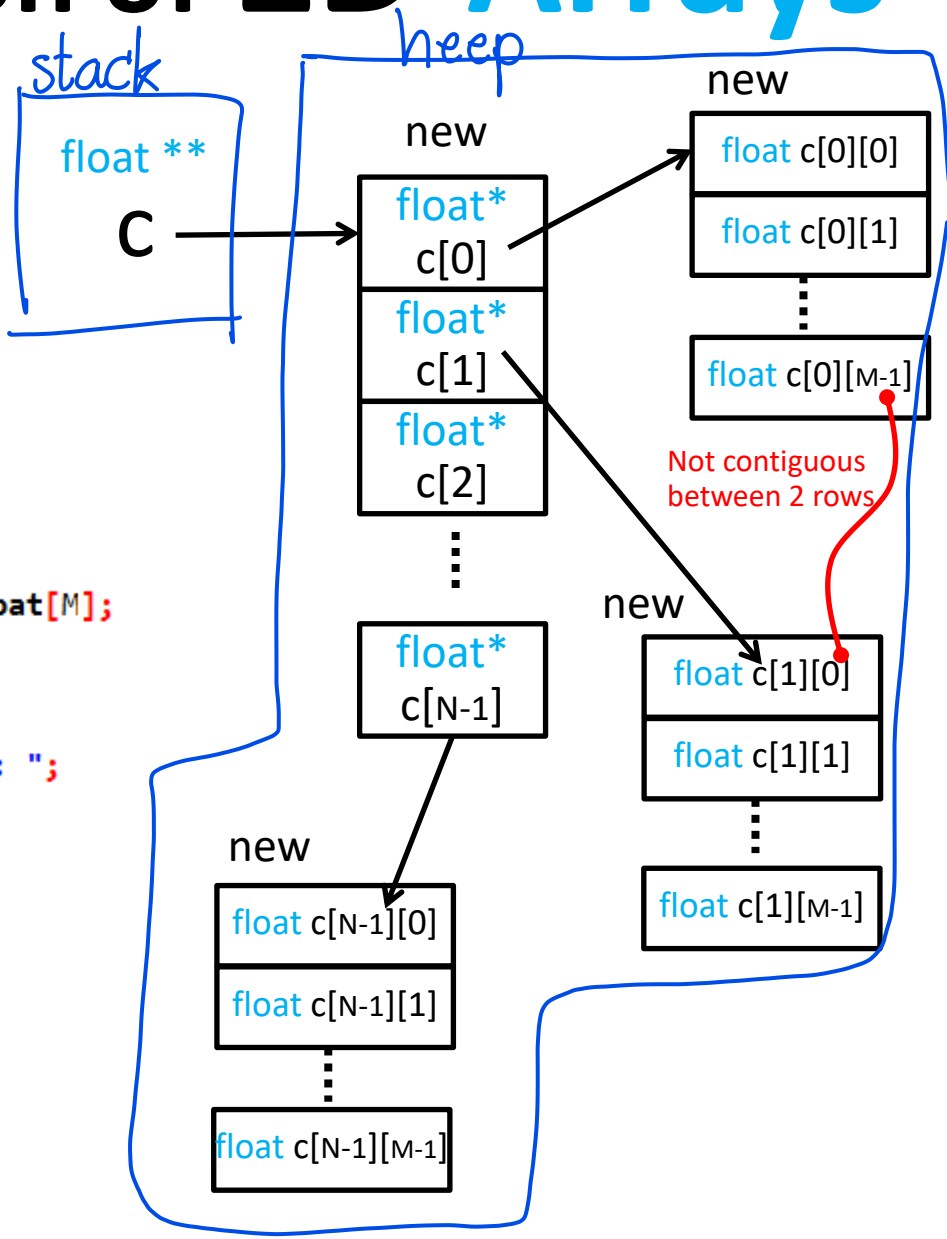
Dynamic Allocation
Equivalent to float c[N][M]

Deallocation of 2D Array

delete N+1 ที่

Dynamic allocation of 2D-Arrays

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int N,M;
7       cout << "Input size of your matrix: ";
8       cin >> N >> M;
9
10      float **c = new float *[N];
11      for(int i = 0; i<N; i++) c[i] = new float[M];
12
13
14      for(int i=0;i<N;i++){
15          cout << "Input row [" << i+1 << "]: ";
16          for(int j=0;j<M;j++){
17          cin >> c[i][j];
18          }
19      }
20
21      for(int i=0; i<N; i++) delete [] c[i];
22      delete [] c;
23
24      return 0;
25  }
```



Dynamic allocation of 2D-Arrays

```
10  float **c = new float *[N];
11  for(int i = 0; i<N; i++) c[i] = new float[M];
12
13
14  for(int i=0;i<N;i++){
15      cout << "Input row [" << i+1 << "]: ";
16      for(int j=0;j<M;j++){
17      cin >> c[i][j];
18      }
19  }
20
21  for(int i=0;i<N;i++){
22      for(int j=0;j<M;j++){
23          cout << &c[i][j] <<" ";
24      }
25      cout << "\n";
26  }
27
28
29  for(int i=0; i<N; i++) delete [] c[i];
30  delete [] c;
```

```
Input size of your matrix: 5 5
Input row [1]: 1 2 3 4 5
Input row [2]: 4 5 5 7 5
Input row [3]: 1 2 3 4 5
Input row [4]: 4 5 6 6 7
Input row [5]: 1 5 7 4 7
0x2f7650 0x2f7654 0x2f7658 0x2f765c 0x2f7660
0x2f7b20 0x2f7b24 0x2f7b28 0x2f7b2c 0x2f7b30
0x2f7b40 0x2f7b44 0x2f7b48 0x2f7b4c 0x2f7b50
0x2f7b60 0x2f7b64 0x2f7b68 0x2f7b6c 0x2f7b70
0x2f7b80 0x2f7b84 0x2f7b88 0x2f7b8c 0x2f7b90
```

Array of Runtime Bound

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

int main()
{
    int N,M;
    cout << "Input size of your matrix: ";
    cin >> N >> M;

    float c[N][M];

    for(int i=0;i<N;i++){
        cout << "Input row [" << i+1 << "]: ";
        for(int j=0;j<M;j++){
        cin >> c[i][j];
        }
    }

    for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
           cout << &c[i][j] <<" ";
        }
        cout << "\n";
    }

    return 0;
}
```

```
Input size of your matrix: 5 5
Input row [1]: 1 2 3 4 5
Input row [2]: 6 7 8 9 10
Input row [3]: 4 5 7 8 3
Input row [4]: 1 1 2 2 2
Input row [5]: 1 4 7 5 5
0x22fd40 0x22fd44 0x22fd48 0x22fd4c 0x22fd50
0x22fd54 0x22fd58 0x22fd5c 0x22fd60 0x22fd64
0x22fd68 0x22fd6c 0x22fd70 0x22fd74 0x22fd78
0x22fd7c 0x22fd80 0x22fd84 0x22fd88 0x22fd8c
0x22fd90 0x22fd94 0x22fd98 0x22fd9c 0x22fda0
```

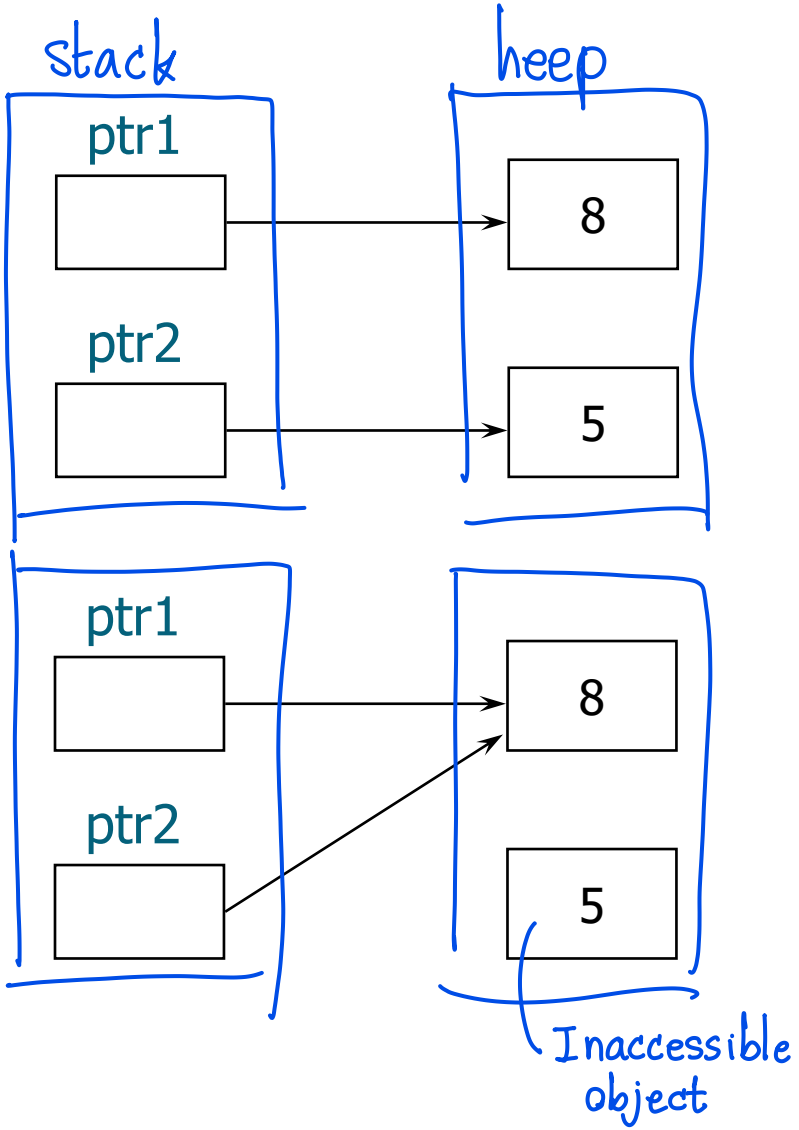
Memory Leaks

- When you dynamically create objects, you can access them through the pointer which is assigned by the `new` operator
- Reassigning a pointer without deleting the memory it pointed to previously is called a memory leak
- It results in loss of available memory space

Memory Leaks

```
int *ptr1 = new int;
int *ptr2 = new int;
*ptr1 = 8;
*ptr2 = 5;
ptr2 = ptr1;
```

How to avoid?



Inaccessible Object

- An inaccessible object is an unnamed object that was created by operator `new` and which a programmer has left without a pointer to it.
- It is a logical error and causes memory leaks.

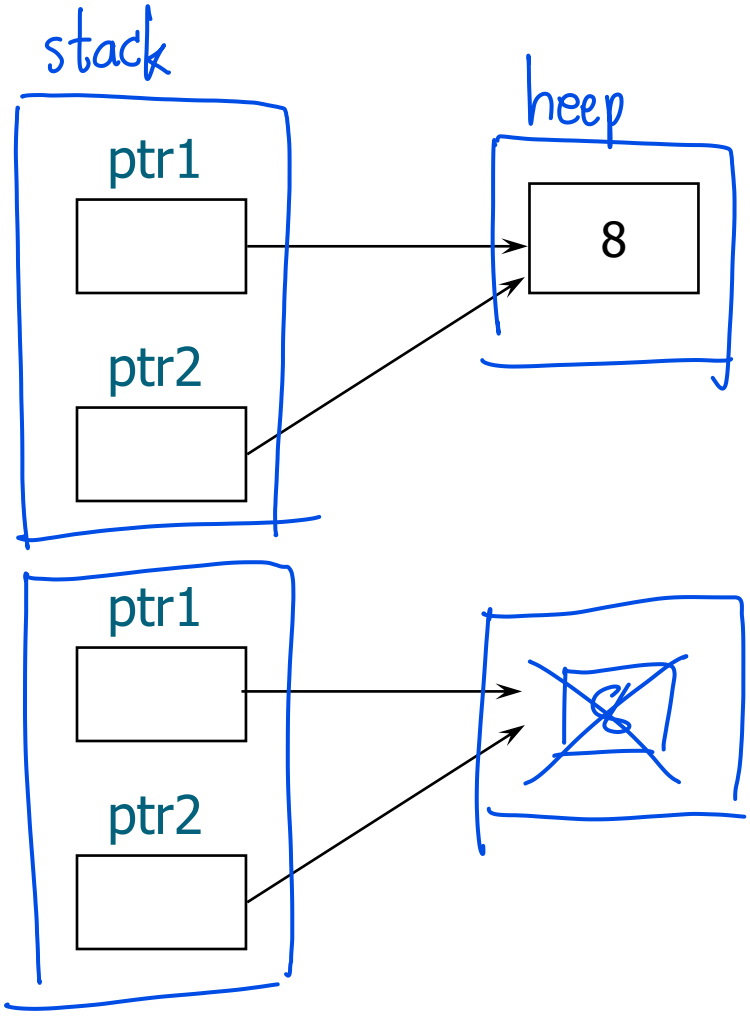
Dangling Pointer

- It is a pointer that points to dynamic memory that has been deallocated.
- The result of dereferencing a dangling pointer is unpredictable.

Dangling Pointer

Dynamic

```
int *ptr1 = new int;
int *ptr2;
*ptr1 = 8;
ptr2 = ptr1;
delete ptr1;
```



How to avoid?

pointer ptr2 ชี้ไปที่ไหน

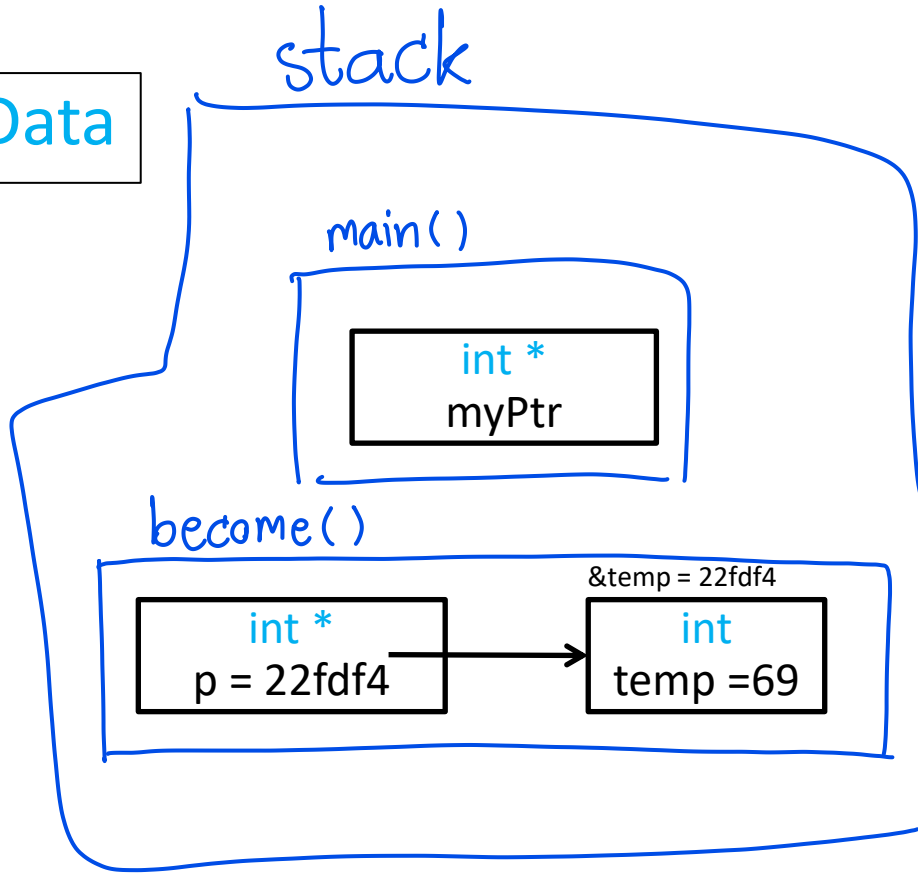
Dangling Pointer

ปัญหาอีกแบบ

```
1   #include <iostream>
2   using namespace std;
3
4   int * become69(){
5       int *p;
6       int temp;
7       p = &temp;
8       temp = 69;
9       cout << "p = " << p << "\n";
10      return p;
11  }
12
13  int main(){
14      int *myPtr;
15      myPtr = become69();
16      cout << "myPtr = " << myPtr << "\n";
17      cout << "*myPtr = " << *myPtr << "\n";
18  }
```

Automatic Data

local

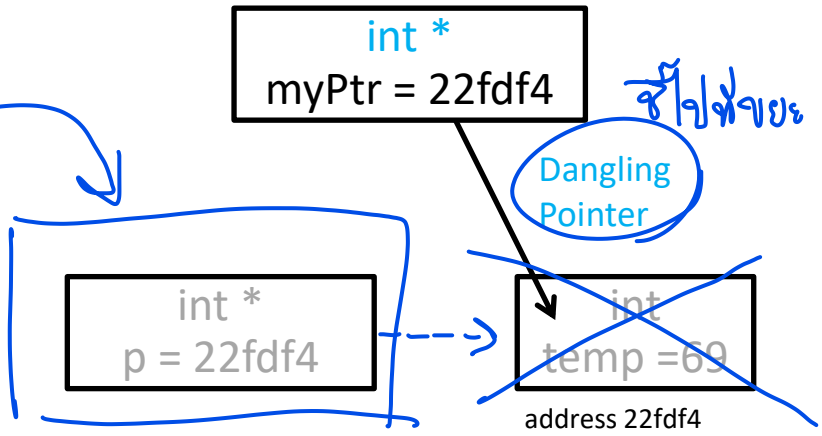



```
p = 0x22fdf4
myPtr = 0x22fdf4
*myPtr = 0
```

Dangling Pointer

```
1   #include <iostream>
2   using namespace std;
3
4   int * become69(){
5       int *p;
6       int temp;
7       p = &temp;
8       temp = 69;
9       cout << "p = " << p << "\n";
10      return p;
11  }
12
13  int main(){
14      int *myPtr;
15      myPtr = become69();
16      cout << "myPtr = " << myPtr << "\n";
17      cout << "*myPtr = " << *myPtr << "\n";
18  }
```

Automatic Data



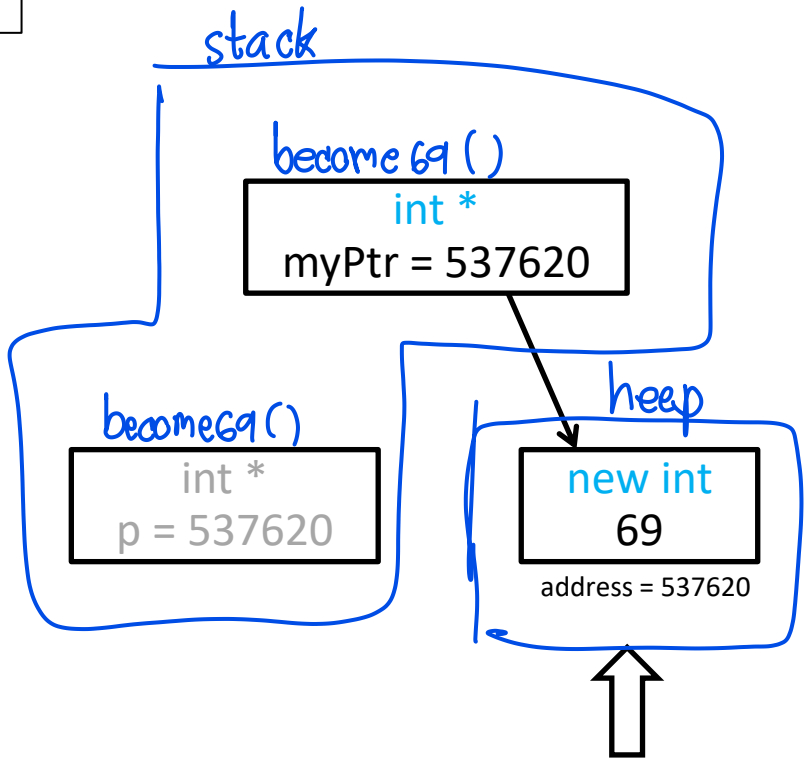
`p` and `temp` are destroyed after leaving function

```
p = 0x22fdf4
myPtr = 0x22fdf4
*myPtr = 0
```

Dangling Pointer

```
1   #include <iostream>
2   using namespace std;
3
4   int * become69(){
5       int *p;
6       p = new int;
7       *p = 69;
8       cout << "p = " << p << "\n";
9       return p;
10  }
11
12  int main(){
13      int *myPtr;
14      myPtr = become69();
15      cout << "myPtr = " << myPtr << "\n";
16      cout << "*myPtr = " << *myPtr << "\n";
17      delete myPtr;
18  }
```

Dynamic Data



Before `delete` operator

```
p = 0x537620
myPtr = 0x537620
*myPtr = 69
```

Dangling Pointer

ถ้าไม่ใช้ new → ใช้ "static"

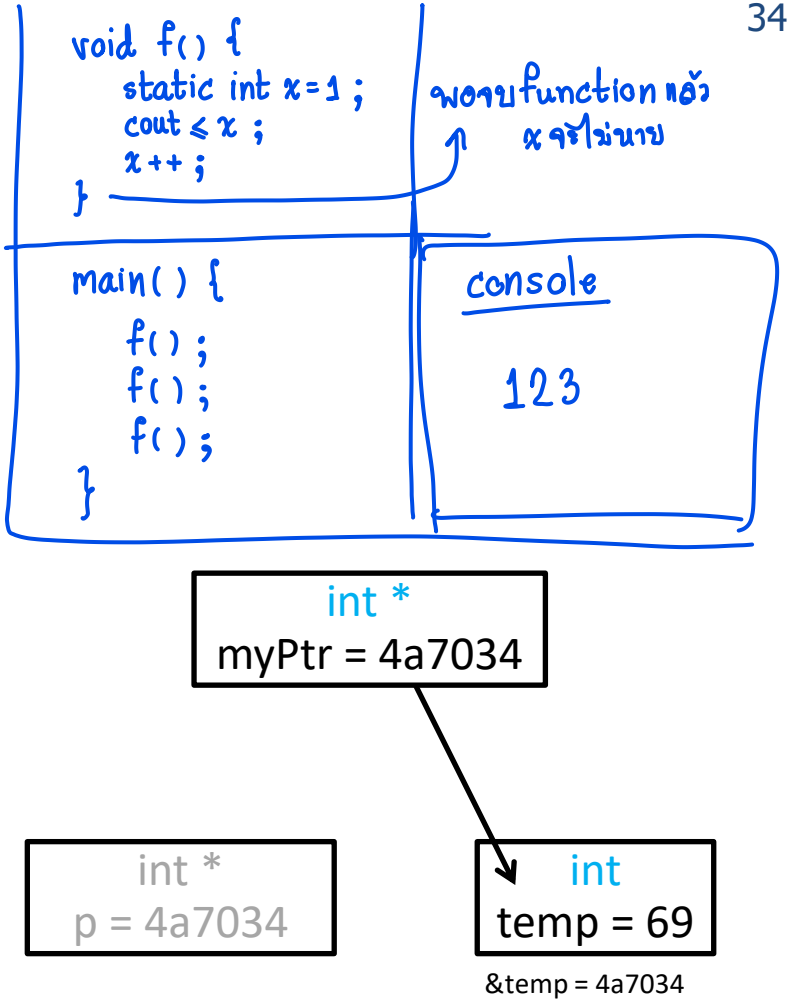
```
1   #include <iostream>
2   using namespace std;
3
4   int * become69(){
5       int *p;
6       static int temp;
7       p = &temp;
8       temp = 69;
9       cout << "p = " << p << "\n";
10      return p;
11  }
12
13  int main(){
14      int *myPtr;
15      myPtr = become69();
16      cout << "myPtr = " << myPtr << "\n";
17      cout << "*myPtr = " << *myPtr << "\n";
18  }
```

Static Data

ค่าเดิมจะไม่หาย

temp ไม่ใช่ local แต่เป็น static

```
p = 0x4a7034
myPtr = 0x4a7034
*myPtr = 69
```



`p` is destroyed after leaving function

`temp` is not destroyed after leaving function

std::Vector

ไลบรารี่ ที่สร้าง array แบบ dynamic ได้

- Vector is a class of sequence container representing array that can change in size.

- Vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements.

- Vectors use a dynamically allocated array to store their elements. Vector need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

http://www.cplusplus.com/reference/vector/vector/

std::Vector

- Vectors = sequence containers representing arrays that can change in size dynamically
- `#include<vector>`
- Declaring a vector:

int

```
std::vector< type > vectorName (vectorSize) ;
std::vector< type > vectorName ;
```

- Referring to an element:

```
vectorName[ index ]
vectorName.at( index )
```

std::Vector

– Adding an element to a vector :

vectorName`.push_back(`*value*`);`

(add new element to the end of vector)

vectorName`.insert(`*position*`,` *value*`);`

(use *vectorName*`.begin()` to obtain position of the 1st element)

– Removing element(s) of a vector :

vectorName`.pop_back();`

(removes the last element in the vector)

vectorName`.erase(`*position*`);`

vectorName`.erase(`*firstPosition*`,`*lastPosition*`);`

vectorName`.clear();`

(removes all elements from the vector)

http://www.cplusplus.com/reference/vector/vector/

std::Vector

– Changing value of an element of a vector :

```
vectorName[ index ] = newValue;
vectorName.at( index ) = newValue;
```

– Vector can be return from a function:

```
vector<type> func(paramenter_list);
```

– Pass a vector to a function:

	By Value	By Reference
Prototype	`type func(vector<type>);`	`type func(vector<type> &);`
Definition	`type func(vector<type> v){` `}`	`type func(vector<type> &v){` `}`
Calling	`func(v)`	`func(v)`

std::Vector

```
0 0 0 1 10 100
17 0 60 1 10 100
```

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main()
6  {
7      int N = 3;
8      vector<int> x(N);
9      x.push_back(1);
10     x.push_back(10);
11     x.push_back(100);
12
13     for(unsigned int i=0; i < x.size();i++) cout << x[i] << " ";
14
15     x[0] = 17;
16     x.at(2) = x.at(4)+x[5]/2;
17
18     cout << "\n";
19     for(unsigned int i=0; i < x.size();i++) cout << x[i] << " ";
20
21     return 0;
22 }
```

std::Vector

myVector.begin() } คล้าย pointer
myVector.end()

cout <<*myVector.begin(); จะพิมพ์สมาชิกตัวแรกใน myVector
cout <<*myVector.end(); จะพิมพ์สมาชิกตัวสุดท้ายใน myVector

```
1   #include<iostream>
2   #include<vector>
3   using namespace std;
4
5   int main(){
6       vector<int> myVector(2);
7       myVector[0] = 4; myVector[1] = 10; // 4 10
8
9       myVector.push_back(55); // 4 10 55
10
11      myVector.insert(myVector.begin()+2,3); // 4 10 3 55
12      myVector.insert(myVector.begin(),8); // 8 4 10 3 55
13
14      myVector.erase(myVector.begin()+1); // 8 10 3 55
15
16      for(int i = 0; i < myVector.size(); i++){
17          cout << &myVector[i] << ": " << myVector[i] << "\n";
18      }
19
20      return 0;
21  }
```

Output

```
0x617bf0: 8
0x617bf4: 10
0x617bf8: 3
0x617bfc: 55
```

std::Vector

Passed by value

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  vector<int> square(vector<int>);
6
7  int main()
8  {
9      vector<int> x;
10     x.push_back(3);
11     x.push_back(-2);
12     x.push_back(7);
13
14     vector<int> y = square(x);
15
16     for(unsigned int i=0; i < y.size();i++) cout << y[i] << " ";
17
18     return 0;
19 }
20
21 vector<int> square(vector<int> v){
22     vector<int> w(v.size());
23     for(unsigned int i=0; i < v.size();i++) w[i] = v[i]*v[i];
24     return w;
25 }
```

Return vector

```
9 4 49
```

std::Vector

```
69 69 69 69 69 69 69
```

```
 1  #include <iostream>
 2  #include <vector>
 3  using namespace std;
 4
 5  void fill69(vector<int> &,int);
 6
 7  int main()
 8  {
 9      vector<int> x;
10
11      fill69(x,7);
12
13      for(unsigned int i=0; i < x.size();i++) cout << x[i] << " ";
14
15      return 0;
16  }
17
18  void fill69(vector<int> &v,int N){
19      for(int i=0; i < N;i++) v.push_back(69);
20  }
```

Passed by reference

261102
Computer Programming

Lecture 19: String and Bitwise Operators

Fundamentals of Characters and Strings

- Character constant
  - Integer value represented as character in single quotes
  - `'z'` is integer value of `z`
    - `122` in ASCII
- String
  - Series of characters treated as single unit
  - Can include letters, digits, special characters `+`, `-`, `*` ...
  - String literal (string constants)
    - Enclosed in double quotes, for example:

      ```
      "I like C++"
      ```
  - Array of characters, ends with null character `'\0'`
  - String is constant pointer
    - Pointer to string's first character
      - Like arrays

ASCII Code

- American Standard Code for Information Interchange

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://simple.wikipedia.org/wiki/ASCII

C-Style String (`char []`)

- String assignment
  - Character array
    - `char color[] = "blue";`
      - Creates 5 element **`char`** array **`color`**
        - » last element is **`'\0'`**
  - Variable of type **`char *`**
    - `char *colorPtr = "blue";`
      - Creates pointer **`colorPtr`** to letter **`b`** in string **“blue”**
        - » **“blue”** somewhere in memory
  - Alternative for character array
    - `char color[] = { ‘b’, ‘l’, ‘u’, ‘e’, ‘\0’ };`

C++ String Class (string)

- #include <string>
- String assignment
  - `string color = "blue";`
- Convert c++ string to c string

```
string cppStr = "Hello";
const char * cStr = cppStr.c_str();
```

return type is const char *

- Convert c string to c++ string

```
char cStr[] = "Hello";
string cppStr = cStr;
```

String Manipulation Functions

	C Style	C++ Style
Data Type	`char [], char *`	`string`
include	`#include <cstring>`	`#include <string>`
ความยาว Length	`strlen(s)` "Happy" ⑤ `size_t strlen(const char *s);`	`s.size();` `size_t size() const;`
Copy (s2 to s1)	`strcpy(s1,s2)` `char *strcpy(char *s1,const char *s2);`	`s1 = s2;` `string& operator= (const string& str);`
Append (s2 to s1)	`strcat(s1,s2)` `char *strcat(char *s1,const char *s2);`	`s1 + s2;` `string operator+ (const string& s1, const string& s2);`
เปรียบเทียบ Compare	`strcmp(s1,s2)` `int strcmp(const char *s1,const char *s2);`	`s1.compare(s2);` `int compare (const string& str) const;`

String Manipulation Functions

	C Style	C++ Style
Element Access	`s[idx]`	`s[idx]` `s.at(idx)`
Find Character (of `s2` in `s1`)	`strpbrk(s1,s2)` `char *strpbrk(char * s1,const char *s2);`	`s1.find_first_of(s2)` `size_t find_first_of(const string& str, size_t pos = 0) const;`
Find Substring	`strstr(s1,s2)` `char *strstr(char * s1,const char *s2);`	`s1.find(s2)` `size_t find(const string& str, size_t pos = 0) const;`
Create Substring		`s1.substr(idx,length)` `string substr (size_t pos = 0, size_t len = npos) const;`
Insert		`s1.insert(idx,s2)` `string& insert (size_t pos, const string& str);`
Erase		`s.erase(idx,length)` `string& erase (size_t pos = 0, size_t len = npos);`

String Manipulation Functions

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

int main(){
    char x[] = "Hello";
    char y[6];
    strcpy(y,x);
    cout << strlen(x) << "\n";
    cout << y << "\n";
    return 0;
}
```

Output

```
5
Hello
```

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

int main(){
    string x = "Hello";
    string y = x;
    cout << x.size() << "\n";
    cout << y << "\n";
    return 0;
}
```

Output

```
5
Hello
```

String Manipulation Functions

```
1  #include<iostream>
2  #include<cstring>
3  using namespace std;
4
5  int main(){
6      char x[] = "Hello";
7      char y[] = "World";
8      strcat(x,y);
9      cout << x << "\n";
10     cout << y << "\n";
11     return 0;
12 }
```

ไม่ได้เอาตัวแปรรับค่าที่ return จาก strcat

Output

```
HelloWorld
World
```

World \0

↑

Hello ~~\0~~ World \0

↑

```
char x[] = "Hello";
Char y[] = "World";
cout << strcat(x,y);
```

output : HelloWorld

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  int main(){
6      string x = "Hello";
7      string y = "World";
8      string z = x+y;
9      cout << x << "\n";
10     cout << y << "\n";
11     cout << z << "\n";
12     return 0;
13 }
```

Output

```
Hello
World
HelloWorld
```

String Manipulation Functions

เปรียบเทียบ

```
1  #include<iostream>
2  #include<cstring>
3  using namespace std;
4
5  int main(){
6      char x[] = "Apple";
7      char y[] = "Banana";
8      char z[] = "Abcdefg";
9      cout << strcmp(x,x) << "\n";
10     cout << strcmp(x,y) << "\n";
11     cout << strcmp(x,z) << "\n";
12     return 0;
13 }
```

Output

```
0
-1
1
```

//zero // เหมือนกันทุกตัว
//negative value // A<B
//positive value // p>b

อยู่ในตารางASCII

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  int main(){
6      string x = "Apple";
7      string y = "Banana";
8      string z = "Abcdefg";
9      cout << (x == x) << "\n";
10     cout << (x == y) << "\n";
11     cout << x.compare(x) << "\n";
12     cout << x.compare(y) << "\n";
13     cout << x.compare(z) << "\n";
14     return 0;
15 }
```

Output

```
1   – จริง
0   – เท็จ
0
-1
1
```

//zero
//negative value
//positive value

String Manipulation Functions

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

int main(){
    char x[] = "May the force be with you";
    char y[] = "aeiou";

    char *z = strpbrk(x,y);
    cout << z << "\n";

    z = strpbrk(z,y);
    cout << z << "\n";

    z = strpbrk(z+1,y);
    cout << z << "\n";

    return 0;
}
```

Output

```
ay the force be with you
ay the force be with you
e force be with you
```

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

int main(){
    string x = "May the force be with you";
    string y = "aeiou";

    int z = x.find_first_of(y);
    cout << z << "\n";

    z = x.find_first_of(y,z+1);
    cout << z << "\n";

    z = x.find_first_of(y,z+1);
    cout << z << "\n";

    return 0;
}
```

Output

```
1
6
9
```

String Manipulation Functions

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

int main(){
    char x[] = "May the force be with you";
    char y[] = "th";

    char *z = strstr(x,y);
    cout << z << "\n";

    z = strstr(z+1,y);
    cout << z << "\n";

    z = strstr(z+1,y);
    cout << z << "\n"; // z = NULL;

    return 0;
}
```

Output

```
the force be with you
th you
null
```

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

int main(){
    string x = "May the force be with you";
    string y = "th";

    int z = x.find(y);
    cout << z << "\n";

    z = x.find(y,z+1);
    cout << z << "\n";

    z = x.find(y,z+1);
    cout << z << "\n";

    return 0;
}
```

Output

```
4
19
-1
```

-1 — ไม่เจอ

String Manipulation Functions

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  int main(){
6      string x = "This is book";
7      cout << x << "\n";
8
9      x.insert(8,"a ");
10     cout << x << "\n"; //"This is a book";
11
12     x.erase(5,5);
13     cout << x << "\n"; //"This book";
14
15     string y = x.substr(5,4);
16     cout << y << "\n";
17
18     return 0;
19 }
```

แทรก "a " ใน x ตำแหน่งที่ 8

ลบ x ตำแหน่งที่ 5 ไป 5 ตัวอักษร

Output

```
This is book
This is a book
This book
book
```

String Manipulation Functions

```
1   #include<iostream>
2   #include<cstring>
3   using namespace std;
4
5   int main(){
6       char x[] = "May the force be with you";
7       char y[strlen(x)+1];
8       strcpy(y,x);
9       char *start =  y;
10      char *end = strpbrk(y," ");
11      while(end){
12          *end = '\0';
13          cout << start << "\n";
14          start = end+1;
15          end = strpbrk(start," ");
16      }
17      cout << start << "\n";
18      cout << x << "\n";
19
20      return 0;
21  }
```

Output

```
May \0
the\0
force \0
be\0
with \0
you
May the force be with you
```

String Manipulation Functions

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

int main(){
    string x = "May the force be with you";
    int start = 0;
    int end = x.find_first_of(" ");
    while(end != -1){
        cout << x.substr(start,end-start) << "\n";
        start = end+1;
        end = x.find_first_of(" ",start);
    }
    cout << x.substr(start,x.size()-start) << "\n";
    cout << x << "\n";

    return 0;
}
```

Output

```
May
the
force
be
with
you
May the force be with you
```

Example 16-A: Love Song

```
1  #include<iostream>
2  #include<fstream>
3  #include<string>
4  #include<cctype>
5  using namespace std;
6
7  int main(){
8      ifstream fin("song.txt");
9      int count = 0;
10     string s;
11     while(getline(fin,s)){
12         int i = 0;
13         while(s[i]){
14             s[i] = toupper(s[i]);
15             i++;
16         }
17
18         int idx = s.find("LOVE");
19         while(idx != -1){
20             count++;
21             idx = s.find("LOVE",idx+4);
22         }
23     }
24     cout << "Love Song Level = " << count;
25     return 0;
26 }
```

แปลงตัวอักษรทั้งหมดให้เป็นตัวใหญ่

song.txt

I feel it in my fingers
I feel it in my toes
Love is all around me
And so the feeling grows

It's written on the wind
It's everywhere I go
So if you really love me
Come on and let it show

You know I love you, I always will
My mind's made up by the way that I feel
There's no beginning, there'll be no end
'Cause on my love you can depend

Output

```
Love Song Level = 4
```

sscanf()

string scan formatted

```
1   #include<iostream>
2   #include<string>
3   using namespace std;
4   int main(){
5       char text[] = "590610999 14.5 15.2";
6       char format[] = "%d %f %f";
7       int ID;
8       float a,b;
9       sscanf(text,format,&ID,&a,&b);
10
11      cout << ID << " = " << a+b;
12
13      return 0;
14  }
```

590610999 = 29.7

sscanf()

```
1   #include<iostream>
2   #include<string>
3   using namespace std;
4   int main(){
5       char text[] = "590610999=[14.5,15.2,10]";
6       char format[] = "%d=[%f,%f,%f]";
7       int ID;
8       float a,b,c;
9       sscanf(text,format,&ID,&a,&b,&c);
10
11      cout << ID << " = " << a+b+c;
12
13      return 0;
14  }
15
```

590610999 = 39.7

sscanf()

```
1   #include<iostream>
2   #include<string>
3   using namespace std;
4   int main(){
5       char text[] = "KarnInwZaa 14.5 15.2 A+";
6       char format[] = "%s %f %f %s";
7       char name[100];
8       char grade[3];
9       float a,b,c;
10      sscanf(text,format,name,&a,&b,grade);
11
12      cout << name << " = " << a+b << " = " << grade;
13
14      return 0;
15  }
16
```

KarnInwZaa = 29.7 = A+

sscanf()

```
1   #include<iostream>
2   #include<string>
3   using namespace std;
4   int main(){
5       char text[] = "Karn (-3-) 14.5 15.2 A+";
6       char format[] = "%s %f %f %s";
7       char name[100];
8       char grade[3];
9       float a,b,c;
10      sscanf(text,format,name,&a,&b,grade);
11
12      cout << name << " = " << a+b << " = " << grade;
13
14      return 0;
15  }
```

sscanf()

```
1   #include<iostream>
2   #include<string>
3   using namespace std;
4   int main(){
5       char text[] = "Karn (-3-):14.5,15.2,A+";
6       char format[] = "%[^:]:%f,%f,%s";
7       char name[100];
8       char grade[3];
9       float a,b,c;
10      sscanf(text,format,name,&a,&b,grade);
11
12      cout << name << " = " << a+b << " = " << grade;
13
14      return 0;
15  }
```

not เอาทุกตัวไปจนกว่าจะเจอ :

Karn (-3-) = 29.7 = A+

sscanf()

function

sscanf

<cstdio>

```
int sscanf ( const char * s, const char * format, ...);
```

Read formatted data from string

Reads data from *s* and stores them according to parameter *format* into the locations given by the additional arguments, as if scanf was used, but reading from *s* instead of the standard input (stdin).

The additional arguments should point to already allocated objects of the type specified by their corresponding format specifier within the *format* string.

Parameters

`s`

C string that the function processes as its source to retrieve the data.

`format`

C string that contains a format string that follows the same specifications as *format* in scanf (see scanf for details).

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a pointer to allocated storage where the interpretation of the extracted characters is stored with the appropriate type.
There should be at least as many of these arguments as the number of values stored by the *format specifiers*.
Additional arguments are ignored by the function.

Return Value

On success, the function returns the number of items in the argument list successfully filled. This count can match the expected number of items or be less (even zero) in the case of a matching failure.
In the case of an input failure before any data could be successfully interpreted, EOF is returned.

http://www.cplusplus.com/reference/cstdio/sscanf/

sscanf()

Examples of format specifiers

%d	Any number of decimal digits (0-9) and a sign (+ or -). จำนวนเต็ม
%f	A series of decimal digits, a decimal point, a sign (+ or -) and the e or E character and a decimal integer
%s	Any number of non-whitespace characters, stopping at the first whitespace character found.
%[*character*]	Any number of the *characters* specified between the brackets.
%[^*character*] (not)	Any number of characters none of them specified as *characters* between the brackets.

%[^abc] คือ เอาทุกตัวไปจนกว่าจะเจอ a,b,c ตัวใดตัวหนึ่ง

http://www.cplusplus.com/reference/cstdio/scanf/

printf() formatted

```c
#include <cstdio>

int main(){
    char *name = "Kasemsit InWJrimJirm";
    int score[] = {69,99,96,66};
    float grade = 3.9999999999;

    printf("[261102] %s:\n-> score = %d, %d, %d, and %d\n-> grade = %.2f",name,score[0],score[1],score[2],score[3],grade);

    return 0;
}
```

[261102] Kasemsit InWJrimJirm:
-> score = 69, 99, 96, and 66
-> grade = 3.70

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

int main(){
    char *name = "Kasemsit InWJrimJirm";
    int score[] = {69,99,96,66};
    float grade = 3.6969696969;

    cout << "[261102] " << name << ":\n-> score = " << score[0] << ", ";
    cout << score[1] << ", " << score[2] << ", and " << score[3];
    cout << "\n-> grade = " << fixed << setprecision(2) << grade;

    return 0;
}
```

↳ทศนิยม 2 ตำแหน่ง

0, 1

Bitwise Operators

- Data represented internally as sequences of bits
  - Each bit can be **0 or 1**
  - 8 bits form a byte (commonly)
    - **`char`** is one byte = 8 bits
    - Other data types larger (**`int`**, **`long`**, etc.)
  - Example:

    unsigned integer **3** (4-byte)

    เลขฐาน 10 (3_{10})

    ฐาน 2 ฐาน 10

    - `00000000 00000000 00000000 00000011`$_2$ = 3_{10}

    char **'3'** (1-byte ASCII)

    - `00110011`$_2$ = 51_{10} → ในตาราง ASCII

      $(1\times2^0)+(1\times2^1)+(0\times2^2)+(0\times2^3)+(1\times2^4)+(1\times2^5)+(0\times2^6)+(0\times2^7)$

      7 6 5 4 3 2 1 0

Bitwise Operators

& (bitwise AND)

- **1** if both bits **1**, **0** otherwise

| (bitwise OR)

- **1** if either bit **1**, **0** otherwise

^ (bitwise Exclusive OR)

- **1** if exactly one bit is **1**, **0** otherwise
- Alternatively: **1** if the bits are different

~ (bitwise one's complement)

- Flips **0** bits to **1**, and vice versa

Bitwise Operators

<< (left shift)

- Moves all bits left by specified amount
- Fills from right with 0
- `1 << SHIFTAMOUNT`

>> (right shift with sign extension)

- Moves bits right by specified amount
- Fill from left can vary

Bitwise Operators

0 = เท็จ
1 = จริง

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

XOR

3: 00000000 00000000 00000000 00000011

6: 00000000 00000000 00000000 00000110

3 & 6: 00000000 00000000 00000000 00000010

3 | 6: 00000000 00000000 00000000 00000111

3 ^ 6: 00000000 00000000 00000000 00000101

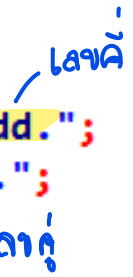
A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

Output

```
2
7
5
```

Bitwise Operators

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main(){
6      int x;
7      cout << "Input x = ";
8      cin >> x;
9
10     if(x&1) cout << "x is odd.";
11     else cout << "x is even.";
12
13     return 0;
14 }
```



Output

Input x = 69
x is odd.

Input x = 12
x is even.

Bitwise Operators

110_2 111_2 000_2 001_2 010_2

-2_{10} -1_{10} 0_{10} 1_{10} 2_{10}

```
1   #include<iostream>
2
3
4   using namespace std;
5
6   int main(){
7
8       cout << (~0)<< "\n"; // signed int
9       cout << (~0u)<< "\n"; // unsigned int
10
11      cout << (~1)<< "\n"; // signed int
12      cout << (~1u)<< "\n"; // unsigned int
13
14      return 0;
15  }
```

int ±

ค่าสูงสุด

0u: 00000000 00000000 00000000 00000000

~0u : 11111111 11111111 11111111 11111111

signed int = -1

unsigned int = 4294967295

1u: 00000000 00000000 00000000 00000001

~1u : 11111111 11111111 11111111 11111110

signed int = -2

unsigned int = 4294967294

Output

```
-1
4294967295
-2
4294967294
```

Bitwise Operators

≪ leftshift

```cpp
#include<iostream>


using namespace std;

int main(){
    int a = 7;
    int b = a << 1;
    int c = a << 2;
    int d = a << 3;

    cout << a << "\n";
    cout << b << "\n";
    cout << c << "\n";
    cout << d << "\n";

    return 0;
}
```

Output

```
7
14
28
56
```

a: 00000000 00000000 00000000 00000111

$$2^2 + 2^1 + 2^0 = 7$$

×2

b: 00000000 00000000 00000000 00001110

$$2^3 + 2^2 + 2^1 = 2 \times \left(2^2 + 2^1 + 2^0\right)$$

×4

c: 00000000 00000000 00000000 00011100

$$2^4 + 2^3 + 2^2 = 4 \times \left(2^2 + 2^1 + 2^0\right)$$

×8

d: 00000000 00000000 00000000 00111000

$$2^5 + 2^4 + 2^3 = 2^3 \times \left(2^2 + 2^1 + 2^0\right)$$

Bitwise Operators

» rightshift

```
1  #include<iostream>
2
3
4  using namespace std;
5
6  int main(){
7      int a = 7;
8      int b = a >> 1;
9      int c = a >> 2;
10     int d = a >> 3;
11
12     cout << a << "\n";
13     cout << b << "\n";
14     cout << c << "\n";
15     cout << d << "\n";
16
17     return 0;
18 }
```

Output

```
7
3
1
0
```

a: 00000000 00000000 00000000 00000111

b: 00000000 00000000 00000000 00000011

c: 00000000 00000000 00000000 00000001

d: 00000000 00000000 00000000 00000000

Bitwise Operators

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

template <typename T>
void showbinary(T x){
    unsigned int mask = 1 << (sizeof(x)*8-1);
    for(unsigned int i = 0; i < sizeof(x)*8; i++){

        if(x&mask) cout << 1;
        else cout << 0;

        mask = mask >> 1;
    }
}
```

```cpp
int main(){
    int a = 1034;
    cout << "1034 = ";
    showbinary(a);

    int b = 1035;
    cout << "\n1035 = ";
    showbinary(b);

    int c = -3;
    cout << "\n-3 = ";
    showbinary(c);

    int d = -4;
    cout << "\n-4 = ";
    showbinary(d);

    char e = 'E';
    cout << "\nE = ";
    showbinary(e);

    char f = 'F';
    cout << "\nF = ";
    showbinary(f);

    return 0;
}
```

Output

```
1034 = 00000000000000000000010000001010
1035 = 00000000000000000000010000001011
-3 = 11111111111111111111111111111101
-4 = 11111111111111111111111111111100
E = 01000101
F = 01000110
```

261102
Computer Programming

Lecture 20: Structures

Structure

คอลเลคชั่นของตัวแปร

- A collection of one or more variables, typically of different types, grouped together under a single name for convenient handling

- Known as **`struct`** in *C* and *C++*

Structure Definition

Defines a new *type*

Name of the type

(name of type is optional if you are just declaring a single)

```
struct Teacher{
   string name;
   int age;
   float height;
   float weight;
   bool handsome;
};
```

Members of the **`struct`**

Declaring `struct` variables

Structure variables declared like variables of other types

ชนิดตัวแปร ชื่อ Teacher / ประเภท

```
Teacher k,s,p;
```
(Declares three variables – k, s, and p – each of type struct Teacher)

```
Teacher allteacher[3];
```
(Declares a 3-element array of struct Teacher)

```
Teacher *tPtr = &k;
```
(Declares a pointer to an object of type struct Teacher)

```
Teacher &tRef = k;
```
(Declares a reference variable of type struct Teacher)

Initialization of `struct` variables

Structure Initialization

```
Teacher k = {"Karn",18,174.5,69.69,true};
Teacher s = {"Santi",40,150.0,50.0,false};
Teacher p = {"Patiwet",30,169,99.99,false};
```

Structure Definition

```
struct Teacher{
    string name;
    int age;
    float height;
    float weight;
    bool handsome;
};
```

Accessing Members of a `struct`

- Member access operators
  - (1) – Dot operator (.) for structure and class members
  - (2) – Arrow operator (->) for structure and class members via pointer to object
  - – Print member **`name`** of **`Teacher k`**:

```
cout << k.name;
```

OR

```
Teacher *p = &k;
cout << p->name;
```

*p = k

(3) – **`p->name`** same as **`(*p).name`**

- Parentheses required
  - – * lower precedence than .

Accessing Members of a `struct`

```cpp
#include<iostream>
#include<string>

using namespace std;

struct Teacher{
    string name;
    int age;
    float height;
    float weight;
    bool handsome;
};

int main(){

    Teacher k = {"Karn",18,174.5,69.69,true};

    cout << k.name;
    if(k.handsome) cout << " is very handsome.\n";

    cout << "AGE = " << k.age << "\n";

    cout << "BMI = " << 10000*k.weight/k.height/k.height << "\n";

    return 0;
}
```

Output

```
Karn is very handsome.
AGE = 18
BMI = 22.8865
```

Accessing Members of a `struct`

```
14  int main(){
15
16      Teacher k = {"Karn",18,174.5,69.69,true};
17      Teacher *p = &k;
18
19      cout << (*p).name;
20      if((*p).handsome) cout << " is very handsome.\n";
21
22      cout << "AGE = " << (*p).age << "\n";
23
24      cout << "BMI = " << 10000*(*p).weight/(*p).height/(*p).height << "\n";
25
26      return 0;
27  }
```

Output

```
Karn is very handsome.
AGE = 18
BMI = 22.8865
```

```
14  int main(){
15
16      Teacher k = {"Karn",18,174.5,69.69,true};
17      Teacher *p = &k;
18
19      cout << p->name;
20      if(p->handsome) cout << " is very handsome.\n";
21
22      cout << "AGE = " << p->age << "\n";
23
24      cout << "BMI = " << 10000*p->weight/p->height/p->height << "\n";
25
26      return 0;
27  }
```

Accessing Members of a `struct`

```
14  int main(){
15
16      Teacher *p = new Teacher;
17      p->name = "Karn";
18      p->age = 18;
19      p->height = 174.5;
20      p->weight = 69.69;
21      p->handsome = true;
22
23      cout << p->name;
24      if(p->handsome) cout << " is very handsome.\n";
25
26      cout << "AGE = " << p->age << "\n";
27
28      cout << "BMI = " << 10000*p->weight/p->height/p->height << "\n";
29
30      delete p;
31
32      return 0;
33  }
```

heep

Output

```
Karn is very handsome.
AGE = 18
BMI = 22.8865
```

Dynamic Allocation

Members of a `struct`

```
1  #include<iostream>
2  #include<string>
3
4  using namespace std;
5
6  struct Student{
7      int id;
8      double score[3];
9      string grade;
10 };
11
12 int main(){
13
14     Student x = {590610999,{20.5,25.6,36.7},"B+"};
15     cout << "ID = " << x.id;
16     cout << "\nScore = " << x.score[0] << " " << x.score[1] << " " << x.score[2] ;
17     cout << "\nGrade = " << x.grade;
18
19     return 0;
20 }
```

⇦ Array

Initialization

Access elements

Output

```
ID = 590610999
Score = 20.5 25.6 36.7
Grade = B+
```

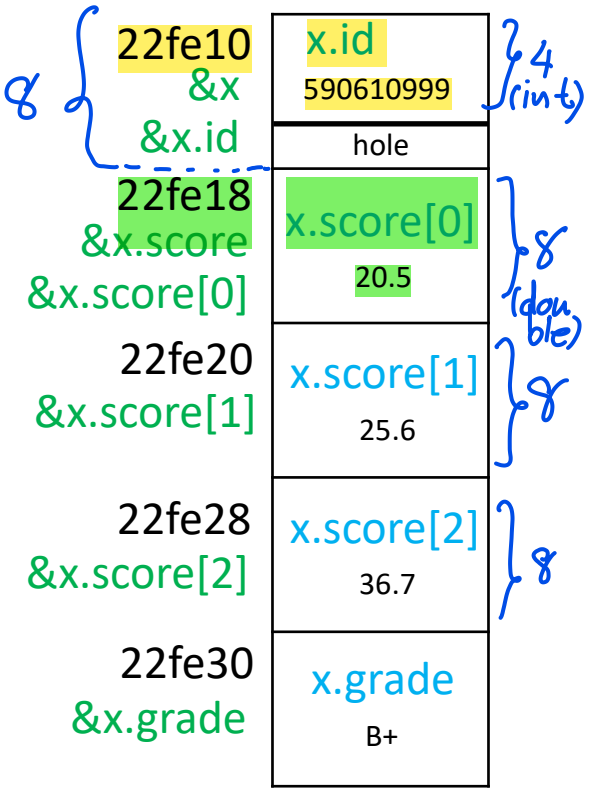
Members of a `struct`

int ID　　double score[3]　　string grade

```
12  int main(){
13
14      Student x = {590610999,{20.5,25.6,36.7},"B+"};
15      cout << "&x = " << &x;
16      cout << "\n&x.id = " << &x.id;
17      cout << "\n&x.sccore = " << &x.score;
18      cout << "\n&x.sccore[0] = " << &x.score[0];
19      cout << "\n&x.sccore[1] = " << &x.score[1];
20      cout << "\n&x.sccore[2] = " << &x.score[2];
21      cout << "\n&x.grade = " << &x.grade;
22
23      return 0;
24  }
```

Output

```
&x = 0x22fe10
&x.id = 0x22fe10
&x.sccore = 0x22fe18
&x.sccore[0] = 0x22fe18
&x.sccore[1] = 0x22fe20
&x.sccore[2] = 0x22fe28
&x.grade = 0x22fe30
```



Structure may not be in consecutive bytes of memory

Members of a `struct`

```
1   #include<iostream>
2   #include<string>
3
4   using namespace std;
5
6   struct Student{
7       int id;
8       int N;
9       double score[N];
10      string grade;
11  };
```

Illegal to use variable array size

```
1   #include<iostream>
2   #include<string>
3
4   using namespace std;
5
6   struct Student{
7       int id;
8       int N;
9       double score[99];
10      string grade;
11  };
12
13  int main(){
14      Student x,y;
15      x.N = 2;
16      x.score[0] = 55.55;
17      x.score[1] = 69.69;
18
19      y.N = 3;
20      y.score[0] = 12.34;
21      y.score[1] = 23.45;
22      y.score[2] = 34.56;
23
24      return 0;
25  }
```

กำหนดไว้เลย

Over-size declaration

Members of a `struct`

```cpp
#include<iostream>
#include<string>
#include<vector>

using namespace std;

struct Student{
    int id;
    vector<double> score;
    string grade;
};

int main(){

    Student x;
    x.score.push_back(55.55);
    x.score.push_back(69.69);

    cout << x.score[0] << "\n";
    cout << x.score[1] << "\n";
    cout << x.score.size() << "\n";

    return 0;
}
```

Use vector as member

```cpp
#include<iostream>
#include<string>

using namespace std;

struct Student{
    int id;
    int N;
    double *score;
    string grade;
};

int main(){

    Student x,y;
    x.N = 2;
    x.score = new double[x.N];
    x.score[0] = 55.55;
    x.score[1] = 69.69;

    y.N = 1;
    y.score = new double[y.N];
    y.score[0] = 12.34;
    delete [] x.score;
    return 0;
}
delete [] y.score;
```

Use pointer as member

(point to dynamic array)

Members of a `struct`

Members of `struct` can be `struct` of other type

```
1   #include<iostream>
2   #include<string>
3
4   using namespace std;
5
6   struct People{
7       string name;
8       int age;
9       char sex;
10      string nationality;
11  };
12
13  struct Room{
14      string name;
15      string type;
16      int rate;
17      int capacity;
18      People guest;
19  };
20
21  int main(){
22      People x = {"Prayath",69,'M',"Thai"};
23      Room y = {"4-402","Standard",690,2,x};
24
25      cout << y.guest.name << " is staying at room " << y.name << ".";
26
27      return 0;
28  }
```

People

Room

Room y ;

y.name = name of the Room

y.guest.name = name of the People

Output

Prayath is staying at room 4-402.

Members of a `struct`

```
1  #include<iostream>
2  #include<string>
3
4  using namespace std;
5
6
7  struct People{
8      string name;
9      int age;
10     People father;
11     People mother;
12 };
```

ยังไม่รู้ขนาด

This is illegal.

```
1  #include<iostream>
2  #include<string>
3
4  using namespace std;
5
6  struct People{
7      string name;
8      int age;
9      People *father;
10     People *mother;
11 };
12
13 int main(){
14     People x = {"Peter",23};
15     People y = {"Sara",53};
16     People z = {"John",60};
17     x.father = &z;
18     x.mother = &y;
19
20     cout << x.father->name << " is " << x.name << "'s father.\n";
21     cout << x.mother->name << " is " << x.name << "'s mother.\n";
22
23     return 0;
24 }
```

8 bytes (ขนาดfix size)

Output

```
John is Peter's father.
Sara is Peter's mother.
```

Members of a `struct`

Room not decare

This is illegal.

```
1   #include<iostream>
2   #include<string>
3
4   using namespace std;
5
6   struct People{
7       string name;
8       int age;
9       char sex;
10      string nationality;
11      Room stayin;
12  };
13
14  struct Room{
15      string name;
16      string type;
17      int rate;
18      int capacity;
19      People guest;
20  };
```

Array of struct

```cpp
#include<iostream>
#include<string>

using namespace std;

struct People{
    string name;
    int age;
    char sex;
    string nationality;
};

int main(){
    People x = {"Prayath",69,'M',"Thai"};
    People y[3];

    y[0] = x;

    y[1].name = "Tukkie"; y[1].age = 55; y[1].sex = 'F'; y[1].nationality = "Thai";

    y[2] = y[1];
    y[2].name = "Somsri";

    cout << y[0].name << " " << y[0].age << " " << y[0].sex << " " << y[0].nationality << "\n";
    cout << y[1].name << " " << y[1].age << " " << y[1].sex << " " << y[1].nationality << "\n";
    cout << y[2].name << " " << y[2].age << " " << y[2].sex << " " << y[2].nationality << "\n";

    return 0;
}
```

Output

y[0]	Prayath 69 M Thai
y[1]	Tukkie 55 F Thai
y[2]	Somsri 55 F Thai

Vector

~~Array~~ of struct

```cpp
#include<iostream>
#include<string>
#include<vector>

using namespace std;

struct People{
    string name;
    int age;
    char sex;
    string nationality;
};

int main(){

    vector<People> y;
    y.push_back({"Prayath",69,'M',"Thai"});
    People x = {"Tukkie",55,'F',"Thai"};
    y.push_back(x);

    cout << y[0].name << " " << y[0].age << " " << y[0].sex << " " << y[0].nationality << "\n";
    cout << y[1].name << " " << y[1].age << " " << y[1].sex << " " << y[1].nationality << "\n";

    People *p = &y[0];
    cout << p->name << " " << p->age << " " << p->sex << " " << p->nationality << "\n";

    return 0;
}
```

vector เก็บ y

y[1] คือ x

p -> name ⇒ Prayath

(p+1) -> name ⇒ Tukkie

Array of `struct`

ใน Room มี Array ของ People อยู่

```cpp
#include<iostream>
#include<string>

using namespace std;

struct People{
    string name;
    int age;
    char sex;
    string nationality;
};

struct Room{
    string name;
    string type;
    int rate;
    int capacity;
    People guest[5];
};

int main(){
    People a = {"Prayath",69,'M',"Thai"};
    People b = {"Tukkie",55,'F',"Thai"};
    Room y = {"4-6969","Standard",690,2,a,b};

    cout << y.guest[0].name << " is staying at room "
    << y.name << " with " << y.guest[1].name << ".";

    return 0;
}
```

Room People ตัวที่ 1 name ใน People

Output

```
Prayath is staying at room 4-6969 with Tukkie.
```

Array of `struct`

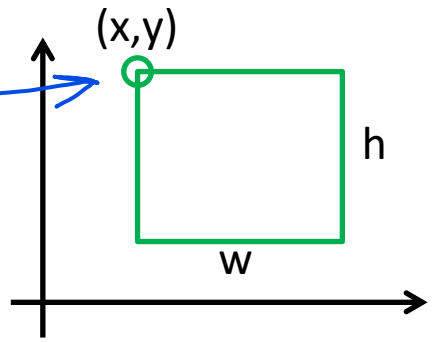
Dynamic allocation

```cpp
#include<iostream>
#include<string>

using namespace std;

struct People{
    string name;
    int age;
    char sex;
    string nationality;
};

struct Room{
    string name;
    string type;
    int rate;
    int capacity;
    People *guest;
};

int main(){
    People *p = new People[2];
    p[0].name = "Prayath"; p[0].age = 69; p[0].sex = 'M'; p[0].nationality = "Thai";
    p[1].name = "Tukkie"; p[1].age = 55; p[1].sex = 'F'; p[1].nationality = "Thai";
    Room y = {"4-6969","Standard",690,2,p};

    cout << y.guest[0].name << " is staying at room "
    << y.name << " with " << y.guest[1].name << ".";
    delete [] p;
    return 0;
}
```

Output

```
Prayath is staying at room 4-6969 with Tukkie.
```

Function of `struct`



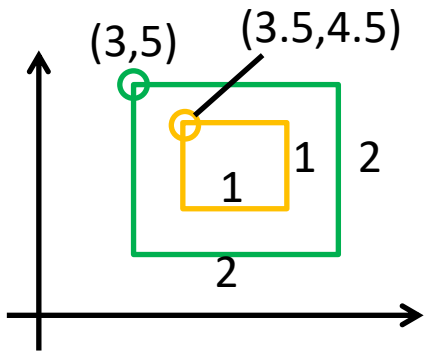

```
1   #include<iostream>
2   using namespace std;
3
4   struct Rect{
5       double x,y,w,h;
6   };
7
8   double findarea(Rect R){
9       return R.w*R.h;
10  }
11
12  int main(){
13      Rect R1 = {3,5,2,2};
14      Rect R2 = {10,8,4,3};
15
16      cout << findarea(R1) << " " << findarea(R2);
17      return 0;
18  }
```

Pass `struct` by value

Output

```
4 12
```

Function of `struct`




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

struct Rect{
    double x,y,w,h;
};

Rect resize(Rect R, double s){
    Rect out;
    out.w = R.w*s;
    out.h = R.h*s;
    out.x = R.x+R.w/2-out.w/2;
    out.y = R.y-R.h/2+out.h/2;

    return out;
}

int main(){
    Rect R1 = {3,5,2,2};
    Rect R2 = resize(R1,0.5);

    cout << R2.x << " " << R2.y << " " << R2.w << " " << R2.h;
    return 0;
}
```

Pass by value and return `struct`

Output

```
3.5 4.5 1 1
```

Function of `struct`

```
1  #include<iostream>
2  using namespace std;
3
4  struct Rect{
5      double x,y,w,h;
6  };
7
8  void resize(Rect &R, double s){
9      R.x = R.x+R.w/2-R.w*s/2;
10     R.y = R.y-R.h/2+R.h*s/2;
11     R.w *= s;
12     R.h *= s;
13 }
14
15 int main(){
16     Rect R1 = {3,5,2,2};
17     resize(R1,0.5);
18
19     cout << R1.x << " " << R1.y << " " << R1.w << " " << R1.h;
20     return 0;
21 }
```

Pass by reference argument

Output

3.5 4.5 1 1

Function of `struct`

```
1  #include<iostream>
2  using namespace std;
3
4  struct Rect{
5      double x,y,w,h;
6  };
7
8  void resize(Rect *R, double s){
9      R->x = R->x+R->w/2-R->w*s/2;
10     R->y = R->y-R->h/2+R->h*s/2;
11     R->w *= s;
12     R->h *= s;
13 }
14
15 int main(){
16     Rect R1 = {3,5,2,2};
17     resize(&R1,0.5);
18
19     cout << R1.x << " " << R1.y << " " << R1.w << " " << R1.h;
20     return 0;
21 }
```

Pass by pointer argument

Output

3.5 4.5 1 1

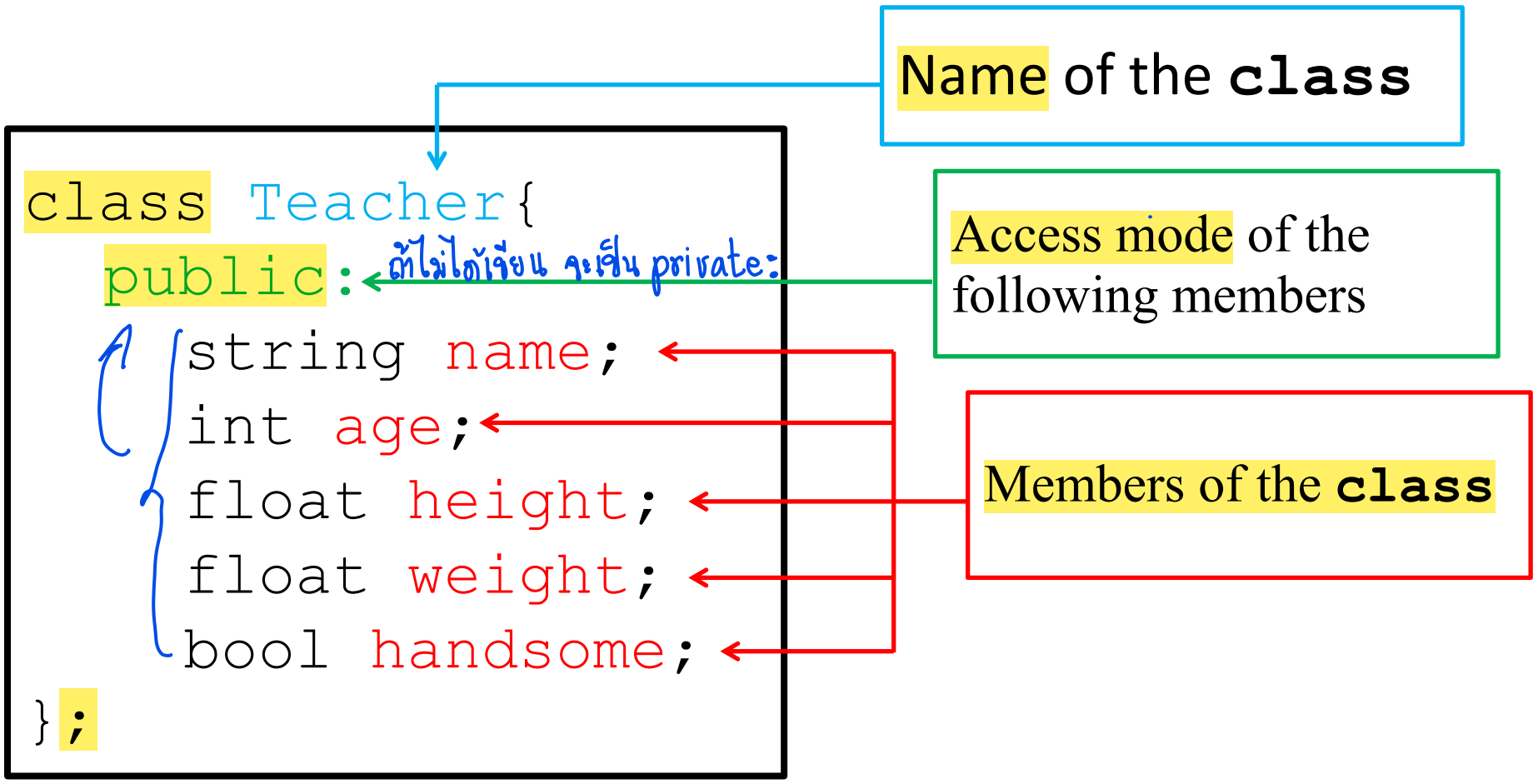
261102
Computer Programming

Lecture 21: Classes I

Class

- Model objects
  - Attributes (data members) ข้อมูล
  - Behaviors (member functions) พฤติกรรม
- Defined using keyword **`class`**
- Controlling Access to Members
  - **`private`** เข้าได้แค่ใน class/structure นั้น : int main เข้าสั่งไม่ได้
    - Default access mode — class ~ ส่วนใหญ่เป็น private
    - Accessible to member functions and **`friend`**s
  - **`public`** เข้าสั่งได้หมด — struct ~ ส่วนใหญ่เป็น public
    - Accessible to any function in program with handle to class object
  - **`protected`** ยังไม่เรียน
    - Inheritance

Class Definition



Declaring `class` objects

Class objects declared like variables of other types

```
Teacher k,s,p;
```
(Declares three objects – k, s, and p – each of class Teacher)

```
Teacher allteacher[3];
```
(Declares a 3-element array of class Teacher)

```
Teacher *tPtr = &k;
```
(Declares a pointer to an object of class Teacher)

```
Teacher &tRef = k;
```
(Declares a reference variable of class Teacher)

Accessing Members of a `class`

- Identical to those for **`struct`**
- Dot member selection operator (**`.`**)
  - Object
  - Reference to object
- Arrow member selection operator (**`->`**)
  - Pointers

`class` member access

Default **`private`**

Explicitly set to **`private`**, **`public`**, **`protected`**

`struct` member access

Default **`public`**

Explicitly set to **`private`**, **`public`**, **`protected`**

Data Members

```cpp
#include<iostream>
#include<string>

using namespace std;

struct Teacher{
    string name;
    int age;
    float height;
    float weight;
    bool handsome;
};

int main(){

    Teacher k = {"Karn",18,174.5,69.69,true};

    cout << k.name;
    if(k.handsome) cout << " is very handsome.\n";

    cout << "AGE = " << k.age << "\n";

    cout << "BMI = " << 10000*k.weight/k.height/k.height << "\n";

    return 0;
}
```

`struct` can be replaced by **`class`**.

```cpp
class Teacher{
  public:
    string name;
    int age;
    float height;
    float weight;
    bool handsome;
};
```

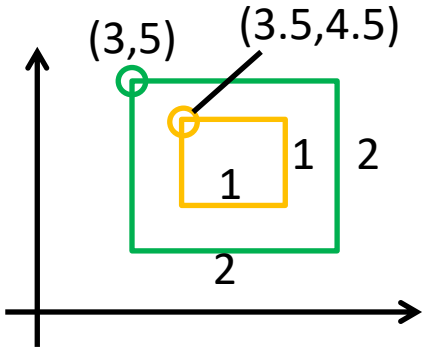
ถ้าเป็น private int main เข้าถึงไม่ได้

Output

```
Karn is very handsome.
AGE = 18
BMI = 22.8865
```

Member Functions

```
1   #include<iostream>
2   using namespace std;
3
4   struct Rect{
5       double x,y,w,h;
6   };
7
8   void resize(Rect &R, double s){
9       R.x = R.x+R.w/2-R.w*s/2;
10      R.y = R.y-R.h/2+R.h*s/2;
11      R.w *= s;
12      R.h *= s;
13  }
14
15  int main(){
16      Rect R1 = {3,5,2,2};
17      resize(R1,0.5);
18
19      cout << R1.x << " " << R1.y << " " << R1.w << " " << R1.h;
20      return 0;
21  }
```



A function related to struct **`Rect`**

Output

```
3.5 4.5 1 1
```

Member Functions

```
1   #include<iostream>
2   using namespace std;
3
4   class Rect{
5       public:
6           double x,y,w,h;
7
8           void resize(double s){
9               x += w*(1-s)/2;
10              y += w*(s-1)/2;
11              w *= s;
12              h *= s;
13          }
14
15  };
16
17  int main(){
18      Rect R1 = {3,5,2,2};
19
20      R1.resize(0.5);
21
22      cout << R1.x << " " << R1.y << " " << R1.w << " " << R1.h;
23
24      return 0;
25  }
```

Output

```
3.5 4.5 1 1
```

Data members
(Attributes of the class)
ข้อมูล

Member function
(Behavior of the class)
พฤติกรรม

Declare class object and initialize data members

Call member function (this function has one input)

Access public data members

Member Functions

```
1  #include<iostream>
2  using namespace std;
3
4  class Rect{
5      public:
6          double x,y,w,h;
7
8          void resize(double s){
9              x += w*(1-s)/2;
10             y += w*(s-1)/2;
11             w *= s;
12             h *= s;
13         }
14
15 };
16
17 int main(){
18     Rect R1 = {3,5,2,2};
19     Rect *p = &R1;
20
21     p->resize(0.5);
22
23     cout << p->x << " " << p->y << " " << p->w << " " << p->h;
24
25     return 0;
26 }
```

Pointer to the object class `Rect`

Output

```
3.5 4.5 1 1
```

Member Functions

ประกาศข้างใน class

- Member functions defined inside class definition
  - Do not need scope resolution operator, class name
  - Compiler attempts **`inline`**
    - Outside class, inline explicitly with keyword **`inline`**

ประกาศ นอก class

- Member functions defined outside class definition
  - Binary scope resolution operator (**`::`**)
    - "Ties" member name to class name
    - Uniquely identify functions of particular class
    - Different classes can have member functions with same name
  - Format for defining member functions

```
ReturnType ClassName::MemberFunctionName( ){
   …
}
```

  - Does not change whether function **`public`** or **`private`**

Member Functions

```
1   #include<iostream>
2   using namespace std;
3
4   class Rect{
5       public:
6           double x,y,w,h;
7           void resize(double);
8   };
9
10  void Rect::resize(double s){
11      x += w*(1-s)/2;
12      y += w*(s-1)/2;
13      w *= s;
14      h *= s;
15  }
16
17  int main(){
18      Rect R1 = {3,5,2,2};
19
20      R1.resize(0.5);
21
22      cout << R1.x << " " << R1.y << " " << R1.w << " " << R1.h;
23
24      return 0;
```

class

Function prototype must be given in the class definition.

Definition of a function named **`resize`** which is a member functions of class **`Rect`**

Data members of class **`Rect`** can be accessed here.

Access Modes

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

class Student{
    private:
        int age;
    public:
        string name;
        void setProfiles(string,int);
        void hbd();
        void letsGo2ThePub();
};

void Student::setProfiles(string n,int a){
    name = n;
    age = a;
}

void Student::hbd(){
    age++;
    cout << "Happy Birthday!!!\n";
}

void Student::letsGo2ThePub(){
    if(age >= 20){
        cout << "Let's go to the pub!!!\n";
    }else{
        cout << "Sorry, " << name << " can not enter the pub yet.\n";
    }
}
```

Class object can be copied by '='

```cpp
int main(){
    Student x,y;
    x.setProfiles("Karn",18);
    y = x;
    cout << y.name << "\n";
    y.name = "Santi";
    cout << y.name << "\n";
    //cout << y.age << "\n";

    return 0;
}
```

`name` is public members:
`name` can be accessed outside class definition

`age` is private members:
`age` can not be accessed outside class definition

Output

```
Karn
Santi
```

Access Modes

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

class Student{
    private:
        int age;
    public:
        string name;
        void setProfiles(string,int);
        void hbd();
        void letsGo2ThePub();
};

void Student::setProfiles(string n,int a){
    name = n;
    age = a;
}

void Student::hbd(){
    age++;
    cout << "Happy Birthday!!!\n";
}

void Student::letsGo2ThePub(){
    if(age >= 20){
        cout << "Let's go to the pub!!!\n";
    }else{
        cout << "Sorry, " << name << " can not enter the pub yet.\n";
    }
}
```

`age` can be accessed by member functions of the class

```cpp
int main(){
    Student x;
    x.setProfiles("Karn",18);
    x.letsGo2ThePub();
    x.hbd();
    x.letsGo2ThePub();
    x.hbd();
    x.letsGo2ThePub();

    return 0;
}
```

Output

```
Sorry, Karn can not enter the pub yet. (18)
Happy Birthday!!! (19)
Sorry, Karn can not enter the pub yet. (19)
Happy Birthday!!! (20)
Let's go to the pub!!! (20)
```

Access Modes

```
1  #include<iostream>
2  using namespace std;
3
4  class Student{
5      private:
6          int age;
7          bool confirmProfiles();
8      public:
9          string name;
10         void setProfiles(string,int);
11         void hbd();
12         void letsGo2ThePub();
13 };
14
15 bool Student::confirmProfiles(){
16     char ans;
17     cout << "Your name is " << name;
18     cout << " and you are " << age << " years old.\n";
19     cout << "Comfirm? (y/n):";
20     cin >> ans;
21     if(ans == 'y') return true;
22     else return false;
23 }
```

```
26 void Student::setProfiles(string n,int a){
27     name = n;
28     age = a;
29     if(!confirmProfiles()){
30         cout << "Delete your profiles.";
31         name = '\0';
32         age = 0;
33     }
34 }
35
36 void Student::hbd(){
37     age++;
38     cout << "Happy Birthday!!!\n";
39 }
40
41 void Student::letsGo2ThePub(){
42     if(age >= 20){
43         cout << "Let's go to the pub!!!\n";
44     }else{
45         cout << "Sorry, " << name << " can not enter the pub yet.\n";
46     }
47 }
48
49 int main(){
50     Student x;
51     x.setProfiles("Karn",18);
52     cout << x.name << "\n";
53     // x.confirmProfiles();
54     return 0;
55 }
```

```
Your name is Karn and you are 18 years old.
Confirm? (y/n):y
Karn
```

```
Your name is Karn and you are 18 years old.
Confirm? (y/n):n
Delete your profiles.
```

`confirmProfiles` is private member function: `confirmProfiles` can not be accessed outside class definition but still can be accessed by other member functions of the class

Example 21-A: Hero

```
1  #include<iostream>
2  using namespace std;
3
4  class Hero{
5      string name;
6      int hp;
7      int hpmax;
8      int atk;
9      int def;
10     void showStatus();
11 public:
12     void create(string);
13     void train();
14 };
15
16 void Hero::create(string s){
17     name = s;
18     hp = hpmax = 10;
19     atk = def = 5;
20     showStatus();
21 }
22
23 void Hero::showStatus(){
24     cout << name << ": HP = " << hp << "/" << hpmax;
25     cout << " ATK = " << atk << " DEF = "<< def << "\n";
26 }
27
28 void Hero::train(){
29     atk += 3;
30     def += 3;
31     cout << ">>Increase ATK and DEF of " << name << " by 3\n";
32     showStatus();
33 }
```

private

Create array of objects
(In this example, you can also use normal array or vector)

Hero 2 ตัว

```
35 int main(){
36     Hero *x = new Hero[2];
37     x[0].create("Prayath");
38     x[1].create("Tukkie");
39
40     x[0].train();
41     x[0].train();
42
43     x[1].train();
44
45     delete [] x;
46     return 0;
47 }
```

Output

```
Prayath: HP = 10/10 ATK = 5 DEF = 5
Tukkie: HP = 10/10 ATK = 5 DEF = 5
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 8 DEF = 8
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 11 DEF = 11
>>Increase ATK and DEF of Tukkie by 3
Tukkie: HP = 10/10 ATK = 8 DEF = 8
```

Example 21-A: Hero

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

struct Status{
    int hp;
    int hpmax;
    int atk;
    int def;
};

class Hero{
    string name;
    Status stat;
    void showStatus();
    public:
        void create(string);
        void train();
};

void Hero::create(string s){
    name = s;
    stat.hp = stat.hpmax = 10;
    stat.atk = stat.def = 5;
    showStatus();
}

void Hero::showStatus(){
    cout << name << ": HP = " << stat.hp << "/" << stat.hpmax;
    cout << " ATK = " << stat.atk << " DEF = "<< stat.def << "\n";
}

void Hero::train(){
    stat.atk += 3;
    stat.def += 3;
    cout << ">>Increase ATK and DEF of " << name << " by 3\n";
    showStatus();
}
```

Class members can be structs or other classes

```cpp
int main(){
    Hero *x = new Hero[2];
    x[0].create("Prayath");
    x[1].create("Tukkie");

    x[0].train();
    x[0].train();

    x[1].train();

    delete [] x;
    return 0;
}
```

Output

```
Prayath: HP = 10/10 ATK = 5 DEF = 5
Tukkie: HP = 10/10 ATK = 5 DEF = 5
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 8 DEF = 8
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 11 DEF = 11
>>Increase ATK and DEF of Tukkie by 3
Tukkie: HP = 10/10 ATK = 8 DEF = 8
```

Example 21-A: Hero

```
4  class Hero{
5      string name;
6      int hp;
7      int hpmax;
8      int atk;
9      int def;
10     void showStatus();
11   public:
12       void create(string);
13       void train();
14       void attack(Hero &);
15       void beAttacked(int);
16 };
17
18 void Hero::attack(Hero &target){
19     cout << ">>" << name << " attacked.\n";
20     target.beAttacked(atk);
21 }
22
23 void Hero::beAttacked(int oppatk){
24     int dmg = oppatk-def;
25     if(dmg < 0) dmg = 0;
26     hp = hp-dmg;
27     if(hp <= 0) cout << ">>" << name << " was defeated.\n";
28     else showStatus();
29 }
```

pass by reference operator

A attack B
atk = 10
A.attack(B);
B.beAttacked(atk);

```
56 int main(){
57     Hero *x = new Hero[2];
58     x[0].create("Prayath"); x[1].create("Tukkie");
59
60     x[0].train(); x[0].train();
61
62     x[0].attack(x[1]);
63     x[1].attack(x[0]);
64     x[0].attack(x[1]);
65
66     delete [] x;
67     return 0;
68 }
```

Output

```
Prayath: HP = 10/10 ATK = 5 DEF = 5
Tukkie: HP = 10/10 ATK = 5 DEF = 5
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 8 DEF = 8
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 11 DEF = 11
>>Prayath attacked.
Tukkie: HP = 4/10 ATK = 5 DEF = 5
>>Tukkie attacked.
Prayath: HP = 10/10 ATK = 11 DEF = 11
>>Prayath attacked.
>>Tukkie was defeated.
```

Example 21-A: Hero

```
4  class Hero{
5      string name;
6      int hp;
7      int hpmax;
8      int atk;
9      int def;
10     void showStatus();
11     public:
12         void create(string);
13         void train();
14         void attack(Hero *);
15         void beAttacked(int);
16 };
17
18 void Hero::attack(Hero *target){
19     cout << ">>" << name << " attacked.\n";
20     target->beAttacked(atk);
21 }
22
23 void Hero::beAttacked(int oppatk){
24     int dmg = oppatk-def;
25     if(dmg < 0) dmg = 0;
26     hp = hp-dmg;
27     if(hp <= 0) cout << ">>" << name << " was defeated.\n";
28     else showStatus();
29 }
```

pass by pointer

```
51 int main(){
52     Hero *x = new Hero[2];
53     x[0].create("Prayath"); x[1].create("Tukkie");
54
55     x[0].train(); x[0].train();
56
57     x[0].attack(&x[1]);
58     x[1].attack(&x[0]);
59     x[0].attack(&x[1]);
60
61     delete [] x;
62     return 0;
63 }
```

Output

```
Prayath: HP = 10/10 ATK = 5 DEF = 5
Tukkie: HP = 10/10 ATK = 5 DEF = 5
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 8 DEF = 8
>>Increase ATK and DEF of Prayath by 3
Prayath: HP = 10/10 ATK = 11 DEF = 11
>>Prayath attacked.
Tukkie: HP = 4/10 ATK = 5 DEF = 5
>>Tukkie attacked.
Prayath: HP = 10/10 ATK = 11 DEF = 11
>>Prayath attacked.
>>Tukkie was defeated.
```

Example 21-A: Hero

ไม่ได้เป็น member

```
1  #include<iostream>
2  #include<cstdlib>
3  #include<ctime>
4  using namespace std;
5
6
7  class Hero{
8      string name;
9      int hp;
10     int hpmax;
11     int atk;
12     int def;
13     void showStatus();
14   public:
15         void create(string);
16         void train();
17         void attack(Hero *);
18         void beAttacked(int);
19         bool isDead();
20 };
21
22 bool Hero::isDead(){
23     if(hp <= 0) return true;
24     else return false;
25 }
```

```
59 Hero * deathmatch(Hero *x, Hero *y){
60     while(1){
61         if(rand()%4 == 0){
62             x->attack(y); y->attack(x);
63         }else if(rand()%4 == 1){
64             x->attack(y); y->train();
65         }else if(rand()%4 == 2){
66             y->attack(x); x->train();
67         }else{
68             x->train(); y->train();
69         }
70         if(x->isDead() && y->isDead()) return 0;
71         else if(x->isDead()) return y;
72         else if(y->isDead()) return x;
73     }
74 }
```

```
76 int main(){
77     srand(time(0));
78     Hero x[4];→สร้าง Hero 4 คน
79     x[0].create("Sassard"); x[1].create("Episat");
80     x[2].create("Thanaporn"); x[3].create("Paitoon");
81
82     Hero *winner = 0;
83     Hero *w1 = deathmatch(&x[0],&x[1]);
84     Hero *w2 = deathmatch(&x[2],&x[3]);
85     if(w1 != 0 && w2 != 0) winner = deathmatch(w1,w2);
86     else if(w1 == 0 && w2 != 0) winner = w2;
87     else if(w1 != 0 && w2 == 0) winner = w1;
88
89     return 0;
90 }
```

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

class ABC{
    int val;
    public:
        void set(int n){
            val = n;
        }

        int square(){
            return val*val;
        }
};


int main(){
    ABC x;
    x.set(5);
    cout << x.square();
}
```

data member

function member

=

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

struct ABC{
    void set(int n){
        val = n;
    }

    int square(){
        return val*val;
    }

    private:
        int val;
};


int main(){
    ABC x;
    x.set(5);
    cout << x.square();
}
```

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

struct ABC{
    int val = 69;
};


int main(){
    ABC x;
    cout << x.val;
}
```

non-static data member initializers only available with -std=c++11 or -std=gnu++11

261102
Computer Programming

Lecture 22: Classes II

(Constructor & Destructor)

Initialize data member

ทบทวน

```
1   #include<iostream>
2   using namespace std;
3
4   class Circle{
5       public:
6           double x;
7           double y;
8           double r;
9           void resize(double);
10          void move(double,double);
11  };
12
13  int main(){
14      Circle A = {1,2,3};
15
16      Circle B;
17      B.x = 4; B.y = 5; B.r = 6;
18
19      return 0;
20  }
```

public data members of the class can be initialized by using initializer or can be read or modified by accessing data members directly .

Initialize data member

```
1   #include<iostream>
2   using namespace std;
3
4   class Circle{
5       private:
6           double x;
7           double y;
8           double r;
9       public:
10          void set(double,double,double);
11          void resize(double);
12          void move(double,double);
13  };
14
15  void Circle::set(double a,double b,double c){
16      x = a; y = b; r = c;
17  }
18
19  int main(){
20      Circle A;
21      A.set(1,2,3);
22
23      return 0;
24  }
```

private data member can be read or modified by using access functions.

Get function can be defined for reading **private** data

Set function can be defined for modifying **private** data

Initialize data member by Constructor

ชื่อเหมือน class

ถูกเรียกตอนสร้างตัวแปร

```
6  class Circle{
7      private:
8          double x;
9          double y;
10         double r;
11     public:
12         Circle(double,double,double);
13         void resize(double);
14         void move(double,double);
15 };
16
17 Circle::Circle(double a,double b,double c){
18     x = a; y = b; r = c;
19 }
20
21 int main(){
22     Circle A(1,2,3);
23     Circle *B = new Circle(4,5,6);
24
25     return 0;
26 }
```

แบบเดิม

```
Circle A;
A.set(1,2,3)
```

- Constructor function
  - Special member function
    - Initializes data members
    - Same name as class
  - Called when object instantiated
  - Several constructors
    - Function overloading

สร้าง constructors หลายตัวได้

** No return type **

~~void~~

Constructor function called when the object is declared

Constructor Function

- Constructors
  - Initialize data members
    - Or can set later
  - Same name as class
  - No return type — ~~void~~
- Initializers
  - Passed as arguments to constructor
  - In parentheses to the right of class name before semicolon

Dynamic ; Class *name = new Class(___);

```
Class-type ObjectName( value1,value2,…);
```

Constructor Function

- Default constructors
  - Defaults all arguments
    OR
  - Using function overload
  - Explicitly requires no arguments
  - Can be invoked with no arguments
  - Only one per class

Circle A (1,2,3);
Circle B;

Circle::Circle(double x=0, double y=0, double r=0)

Circle A (1); → error
error

Circle(double, double, double)
Circle();

Default Constructors

```
6  class Circle{
7      private:
8          double x;
9          double y;
10         double r;
11     public:
12         Circle(double,double,double);
13         void resize(double);
14         void move(double,double);
15 };
16
17 Circle::Circle(double a = 0,double b = 0,double c = 1){
18     x = a; y = b; r = c;
19     cout << "A circle with radius = " << r ;
20     cout << " was created at (" << x << "," << y << ")\n";
21 }
22
23 int main(){
24     Circle A(1,2,3);
25     Circle B;
26     Circle *C = new Circle(4,5,6);
27     Circle *D = new Circle;
28
29     return 0;
30 }
```

Output

```
A circle with radius = 3 was created at (1,2)
A circle with radius = 1 was created at (0,0)
A circle with radius = 6 was created at (4,5)
A circle with radius = 1 was created at (0,0)
```

Defaults all arguments

Initial by using default arguments

Circle A(1) ; ~ Error

Default Constructors

```
6  class Circle{
7      private:
8          double x;
9          double y;
10         double r;
11     public:
12         Circle();
13         Circle(double,double,double);
14         void resize(double);
15         void move(double,double);
16 };
17
18 Circle::Circle(){
19     x = 0; y = 0; r = 1;
20     cout << "A circle with radius = " << r ;
21     cout << " was created at (" << x << "," << y << ")\n";
22 }
23
24 Circle::Circle(double a,double b,double c){
25     x = a; y = b; r = c;
26     cout << "A circle with radius = " << r ;
27     cout << " was created at (" << x << "," << y << ")\n";
28 }
29
30 int main(){
31     Circle A(1,2,3);
32     Circle B;
33     Circle *C = new Circle(4,5,6);
34     Circle *D = new Circle;
35
36     return 0;
37 }
```

Output

```
A circle with radius = 3 was created at (1,2)
A circle with radius = 1 was created at (0,0)
A circle with radius = 6 was created at (4,5)
A circle with radius = 1 was created at (0,0)
```

Constructor function overloading
(functions with same name but different arguments)

Initialized by using function overloading

Circle A (1); ~ error

Function Overloading

```
6  class Circle{
7      private:
8          double x;
9          double y;
10         double r;
11     public:
12         Circle(string);
13         Circle(double,double,double);
14         void resize(double);
15         void move(double,double);
16 };
```

```
18 Circle::Circle(string s){
19     if(s == "small"){
20         x = 0; y = 0; r = 0.1*(rand()%11);
21     }else if(s == "big"){
22         x = 0; y = 0; r = 10*(rand()%11);
23     }else{
24         cout << "Invalid Input";
25     }
26     cout << "A circle with radius = " << r ;
27     cout << " was created at (" << x << "," << y << ")\n";
28 }
29
30 Circle::Circle(double a = 0,double b = 0,double c = 1){
31     x = a; y = b; r = c;
32     cout << "A circle with radius = " << r ;
33     cout << " was created at (" << x << "," << y << ")\n";
34
35 }
```

```
37 int main(){
38     srand(time(0));
39     Circle A(1,2,3);
40     Circle B("small");
41     Circle C("big");
42     return 0;
43 }
```

Output

```
A circle with radius = 3 was created at (1,2)
A circle with radius = 0.7 was created at (0,0)
A circle with radius = 30 was created at (0,0)
```

ชื่อเหมือน class + (~)

Destructor Function

- Destructors
  - Special member function
  - Called implicitly by compiler when object is deleted or exits the scope
  - Same name as class
    - Preceded with tilde (~)
  - No arguments
  - No return value
  - Cannot be overloaded
  - Performs "termination housekeeping"
    - Before system reclaims object's memory
      - Reuse memory for new objects
  - Generally, destructor calls reverse order of constructor calls
  - No explicit destructor
    - Compiler creates "empty" destructor"

ถูกเรียก เมื่อ ถูกทำลาย

~Circle();

เคลียร์ memory

Destructor Function

```
6  class Circle{
7      private:
8          double x;
9          double y;
10         double r;
11     public:
12         Circle(double,double,double);
13         ~Circle();
14         void resize(double);
15         void move(double,double);
16 };
17
18 Circle::Circle(double a = 0,double b = 0,double c = 1){
19     x = a; y = b; r = c;
20     cout << "A circle with radius = " << r ;
21     cout << " was created at (" << x << "," << y << ")\n";
22
23 }
24
25 Circle::~Circle(){
26     cout << "A circle with radius = " << r ;
27     cout << " at (" << x << "," << y << ") was destroyed.\n";
28 }
```

```
31 int main(){
32     Circle A(1,2,3);
33
34     Circle *B = new Circle;
35     delete B;
36
37     Circle C(4,5,6);
38
39     for(int i = 0; i < 2; i++){
40         Circle D(7,8,9);
41     }
42
43     return 0;
44 }
```

Output

```
A circle with radius = 3 was created at (1,2)
A circle with radius = 1 was created at (0,0)
A circle with radius = 1 at (0,0) was destroyed.
A circle with radius = 6 was created at (4,5)
A circle with radius = 9 was created at (7,8)
A circle with radius = 9 at (7,8) was destroyed.
A circle with radius = 9 was created at (7,8)
A circle with radius = 9 at (7,8) was destroyed.
A circle with radius = 6 at (4,5) was destroyed.
A circle with radius = 3 at (1,2) was destroyed.
```

Destructor Function

```cpp
class Animal{
    string name;
    string type;
    int age;
    double weight;
    int price;
    public:
        Animal(string,string,int,double,int);
        ~Animal();
};


class Farm{
    string name;
    int space;
    vector<Animal*> animals;
    public:
        Farm(string,int);
        ~Farm();
        void addAnimal(Animal *);
};
```

```cpp
Animal::Animal(string n,string t,int a,double w,int p){
    name = n; type = t;
    age = a; weight = w; price = p;
}

Animal::~Animal(){
    cout << name << " was deleted.\n";
}

Farm::Farm(string n, int s){
    name = n; space = s;
    cout << name << " farm was created.\n";
}

Farm::~Farm(){
    cout << name << " farm was deleted.\n";
}

void Farm::addAnimal(Animal *a){
    animals.push_back(a);
    cout << name << " has " << animals.size() << " animals.\n";
}
```

```cpp
int main(){
    Farm myfarm("KalaLand",69);

    Animal x("PJung","Pig",1,3,6900);
    Animal *y = new Animal("Putin","Rabbit",2,1,555);

    myfarm.addAnimal(&x);
    myfarm.addAnimal(y);
    myfarm.addAnimal(new Animal("Laboon","Whale",30,1000,10000000));
    myfarm.addAnimal(new Animal("Mickey","Mouse",1,0.5,100));

    return 0;
}
```

Dynamic objects are not deleted.

ลบ automatic

อ๊อบตัวอื่นๆ ยังไม่ถูก deleted – เกิด memory leak

Output

```
KalaLand farm was created.
KalaLand has 1 animals.
KalaLand has 2 animals.
KalaLand has 3 animals.
KalaLand has 4 animals.
PJung was deleted.
KalaLand farm was deleted.
```

Destructor Function

```
30 Animal::Animal(string n,string t,int a,double w,int p){
31     name = n; type = t;
32     age = a; weight = w; price = p;
33 }
34
35 Animal::~Animal(){
36     cout << name << " was deleted.\n";
37 }
38
39 Farm::Farm(string n, int s){
40     name = n; space = s;
41     cout << name << " farm was created.\n";
42 }
43
44 Farm::~Farm(){
45     for(int i = 0; i < animals.size(); i++){
46         delete animals[i];
47     }
48     cout << name << " farm was deleted.\n";
49 }
50
51 void Farm::addAnimal(Animal *a){
52     animals.push_back(a);
53     cout << name << " has " << animals.size() << " animals.\n";
54 }
```

```
56 int main(){
57     Farm myfarm("KalaLand",69);
58
59     myfarm.addAnimal(new Animal("PJung","Pig",1,3,6900));
60     myfarm.addAnimal(new Animal("Putin","Rabbit",2,1,555));
61     myfarm.addAnimal(new Animal("Laboon","Whale",30,1000,10000000));
62     myfarm.addAnimal(new Animal("Mickey","Mouse",1,0.5,100));
63
64     return 0;
65 }
```

Output

```
KalaLand farm was created.
KalaLand has 1 animals.
KalaLand has 2 animals.
KalaLand has 3 animals.
KalaLand has 4 animals.
PJung was deleted.
Putin was deleted.
Laboon was deleted.
Mickey was deleted.
KalaLand farm was deleted.
```

We can delete all dynamic `Animal` objects by using delete operator in `Farm` destructor function.

261102
Computer Programming

Lecture 23: Command Line Arguments

Command Line Arguments

hello.cpp

C: > Users > surface > hello.cpp > main()

```cpp
#include<iostream>

using namespace std;

int main(){
    cout << "Yo Yo Say Hello Yo Yo\n";
    cout << "\\(^ ^)/";

    return 0;
}
```

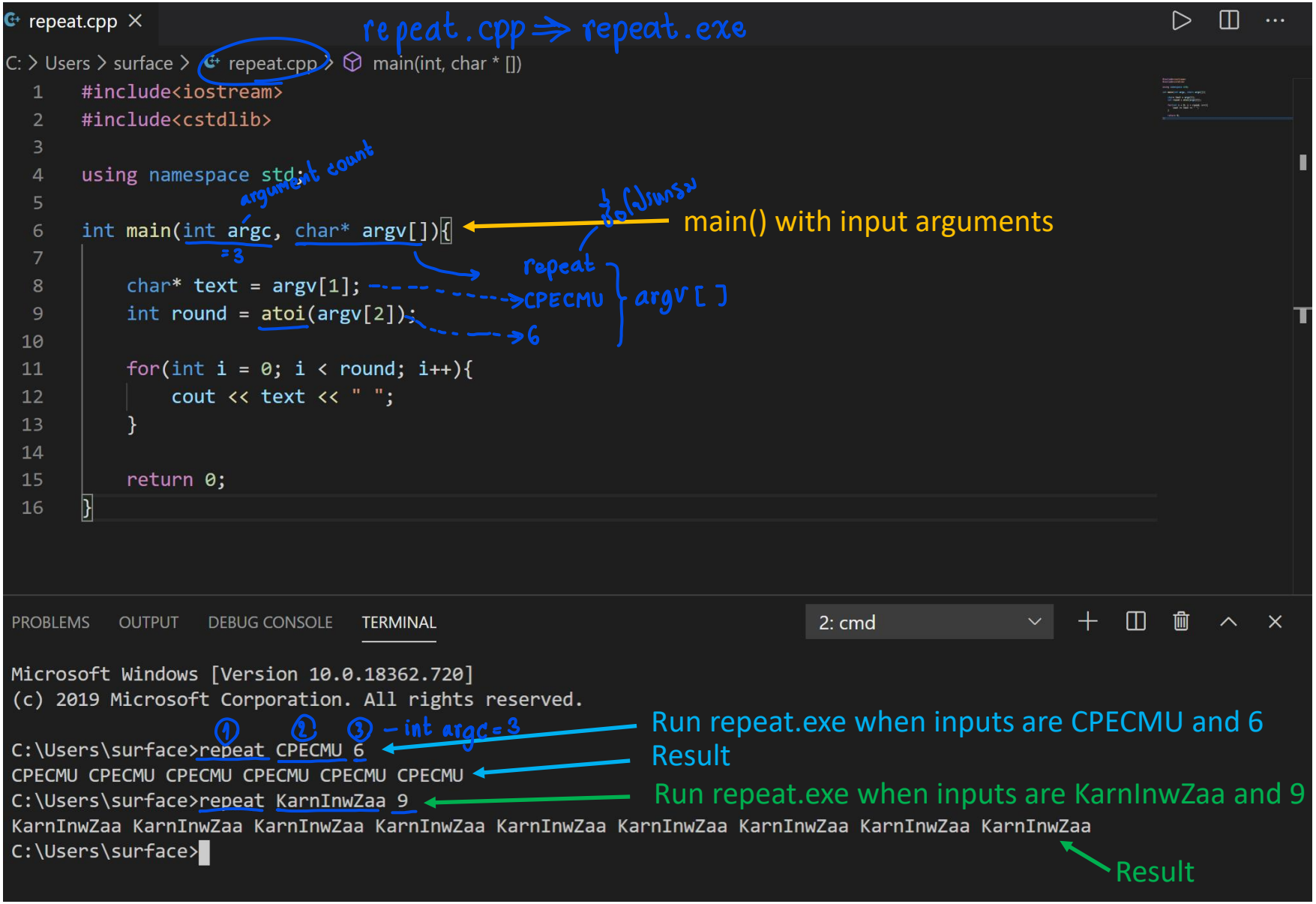
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: cmd

```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\surface>hello
Yo Yo Say Hello Yo Yo
\(^ ^)/
C:\Users\surface>
```

Run hello.exe

Result

Command Line Arguments



Command Line Arguments

- When we compile and link your program, the compiler produces an executable (binary) file – a program
- Command line arguments are optional parameters that user can pass to a program at run-time
- Adding the ability to parse command-line parameters can be done by adding two parameters to main
  - `int main( int argc, char* argv[] )`
  - `int main( int argc, char** argv )`

Command Line Arguments

- `int argc`
  - An integer containing the number of arguments passed to the program
  - `argc` = argument count
  - `argc` will always be at least 1 (1st argument is the program itself)
- `char *argv[]`
  - An array of C-style strings, with the length of `argc`
  - `argv` = argument vector
  - The actual arguments

Command Line Arguments

- The WordCount.exe program can be run in command line as follows:
  - `C:\>WordCount Myfile.txt`
  - `C:\>WordCount file1.txt file2.txt`

- This way your program can be called repetitively by shell scripting
  - Windows: BATCH file (`*.bat`)
  - Linux: shell script (`*.sh, *.pl, *.py, …`)

Command Line Arguments

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

int main(int argc,char *argv[]){

    cout << "#Input = " << argc << "\n";

    for(int i = 0; i < argc;i++){
        cout << "Input [" << i << "] = " << argv[i] << "\n";
    }

    return 0;
}
```

test.cpp -> test.exe

```
D:\temp>test
#Input = 1
Input [0] = test

D:\temp>test aa bb cc 1 2 3 4.5
#Input = 8
Input [0] = test
Input [1] = aa
Input [2] = bb
Input [3] = cc
Input [4] = 1
Input [5] = 2
Input [6] = 3
Input [7] = 4.5

D:\temp>
```

Command Line Arguments

```
1  #include<iostream>
2  #include<cstdlib>
3  using namespace std;
4
5  int main(int argc,char *argv[]){
6
7      int a,b;
8
9      if(argc != 3){
10         cout << "Invalid input arguments\n";
11     }else{
12         a = atoi(argv[1]);
13         b = atoi(argv[2]);
14         cout << "sum = " << a+b;
15     }
16
17     return 0;
18 }
```

plus.cpp -> plus.exe

```
D:\temp>plus
Invalid input arguments

D:\temp>plus 1 2 3
Invalid input arguments

D:\temp>plus 1 2
sum = 3
D:\temp>plus 4 7
sum = 11
D:\temp>
```

Command Line Arguments

wordcount.exe

```
D:\temp>wordcount song.txt love
#LOVE = 4
D:\temp>wordcount song.txt you
#YOU = 4
D:\temp>wordcount song.txt beginning
#BEGINNING = 1
D:\temp>
```

Command Line Arguments

wordcount.exe

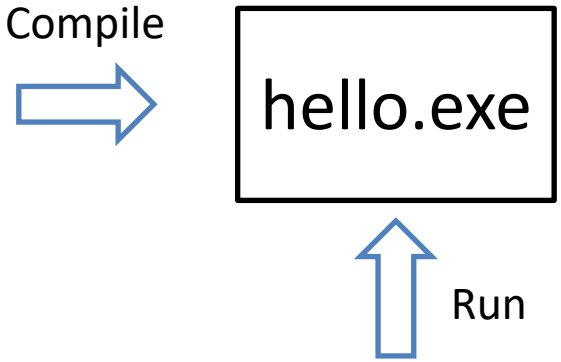
```cpp
#include<iostream>
#include<fstream>
#include<string>
#include<cctype>
using namespace std;

int main(int argc,char *argv[]){

    if(argc != 3){
        cout << "Invalid input arguments\n";
    }else{
        char *key = argv[2];
        int i = 0;
        while(key[i]){
            key[i] = toupper(key[i]);
            i++;
        }

        ifstream fin(argv[1]);
        int count = 0;
        string s;
        while(getline(fin,s)){
            int i = 0;
            while(s[i]){
                s[i] = toupper(s[i]);
                i++;
            }
```

```cpp
        ifstream fin(argv[1]);
        int count = 0;
        string s;
        while(getline(fin,s)){
            int i = 0;
            while(s[i]){
                s[i] = toupper(s[i]);
                i++;
            }

            int idx = s.find(key);
            while(idx != -1){
                count++;
                idx = s.find(key,idx+4);
            }
        }
        cout << "#" << key << " = " << count;
    }

    return 0;
}
```

Running .exe in C++

```cpp
#include<iostream>

using namespace std;

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

hello.cpp




```cpp
#include<windows.h>

int main(){
    system("hello.exe");
    return 0;
}
```

```
Hello
--------------------------------
Process exited after 0.08532 seconds with return value 0
Press any key to continue . . .
```

Running .exe in C++

```
1   #include<iostream>
2   #include<windows.h>
3
4   using namespace std;
5
6   int main(){
7       system("plus.exe 1 2");
8       cout << "\n";
9       system("plus 4 7");
10      return 0;
11  }
```

```
sum = 3
sum = 11
--------------------------------
Process exited after 0.1312 seconds with return value 0
Press any key to continue . . .
```

Running .exe in C++

plus.cpp

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

int main(int argc, char *argv[]){
    int a,b;

    if(argc != 3){
        cout << "error\n";
        return 1;
    }
    else{
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        cout << a+b << "\n";
        return 0;
    }
}
```

เพื่อแสดง error เลยอ่ะไรก็ได้

จบ happy ending

```cpp
#include<windows.h>

int main(){
    int ret = system("plus 1 2 3");
    if(ret == 1) system("plus 1 2");
    return 0;
}
```

ไม่ return

```
error
3
```

Running .exe in C++

```cpp
#include<windows.h>

int main(){
    system("git init");
    system("git add hello.cpp");
    system("git commit -m \"Create hello.cpp.\"");
    return 0;
}
```

```
Initialized empty Git repository in D:/temp/.git/
[master (root-commit) 51bb011] Create hello.cpp.
 1 file changed, 8 insertions(+)
 create mode 100644 hello.cpp


--------------------------------
Process exited after 0.4886 seconds with return value 0
Press any key to continue . . .
```

261102
Computer Programming

Lecture 24: Operator Overloading

Operators

- Use operators with objects
  - Clearer than function calls for certain classes
    - `object2 = object1.add(object2); //using function`
    - `object2 = object2 + object1; //using operator`
- Examples
  - <<
    - Stream insertion, bitwise left-shift
  - +
    - Performs arithmetic on multiple types (integers, floats, etc.)
  - &
    - Returns address of object, bitwise and

Operator Overloading

void operator=( )
void operator*=( )
void operator%( )

- Overloading operators
  - Create a function for the class
  - Name function `operator` followed by symbol
    - `Operator+` for the addition operator +
  - Can use existing operators with user-defined types
    - Cannot create new operators

ไม่สามารถสร้าง operator ใหม่ขึ้นมาได้

Operator overloading

เปลี่ยนความหมายได้

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Bitwise

Bitwise x>>=2 x=x>>2

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Operator overloading

- What CANNOT be changed?
  - How operators act on built-in data types
    - I.e., cannot change integer addition
  - Precedence of operator (order of evaluation)
    - Use parentheses to force order-of-operations
  - Associativity (left-to-right or right-to-left)
  - Number of operands
    - **&** is unitary, only acts on one operand
- Cannot create new operators
- Operators must be overloaded explicitly
  - Overloading **+** does not overload **+=**

Precedence of operators

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++ a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`. ->`	Member access	
3	`++a --a`	Prefix increment and decrement	Right-to-left
	`+a -a`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(`*type*`)`	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`new new[]`	Dynamic memory allocation	
	`delete delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer-to-member	Left-to-right
5	`a*b a/b a%b`	Multiplication, division, and remainder	
6	`a+b a-b`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
9	`== !=`	For relational operators = and ≠ respectively	
10	`a&b`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	
15	`a?b:c`	Ternary conditional[note 2]	Right-to-left
	`throw`	throw operator	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+= -=`	Compound assignment by sum and difference	
	`*= /= %=`	Compound assignment by product, quotient, and remainder	
	`<<= >>=`	Compound assignment by bitwise left shift and right shift	
	`&= ^= |=`	Compound assignment by bitwise AND, XOR, and OR	
16	`,`	Comma	Left-to-right

http://en.cppreference.com/w/cpp/language/operator_precedence

Example 24-A: Using Member Function

```
1   #include<iostream>
2   using namespace std;
3
4   class Circle{
5       private:
6           double x;
7           double y;
8           double r;
9       public:
10          Circle(double,double,double);
11          void showDetail();
12          void resize(double);
13          void move(double,double);
14          bool compare(const Circle &);
15  };
16
17  Circle::Circle(double a = 0,double b = 0,double c = 1){
18      x = a; y = b; r = c;
19  }
20
21  void Circle::showDetail(){
22      cout << "x = " << x << ", y = " << y << ", r = " << r << "\n";
23  }
24
25  bool Circle::compare(const Circle &c){
26      if(x == c.x && y == c.y && r == c.r){
27          return true;
28      }else{
29          return false;
30      }
31  }
```

```
33  int main(){
34      Circle A(1,2,3);
35      Circle B(4,5,6);
36
37      if(A.compare(B)) cout << "yes\n";
38      else cout << "no\n";
39
40
41      if(A.compare(Circle(1,2,3))) cout << "yes\n";
42      else cout << "no\n";
43
44      return 0;
45  }
```

Member function calling

Output

```
no
yes
```

The private data member of class **`Circle`** can be accessed from other **`Circle`** object inside class definition

Example 24-A: Using Non-member Function

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

class Circle{
    public:
        double x;
        double y;
        double r;
        Circle(double,double,double);
        void showDetail();
        void resize(double);
        void move(double,double);
};

Circle::Circle(double a = 0,double b = 0,double c = 1){
    x = a; y = b; r = c;
}

void Circle::showDetail(){
    cout << "x = " << x << ", y = " << y << ", r = " << r << "\n";
}

bool compare(const Circle &c1,const Circle &c2){
    if(c1.x == c2.x && c1.y == c2.y && c1.r == c2.r){
        return true;
    }else{
        return false;
    }
}
```

Public data members

Typical function with two inputs

```cpp
int main(){
    Circle A(1,2,3);
    Circle B(4,5,6);

    if(compare(A,B)) cout << "yes\n";
    else cout << "no\n";


    if(compare(A,Circle(1,2,3))) cout << "yes\n";
    else cout << "no\n";

    return 0;
}
```

Function calling

The non-member function can access only public data members

Output

```
no
yes
```

Example 24-A: Using Operator Overloading

(Member Function)

```
1  #include<iostream>
2  using namespace std;
3
4  class Circle{
5    private:
6        double x;
7        double y;
8        double r;
9    public:
10       Circle(double,double,double);
11       void showDetail();
12       void resize(double);
13       void move(double,double);
14       bool operator==(const Circle &);
15 };
16
17 Circle::Circle(double a = 0,double b = 0,double c = 1){
18     x = a; y = b; r = c;
19 }
20
21 void Circle::showDetail(){
22     cout << "x = " << x << ", y = " << y << ", r = " << r << "\n";
23 }
24
25 bool Circle::operator==(const Circle &c){
26     if(x == c.x && y == c.y && r == c.r){
27         return true;
28     }else{
29         return false;
30     }
31 }
```

Define the operator as member function

argument is only right operand

```
33 int main(){
34     Circle A(1,2,3);
35     Circle B(4,5,6);
36
37     if(A == B) cout << "yes\n";
38     else cout << "no\n";
39
40     Operator calling
41     if(A == Circle(1,2,3)) cout << "yes\n";
42     else cout << "no\n";
43
44     return 0;
45 }
```

Output

```
no
yes
```

The private data member of class **`Circle`** can be accessed from other **`Circle`** object inside class definition

Example 24-A: Using Operator Overloading

(Non-member Function)

```
1  #include<iostream>
2  using namespace std;
3
4  class Circle{
5      public:
6          double x;
7          double y;
8          double r;
9          Circle(double,double,double);
10         void showDetail();
11         void resize(double);
12         void move(double,double);
13 };
14
15 Circle::Circle(double a = 0,double b = 0,double c = 1){
16     x = a; y = b; r = c;
17 }
18
19 void Circle::showDetail(){
20     cout << "x = " << x << ", y = " << y << ", r = " << r << "\n";
21 }
22
23 bool operator==(const Circle &c1,const Circle &c2){
24     if(c1.x == c2.x && c1.y == c2.y && c1.r == c2.r){
25         return true;
26     }else{
27         return false;
28     }
29 }
```

Public data members

```
33 int main(){
34     Circle A(1,2,3);
35     Circle B(4,5,6);
36
37     if(A == B) cout << "yes\n";
38     else cout << "no\n";
39
40
41     if(A == Circle(1,2,3)) cout << "yes\n";
42     else cout << "no\n";
43
44     return 0;
45 }
```

Output

```
no
yes
```

The non-member operator can access only public data members

Example 24-A: Using Operator Overloading

(Friend Function)

```
1  #include<iostream>
2  using namespace std;
3
4  class Circle{
5      double x;
6      double y;
7      double r;
8      public:
9          Circle(double,double,double);
10         void showDetail();
11         void resize(double);
12         void move(double,double);
13         friend bool operator==(const Circle &,const Circle &);
14 };
15
16 Circle::Circle(double a = 0,double b = 0,double c = 1){
17     x = a; y = b; r = c;
18 }
19
20 void Circle::showDetail(){
21     cout << "x = " << x << ", y = " << y << ", r = " << r << "\n";
22 }
23
24 bool operator==(const Circle &c1,const Circle &c2){
25     if(c1.x == c2.x && c1.y == c2.y && c1.r == c2.r){
26         return true;
27     }else{
28         return false;
29     }
30 }
```

private

Define operator/function as a friend of the class by using keyword friend

```
33 int main(){
34     Circle A(1,2,3);
35     Circle B(4,5,6);
36
37     if(A == B) cout << "yes\n";
38     else cout << "no\n";
39
40
41     if(A == Circle(1,2,3)) cout << "yes\n";
42     else cout << "no\n";
43
44     return 0;
45 }
```

Output

```
no
yes
```

The friend function is non-member function that can access private data members

Example 24-B: Incremental Operator ++

```
4  class Circle{
5      private:
6          double x;
7          double y;
8          double r;
9      public:
10         Circle(double,double,double);
11         void showDetail();
12         void resize(double);
13         void move(double,double);
14         bool operator==(const Circle &);
15         void operator++(); //Prefix
16         void operator++(int); //Postfix
17 };
18
19 void Circle::operator++(){
20     r++;
21     cout << "Prefix operator (++A) is called.\n";
22 }
23
24 void Circle::operator++(int){
25     r++;
26     cout << "Postfix operator (A++) is called.\n";
27 }
```

```
45 int main(){
46     Circle A(1,2,3);
47     A.showDetail();
48     ++A;
49     A.showDetail();
50     A++;
51     A.showDetail();
52
53     return 0;
54 }
```

Output

```
x = 1, y = 2, r = 3
Prefix operator (++A) is called.
x = 1, y = 2, r = 4
Postfix operator (A++) is called.
x = 1, y = 2, r = 5
```

Example 24-C: Complex Number

```
4  class ComplexNumber{
5      public:
6          double real;
7          double imag;
8          ComplexNumber(double,double);
9          ComplexNumber operator+(const ComplexNumber &);
10         ComplexNumber operator-(const ComplexNumber &);
11 };
12
13 ComplexNumber::ComplexNumber(double x = 0,double y = 0){
14     real = x; imag = y;
15 }
16
17 ComplexNumber ComplexNumber::operator+(const ComplexNumber &c){
18     return ComplexNumber(real+c.real,imag+c.imag);
19 }
20
21 ComplexNumber ComplexNumber::operator-(const ComplexNumber &c){
22     return ComplexNumber(real-c.real,imag-c.imag);
23 }
24
25 ostream &operator<<(ostream &os, const ComplexNumber &c){
26     return os << "(" << c.real << ")+(" << c.imag << ")*i";
27 }
```

```
29 int main(){
30     ComplexNumber a(3,2),b(1,4);
31     ComplexNumber c = a+b;
32     ComplexNumber d = a-b;
33     cout << c << "\n" << d;
34 }
```

Output

```
(4)+(6)*i
(2)+(-2)*i
```

Define operator << for
left operand = cout (class `ostream`)
right operand = `ComplexNumber`

Now you can use cout with your own class.

Overloaded << and >> operator

- Object appears on the right of the operators

```
cout << classObject
cin  >> classObject
```

- Overloaded **<<** needs object of the type **ostream &**
  - **Output stream**
- Overloaded **>>** needs object of the type **istream &**
  - **Input stream**
- Thus, both must be non-member functions – friend function

Commutative Operators

- Commutative operators
  - May want + to be commutative
    - So both "`a + b`" and "`b + a`" work
- Commutative for two different classes
  - Overloaded operator can only be member function when its class is left operant
    - `myClass + Long int`
    - Can be member function
  - But when its class object is right operant, it needs to be non-member overload function
    - `Long int + myClass`

Example 24-C: Complex Number

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

class ComplexNumber{
    public:
        double real;
        double imag;
        ComplexNumber(double,double);
        ComplexNumber operator+(const ComplexNumber &);
        ComplexNumber operator-(const ComplexNumber &);
};

ComplexNumber::ComplexNumber(double x = 0,double y = 0){
    cout << "("<< x << ")+(" << y << ")*i was created.\n";
    real = x; imag = y;
}

ComplexNumber ComplexNumber::operator+(const ComplexNumber &c){
    return ComplexNumber(real+c.real,imag+c.imag);
}

ComplexNumber ComplexNumber::operator-(const ComplexNumber &c){
    return ComplexNumber(real-c.real,imag-c.imag);
}

ostream & operator<<(ostream &os, const ComplexNumber &c){
    return os << "("<< c.real << ")+(" << c.imag << ")*i";
}
```

```cpp
int main(){
    ComplexNumber a(1,2);
    ComplexNumber c = a+1.5;
    ComplexNumber d = a-5;
    cout << c << "\n" << d;
}
```

Output

```
(1)+(2)*i was created.
(1.5)+(0)*i was created.
(2.5)+(2)*i was created.
(5)+(0)*i was created.
(-4)+(2)*i was created.
(2.5)+(2)*i
(-4)+(2)*i
```

Constructor `ComplexNumber(1.5)` was called when the program executed `a+1.5` since there was no `operator+(double)` defined (there was only `operator+(ComplexNumber)` defined).

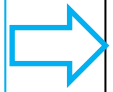
Example 24-C: Complex Number

```
30  int main(){
31      ComplexNumber a(1,2);
32      ComplexNumber b = 1.5+a;
33      ComplexNumber c = 5-a;
34      cout << b << "\n" << c;
35  }
```

[Error] no match for 'operator+' (operand types are 'double' and 'ComplexNumber')

Define non-member functions or friend functions
For double + ComplexNumber and double - ComplexNumber

```
40  ComplexNumber operator+(double s,const ComplexNumber &c){
41      return ComplexNumber(s+c.real,c.imag);
42  }
43
44  ComplexNumber operator-(double s,const ComplexNumber &c){
45      return ComplexNumber(s-c.real,-c.imag);
46  }
```



Output

```
(1)+(2)*i was created.
(2.5)+(2)*i was created.
(4)+(-2)*i was created.
2.5+2*i
4-2*i
```

Example 24-D

```
5  class Item{
6      public:
7          int hpmax;
8          int atk;
9          int def;
10         Item(int,int,int);
11 };
12
13 class Unit{
14     private:
15         string name;
16         int hpmax;
17         int atk;
18         int def;
19     public:
20         Unit(string,int,int,int);
21         void operator+=(Item &);
22         friend ostream & operator<<(ostream &, Unit &);
23 };
```

```
25 Item::Item(int a,int b ,int c){
26     hpmax = a; atk = b; def = c;
27 }
28
29 Unit::Unit(string s,int a,int b,int c){
30     name = s; hpmax = a; atk = b; def = c;
31 }
32
33 void Unit::operator+=(Item &x){
34     hpmax += x.hpmax;
35     atk +=  x.atk;
36     def +=  x.def;
37 }
38
39 ostream & operator<<(ostream &os, Unit &u){
40     return os << u.name << ": HPMAX = " << u.hpmax << " ATK = " << u.atk << " DEF = " << u.def << "\n";
41 }
```

```
46 int main(){
47     Unit hero("Prayath",10,5,5);
48     cout << hero;
49     Item beer(5,1,1), mala(0,2,0);
50     hero+=beer;
51     cout << hero;
52     hero+=mala;
53     cout << hero;
54     hero+=beer;
55     cout << hero;
56     return 0;
57 }
```

Output

```
Prayath: HPMAX = 10 ATK = 5 DEF = 5
Prayath: HPMAX = 15 ATK = 6 DEF = 6
Prayath: HPMAX = 15 ATK = 8 DEF = 6
Prayath: HPMAX = 20 ATK = 9 DEF = 7
```

261102
Computer Programming

Lecture 25: Containers

Containers

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.

The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers). คล้ายๆน้อยเต้อ

Containers replicate structures very commonly used in programming: dynamic arrays (vector), queues (queue), stacks (stack), heaps (priority_queue), linked lists (list), trees (set), associative arrays (map)...

http://www.cplusplus.com/reference/stl/

Containers

Sequence containers:

array C++11	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list C++11	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set C++11	Unordered Set (class template)
unordered_multiset C++11	Unordered Multiset (class template)
unordered_map C++11	Unordered Map (class template)
unordered_multimap C++11	Unordered Multimap (class template)

http://www.cplusplus.com/reference/stl/

Array

- *Automatic Data*: Automatically created at function entry, resides in activation frame of the function, and is destroyed when returning from function
- Can be allocated at both compile time and run time.
- Contiguous memory

```
1   #include<iostream>
2   using namespace std;
3
4   int main(){
5       int myArray[] = {1,2,3,4,5};
6
7       for(int i = 0; i < 5; i++){
8           cout << &myArray[i] << "\n";
9       }
10
11      return 0;
12  }
```

Output

```
0x22fe30
0x22fe34
0x22fe38
0x22fe3c
0x22fe40
```

Array

Insert new element

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int myArray[] = {1,2,3,4,5};
6
7      return 0;
8  }
```

Can not insert more elements.

(The number of elements = size)

แทรกค่า

Output

```
1 2 69 3 4 5
55 1 2 69 3 4
```

```
1  #include<iostream>
2  using namespace std;
3
4
5  void insert(int a[],int size, int data,int idx){
6      for(int i = size; i > idx; i--){
7          a[i] = a[i-1];
8      }
9      a[idx]=data;
10 }
11
12 int main(){
13     int myArray[10] = {1,2,3,4,5};
14
15     insert(myArray,5,69,2);
16     for(int i = 0; i < 6 ;i ++) cout << myArray[i] << " ";
17     cout << "\n";
18
19     insert(myArray,6,55,0);
20     for(int i = 0; i < 7 ;i ++) cout << myArray[i] << " ";
21     cout << "\n";
22
23     return 0;
24 }
```

Create your own insert function

Need to prepare over-size array

(size of 10 for 5 elements)

Dynamic Array

- *Dynamic Data*: Explicitly allocated and deallocated during program execution by C++ instructions written by programmer
- *Dynamic Allocation*: Allocation of memory space at run time.
- Contiguous memory
- Can not use Initializer กำหนดค่าเริ่มต้นไม่ได้
- `std::vector` is implementation of dynamic arrays

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

Output

```
0x5376f0
0x5376f4
0x5376f8
0x5376fc
```

std::Vector

• Vector is a class of sequence container representing array that can change in size.

• Vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements.

Vectors

• Vectors use a dynamically allocated array to store their elements. Vector need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

http://www.cplusplus.com/reference/vector/vector/

std::Vector

- Adding an element to a vector :

vectorName`.push_back(`*value*`);` เอาค่าไปต่อท้าย

(add new element to the end of vector)

vectorName`.insert(`*position*`,` *value*`);` แทรก

(use *vectorName*`.begin()` to obtain position of the 1st element)

- Removing element(s) of a vector :

vectorName`.pop_back();` เอาตัวท้ายออก

(removes the last element in the vector)

vectorName`.erase(`*position*`);`

vectorName`.erase(`*firstPosition*`,`*lastPosition*`);`

vectorName`.clear();`

(removes all elements from the vector)

http://www.cplusplus.com/reference/vector/vector/

std::Vector

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

int main(){
    vector<int> myVector(2);
    myVector[0] = 4; myVector[1] = 10; // 4 10

    myVector.push_back(55); // 4 10 55

    myVector.insert(myVector.begin()+2,3); // 4 10 3 55
    myVector.insert(myVector.begin(),8); // 8 4 10 3 55

    myVector.erase(myVector.begin()+1); // 8 10 3 55

    for(int i = 0; i < myVector.size(); i++){
        cout << &myVector[i] << ": " << myVector[i] << "\n";
    }

    return 0;
}
```

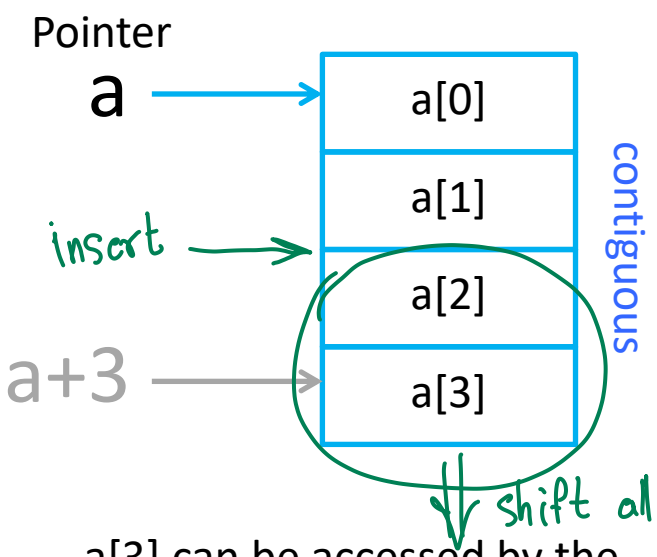
Output

```
0x617bf0: 8
0x617bf4: 10
0x617bf8: 3
0x617bfc: 55
```

Link List

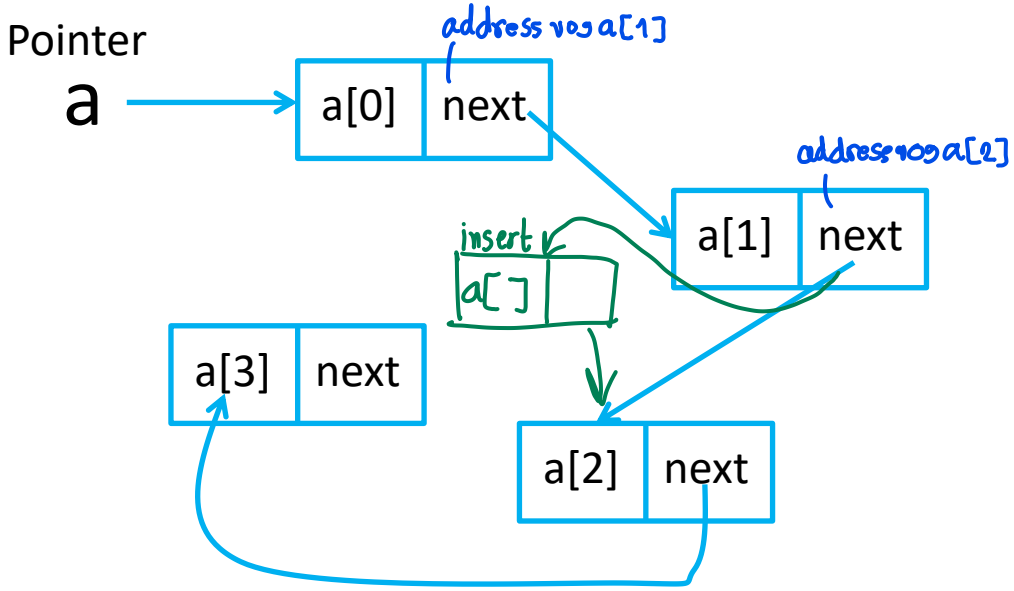
- Sequence Containers (same as array, vector)
- Non-contiguous memory
- Each element not only stores data but it also stores pointer to the next element.
- Cannot randomly access elements

Array



a[3] can be accessed by the pointer a+3

Link List



To access a[3], we have to check the pointers from a[0] to a[1], a[1] to a[2] and a[2] to a[3]

Link List

root → data

root → next → data

struct Node


```
5  struct Node{
6      int data;
7      Node *next;
8  };
9
10 class List{
11     public:
12         Node *root;
13         int size;
14         void show();
15         void append(int);
16         void insert(int,int);
17         void remove(int);
18 };
```

Data to be stored in each element (Node)

Pointer that point to the next element

Pointer to the first element (root)

The current number of the elements in List

The functions that class `List` can do

Link List

Adding root node

1 New Node

n→ | data 5 | next NULL |

2 Pointer

root → | data 5 | next NULL |

set root pointer to the new node

```
43 int main(){
44     List myList = {0,0};
45     myList.append(5);
46     myList.append(7);
47     myList.append(11);
48     myList.append(4);
49     myList.append(12);
50     myList.append(45);
51     myList.show();  cout << "\n";
52
53     return 0;
54 }
```

Output

```
5 7 11 4 12 45
```

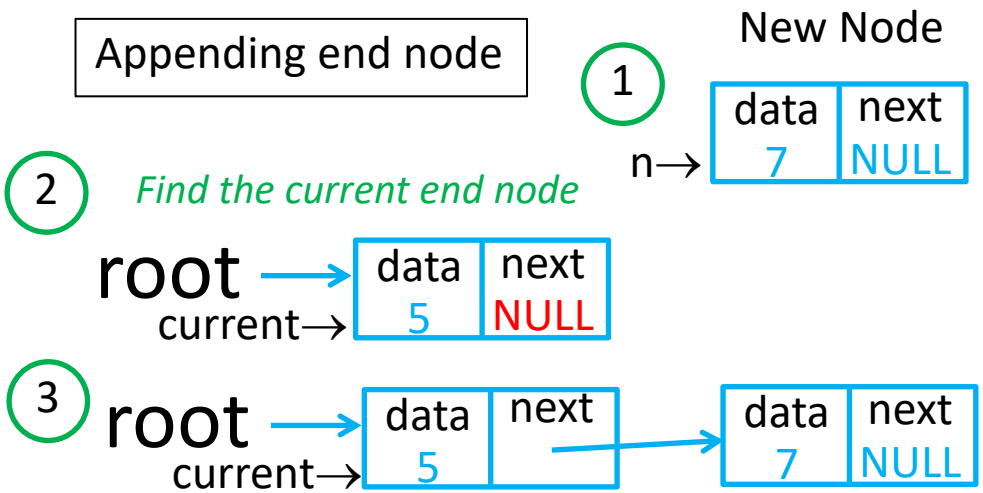
```
5  struct Node{
6      int data;
7      Node *next;
8  };
9
10 class List{
11     public:
12         Node *root; = 0   เริ่มต้น
13         int size; = 0
14         void show();
15         void append(int);
16         void insert(int,int);
17         void remove(int);
18 };
19
20 void List::show(){
21     Node *current = root;
22     cout << current->data << " ";
23     while(current->next){
24         current = current->next;
25         cout << current->data << " ";
26     }
27 }
28
29 void List::append(int d){
30     Node *n = new Node;
31     n->data = d; n->next = NULL;
32     if(root == NULL) root = n;
33     else{(ถ้ามีNodeอยู่แล้ว)
34         Node *current = root;
35         while(current->next){
36             current = current->next;
37         }
38         current->next = n;
39     }
40     size++;
41 }
```

สร้าง Node เพิ่มๆ

เชื่อมค่า

Link List

Appending end node



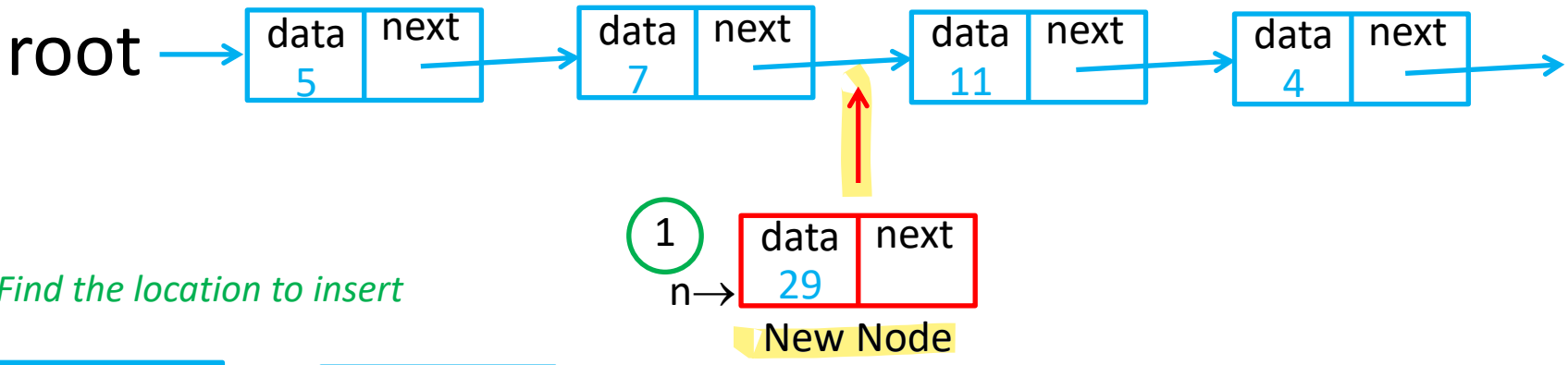

```
43  int main(){
44      List myList = {0,0};
45      myList.append(5);
46      myList.append(7);
47      myList.append(11);
48      myList.append(4);
49      myList.append(12);
50      myList.append(45);
51      myList.show();  cout << "\n";
52
53      return 0;
54  }
```

Output

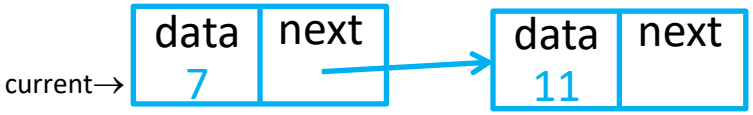
```
5 7 11 4 12 45
```

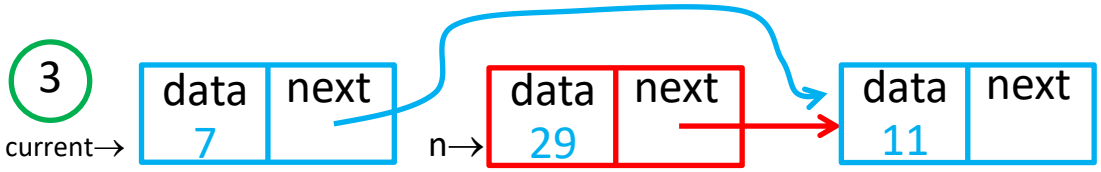
```
5   struct Node{
6       int data;
7       Node *next;
8   };
9
10  class List{
11      public:
12          Node *root;
13          int size;
14          void show();
15          void append(int);
16          void insert(int,int);
17          void remove(int);
18  };
19
20  void List::show(){
21      Node *current = root;
22      cout << current->data << " ";
23      while(current->next){
24          current = current->next;
25          cout << current->data << " ";
26      }
27  }
28
29  void List::append(int d){
30      Node *n = new Node;
31      n->data = d; n->next = NULL;
32      if(root == NULL) root = n;
33      else{
34          Node *current = root;
35          while(current->next){
36              current = current->next;
37          }
38          current->next = n;
39      }
40      size++;
41  }
```

Link List



2 *Find the location to insert*





Set pointer of new node to the next node

```
n->next = current->next
```

Set pointer of previous node to the new node

```
current->next = n
```

```
20  void List::insert(int d,int idx){
21      size++;
22      Node *n = new Node;
23      n->data = d;
24      if(idx == 0){
25          n->next = root;
26          root = n;
27          return;
28      }
29      Node *current = root;
30      for(int i = 0; i < idx-1;i++){
31          current = current->next;
32      }
33      n->next = current->next;
34      current->next = n;
35  }
```

Link List

```
10  class List{
11      public:
12          Node *root;
13          int size;
14          void show();
15          void append(int);
16          void insert(int,int);
17          void remove(int);
18  };
19
20  void List::insert(int d,int idx){
21      size++;
22      Node *n = new Node;
23      n->data = d;
24      if(idx == 0){
25          n->next = root;
26          root = n;
27          return;
28      }
29      Node *current = root;
30      for(int i = 0; i < idx-1;i++){
31          current = current->next;
32      }
33      n->next = current->next;
34      current->next = n;
35  }
```

```
76  int main(){
77      List myList = {0,0};
78      myList.append(5);
79      myList.append(7);
80      myList.append(11);
81      myList.append(4);
82      myList.append(12);
83      myList.append(45);
84      myList.show();  cout << "\n";
85
86
87      myList.insert(29,2);
88      myList.show();  cout << "\n";
89      myList.insert(33,3);
90      myList.show();  cout << "\n";
91      myList.insert(77,5);
92      myList.show();  cout << "\n";
93      myList.insert(69,0);
94      myList.show();  cout << "\n";
95
96      return 0;
97  }
```

Output

```
5 7 11 4 12 45
5 7 29 11 4 12 45
5 7 29 33 11 4 12 45
5 7 29 33 11 77 4 12 45
69 5 7 29 33 11 77 4 12 45
```

Link List

```
10  class List{
11      public:
12          Node *root;
13          int size;
14          void show();
15          void append(int);
16          void insert(int,int);
17          void remove(int);
18  };
19
20
21  void List::remove(int idx){
22      size--;
23
24
25
26
27
28
29
30
31
32
33
34  }
35
```

Write by yourself in LAB

```
77  int main(){
78      List myList = {0,0};
79      myList.append(5);
80      myList.append(7);
81      myList.append(11);
82      myList.append(4);
83      myList.append(12);
84      myList.append(45);
85
86      myList.insert(29,2);
87      myList.insert(33,3);
88      myList.insert(77,5);
89      myList.insert(69,0);
90      myList.show();  cout << "\n";
91
92      myList.remove(2);
93      myList.show();  cout << "\n";
94      myList.remove(4);
95      myList.show();  cout << "\n";
96      myList.remove(0);
97      myList.show();  cout << "\n";
98
99      return 0;
100 }
```

Output

```
69 5 7 29 33 11 77 4 12 45
69 5 29 33 11 77 4 12 45
69 5 29 33 77 4 12 45
5 29 33 77 4 12 45
```

Link List

```
10  class List{
11      public:
12          Node *root;
13          int size;
14          void show();
15          void append(int);
16          void insert(int,int);
17          void remove(int);
18          ~List();
19  };
20
21  List::~List(){
22      Node *current = root;
23      while(current->next){
24          Node *p = current;
25          current = current->next;
26          cout << p->data << " was deleted." << "\n";
27          delete p;
28      }
29      cout << current->data << " was deleted." << "\n";
30      delete current;
31  }
32
```

Define destructor to delete all nodes when the list is destroyed.

```
90  int main(){
91      List myList = {0,0};
92      myList.append(5);
93      myList.append(7);
94      myList.append(11);
95      myList.append(4);
96      myList.append(12);
97      myList.append(45);
98
99      myList.insert(29,2);
100     myList.insert(33,3);
101     myList.insert(77,5);
102     myList.insert(69,0);
103
104     myList.remove(2);
105     myList.remove(4);
106     myList.remove(0);
107     myList.show();  cout << "\n";
108
109     return 0;
110 }
```

Output

```
5 29 33 77 4 12 45
5 was deleted.
29 was deleted.
33 was deleted.
77 was deleted.
4 was deleted.
12 was deleted.
45 was deleted.
```

Link List

```
66  void List::show(){
67      Node *current = root;
68      cout << current << ": " << current->data << "\n";
69      while(current->next){
70          current = current->next;
71          cout << current << ": " << current->data << "\n";
72      }
73  }
```

```
90  int main(){
91      List myList = {0,0};
92      myList.append(5);
93      myList.append(7);
94      myList.append(11);
95      myList.append(4);
96      myList.append(12);
97      myList.append(45);
98
99      myList.insert(29,2);
100     myList.insert(33,3);
101     myList.insert(77,5);
102     myList.insert(69,0);
103
104     myList.remove(2);
105     myList.remove(4);
106     myList.remove(0);
107     myList.show();  cout << "\n";
108
109     return 0;
110 }
```

Output

Addresses are not contiguous

```
0x2f76d0: 5
0x2f7c30: 29
0x2f7c50: 33
0x2f7c70: 77
0x2f7bd0: 4
0x2f7bf0: 12
0x2f7c10: 45
```

std::List

ข้อดี

• Lists are sequence containers that allow constant time insert and erase operations anywhere within the sequence, and iteration in both directions.

• Compared to other base standard sequence containers (array, vector), lists perform generally better in inserting, extracting and moving elements in any position within the container for which an iterator has already been obtained, and therefore also in algorithms that make intensive use of these, like sorting algorithms.

• The main drawback of lists compared to these other sequence containers is that they lack direct access to the elements by their position

ข้อเสีย

http://www.cplusplus.com/reference/list/list/

std::List

– Adding an element to a list :

listName`.push_back(`*value*`);` (Add element at the end)

listName`.push_front(`*value*`);` (Insert element at beginning)

listName`.insert(`*position*`,` *value*`);`

(use *listName*`.begin()` to obtain position of the 1st element)

– Removing element(s) of a list :

listName`.pop_back();` ลบตัวสุดท้าย (Delete last element)

listName`.pop_front();` ลบตัวแรก (Delete first element)

listName`.erase(` *position* `);`

listName`.erase(` *firstPosition*,*lastPosition* `);`

listName`.clear();` (removes all elements from the vector)

http://www.cplusplus.com/reference/list/list/

std::List

```
for ( list<int>:: iterator i = myList.begin() ; i ≠ myList.end() ; i++ ){
    cout << *i ;
}
```

```
vector<int> v ;
list <int> v ;
```

```
1  #include<iostream>
2  #include<list>
3  using namespace std;
4
5  int main(){
6      list<int> myList;
7      myList.push_back(4); // 4
8      myList.push_back(5); // 4 5
9      myList.push_front(10); // 10 4 5
10     myList.push_front(69); // 69 10 4 5
11
12     list<int>::iterator current =  myList.begin();
13     myList.insert(current,74); // 74 69 10 4 5
14     for(int i = 0; i < 2;i++) current++;
15     myList.insert(current,41); // 74 69 10 41 4 5
16
17     myList.erase(current); // 74 69 10 41 5
18     current =  myList.begin();  current++;
19     myList.erase(current); // 74 10 41 5
20
21     while (!myList.empty()){
22         cout << myList.front() << " ";
23         myList.pop_front();
24     }
25
26     return 0;
27 }
```

List หลายเรื่องๆ

เหมือน List เฉยๆ

Output

```
74 10 41 5
```

std::Set

สมาชิกไม่ซ้ำกัน

- Sets are containers that store unique elements following a specific order.

- The value of the elements in a set cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.

```
1  #include<iostream>
2  #include<string>
3  #include<set>
4  using namespace std;
5
6  int main(){
7      set<string> mySet;
8      mySet.insert("Alice"); // Alice
9      mySet.insert("Oscar"); // Alice Oscar
10     mySet.insert("Eva"); // Alice Eva Oscar
11     mySet.insert("Charlie"); // Alice Charlie Eva Oscar
12
13     mySet.insert("Oscar"); // Alice Charlie Eva Oscar
14
15     set<string>::iterator  i;
16     for (i = mySet.begin(); i != mySet.end(); i++){
17         cout << *i << "\n";
18     }
19                          mySet.size()
20     return 0;
21 }
```

Output

The elements are sorted and unique

```
Alice
Charlie
Eva
Oscar
```

http://www.cplusplus.com/reference/set/set/