261102
Computer Programming

Lecture 18: Dynamic Memory Allocation

Types of Program Data

Global variable

- *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

จองหน่วยความจำ คืนหน่วยความจำ ให้ระบบ

Allocation of Memory

int x;
int *i; int y[100];

- *Static Allocation*: Allocation of memory space at compile time.

  ตอน ตอน php

- *Dynamic Allocation*: Allocation of memory space at run time.

  ตอน compile ยังไม่กำหนดตำแหน่ง

  ตอน รัน exe. → ถึงตอนนั้น

Dynamic Memory Allocation

- Dynamic allocation is useful when
  - arrays need to be created whose extent is not known until run time
    จำนวน
    int x [1000]; เปลี่ยน
  - 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
    class

Dynamic Memory Allocation

คำสั่ง

- Pointers need to be used for dynamic allocation of memory

จอง mem

- Use the operator **`new`** to dynamically allocate space

คืน mem

- Use the operator **`delete`** to later free this space

The `new` operator

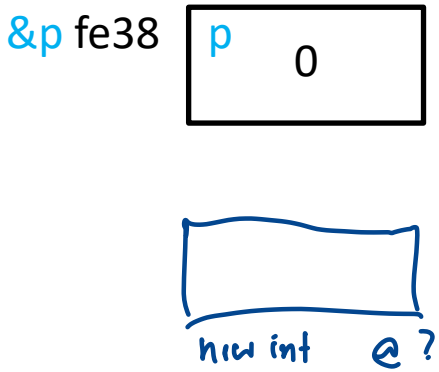
- If memory is available, the `new` operator allocates memory space for the requested object/array, and returns a pointer to (address of) the memory allocated.
- If sufficient memory is NOT available, the `new` operator returns `NULL`.
- The dynamically allocated object/array exists until the `delete`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

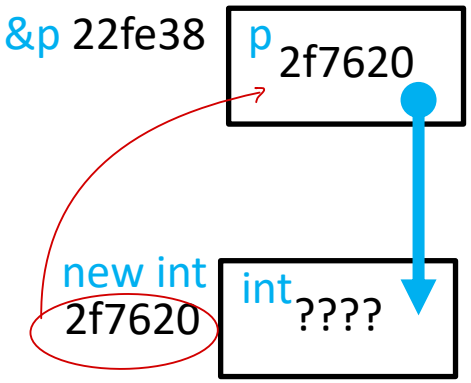
```
#include<iostream>
using namespace std;

int main(){

    int *p = NULL;
    cout << "p = " << p << "\t\t&p = " << &p << "\n";

    p = new int;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    *p = 12;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    delete p;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    p = NULL;
    cout << "p = " << p << "\t\t&p = " << &p ;

    return 0;
}
```




```
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

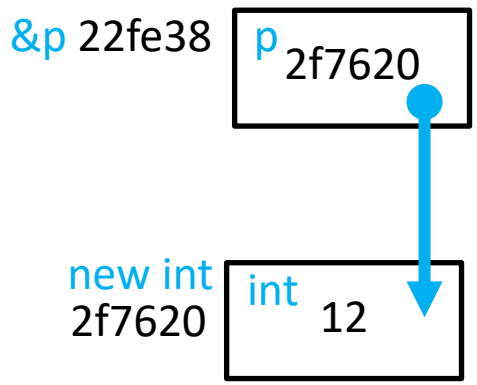
```
#include<iostream>
using namespace std;

int main(){

    int *p = NULL;
    cout << "p = " << p << "\t\t&p = " << &p << "\n";

    p = new int;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    *p = 12;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    delete p;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    p = NULL;
    cout << "p = " << p << "\t\t&p = " << &p ;

    return 0;
}
```




```
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  }
```

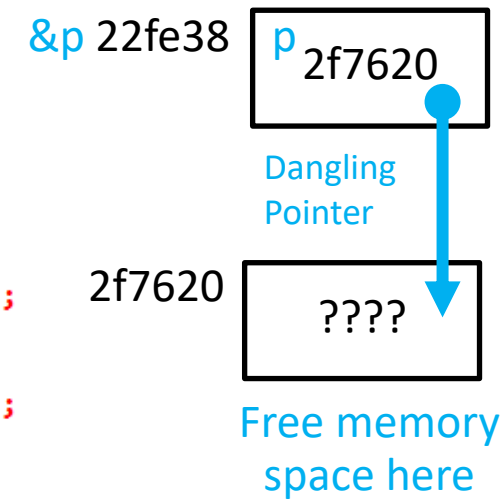



```
p = 0              &p = 0x22fe38
p = 0x2f7620       &p = 0x22fe38    *p = 3111712
p = 0x2f7620       &p = 0x22fe38    *p = 12
p = 0x2f7620       &p = 0x22fe38    *p = 3111712
p = 0              &p = 0x22fe38
```

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  }
```

pointer




```
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

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

int main(){

    int *p = NULL;
    cout << "p = " << p << "\t\t&p = " << &p << "\n";

    p = new int;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    *p = 12;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    delete p;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    p = NULL;
    cout << "p = " << p << "\t\t&p = " << &p ;

    return 0;
}
```

&p 22fe38 | p | 0

2f7620 | ????

```
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  }
```

ผิด , ต้องเขียนแยก

delete p;

delete q;

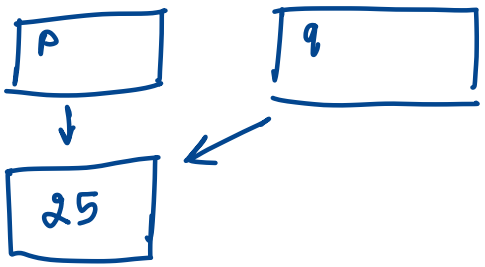
Output

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

Dynamic Memory Allocation

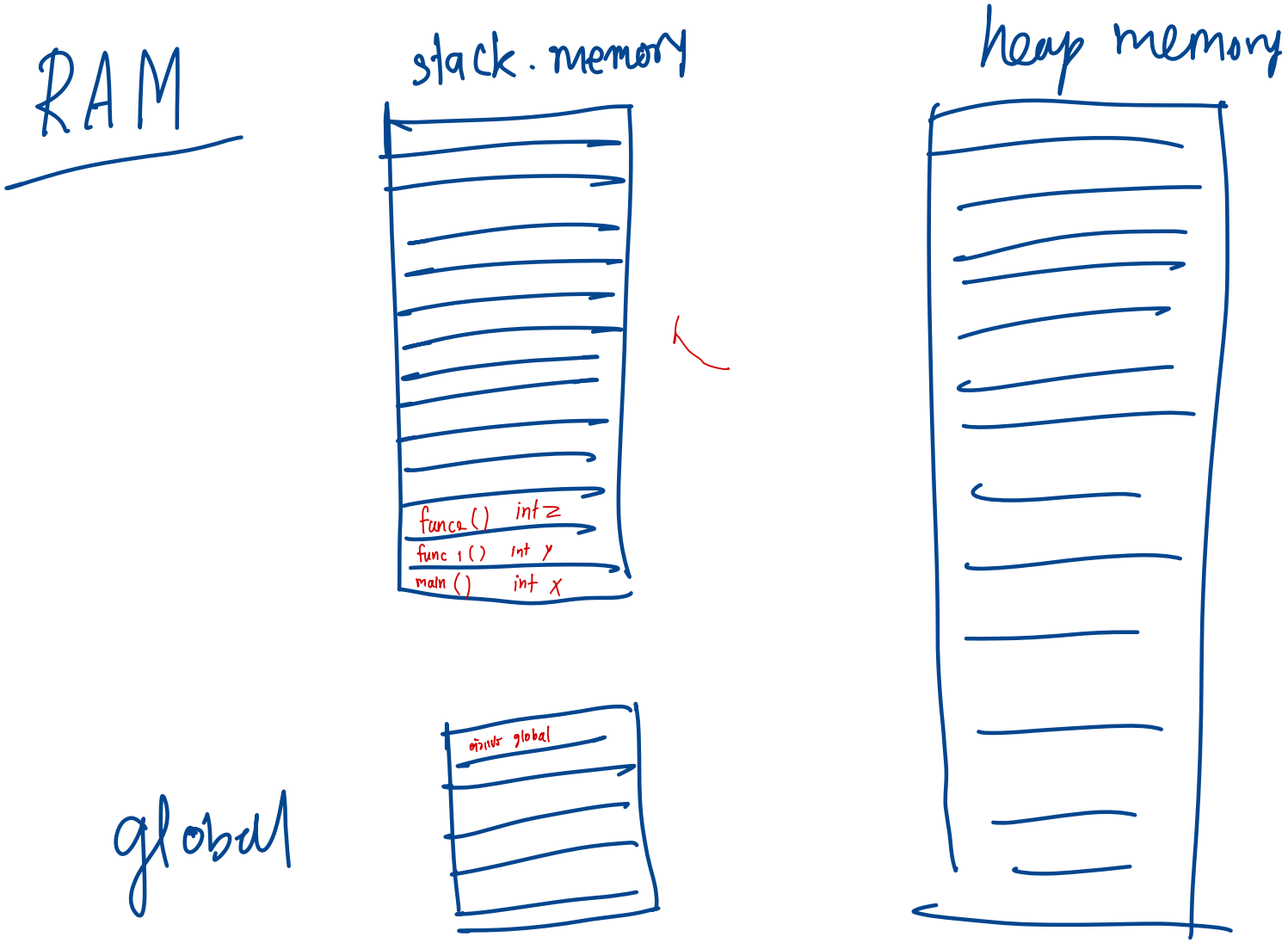
```
#include<iostream>
using namespace std;

int main(){

    int *p = new int;
    *p = 12;
    cout << "p = " << p << "\t&p = " << &p << "\t*p = " << *p << "\n";

    delete p;

    int *q = new int;
    *q = 25;
    cout << "q = " << q << "\t&q = " << &q << "\t*q = " << *q << "\n";
    cout << "*p = " << *p << "\n";

    delete q;

    return 0;
}
```

ค่าไม่ใช่ 25



Output

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


RAM
stack. memory
heap memory
func2() int z
func1() int y
main() int x
global
ตัวแปร global

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++.

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`**)

(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

```
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
```

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

int main()
{
    int N;
    cout << "Input number of your data: ";
    cin >> N;
    float *c = new float[N];  //float c[N];
    for(int i=0;i<N;i++){
        cout << "Input your data [" << i+1 << "]: ";
        cin >> c[i];
    }

    cout << "Your data = ";
    for(int i=0;i<N;i++) cout << c[i] << " ";

    delete [] c; //Don't forget to free memory

    return 0;
}
```

float *c;

c = new float[10]

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;
```

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];
```

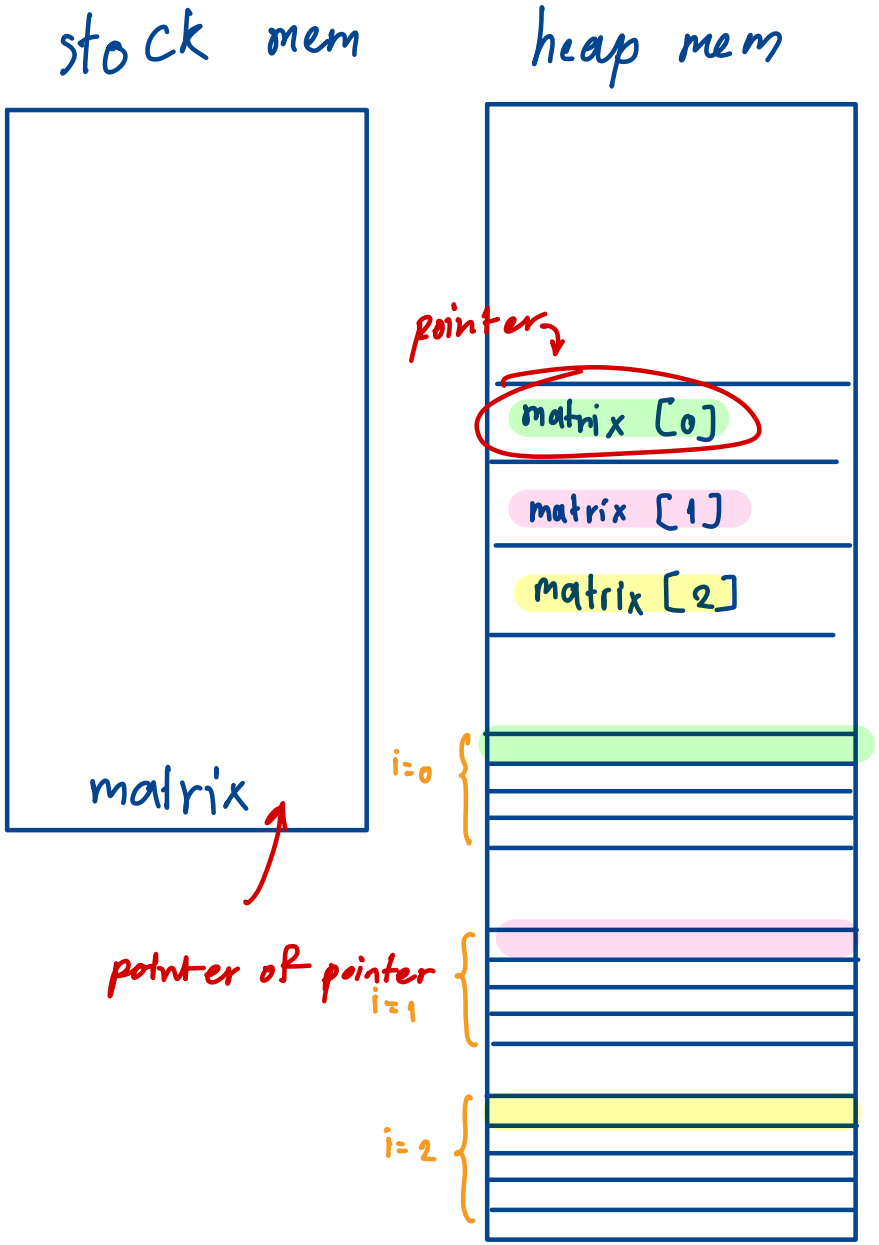
assign ตำแหน่งใน RAM ไม่เรียงกัน

```
int **matrix;
matrix = new int*[3]
for (int i=0 ; i < 3 ; i++)
    metrix[i] = new int[4]
```

ตัวคอลัมน์

matrix

แถวที่ 0
แถวที่ 1
แถวที่ 2



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) ไม่เรียงติดกัน
- 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

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  }
```

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
```

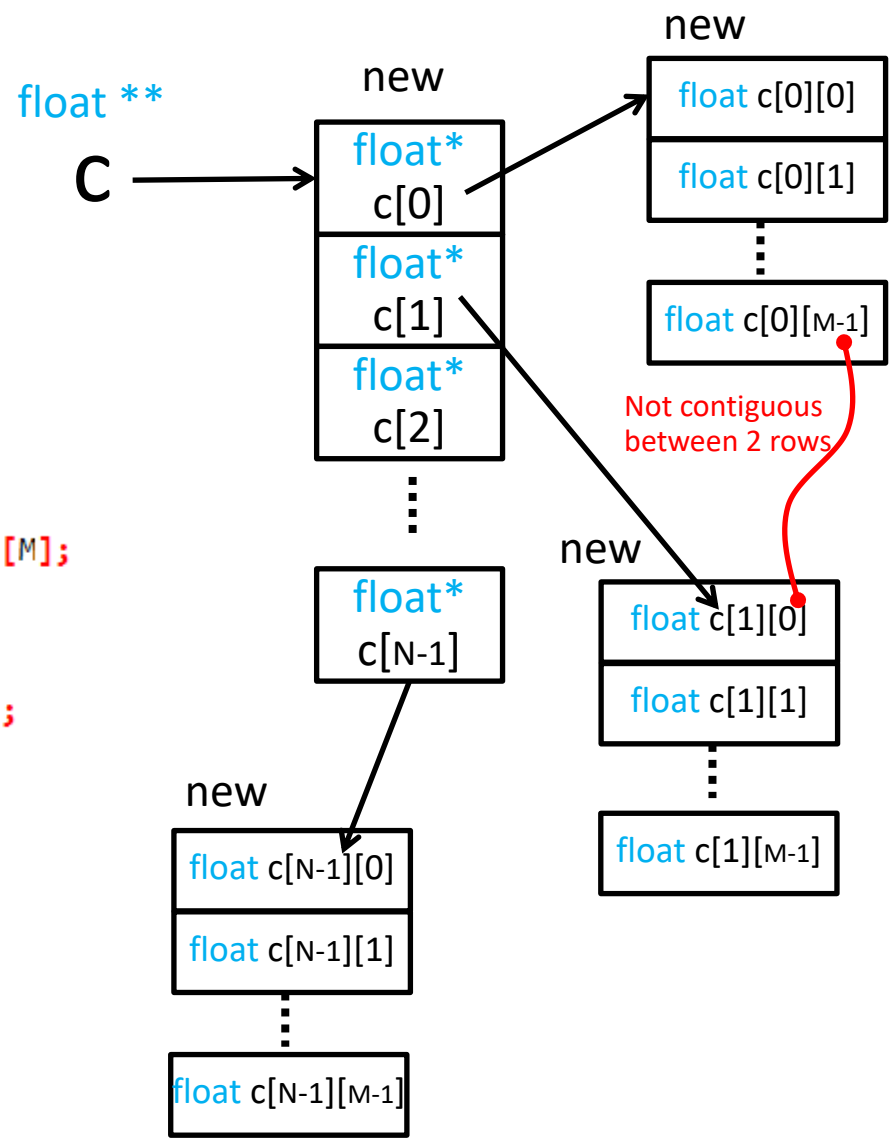
```
#include <iostream>
using namespace std;

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

    float **c = new float *[N];
    for(int i = 0; i<N; i++) c[i] = new float[M];


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

    return 0;
}
```

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

Deallocation of 2D Array

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

```
float **c = new float *[N];
for(int i = 0; i<N; i++) c[i] = new float[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";
}


for(int i=0; i<N; i++) delete [] c[i];
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

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

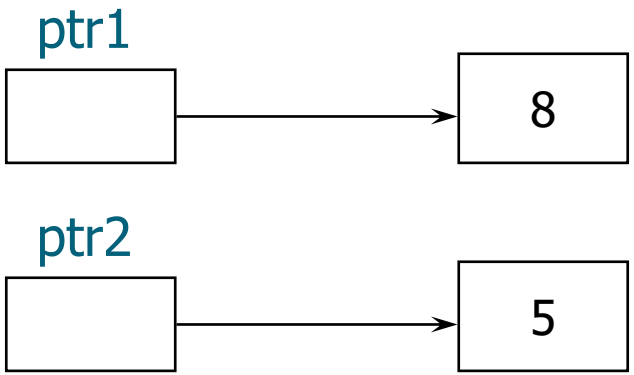
```
Input size of your matrix: 5 5
Input row [1]: 1 2 3 4 5
Input row [2]: 6 7 8 9 10
Input row [3]: 4 5 7 8 3
Input row [4]: 1 1 2 2 2
Input row [5]: 1 4 7 5 5
0x22fd40 0x22fd44 0x22fd48 0x22fd4c 0x22fd50
0x22fd54 0x22fd58 0x22fd5c 0x22fd60 0x22fd64
0x22fd68 0x22fd6c 0x22fd70 0x22fd74 0x22fd78
0x22fd7c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

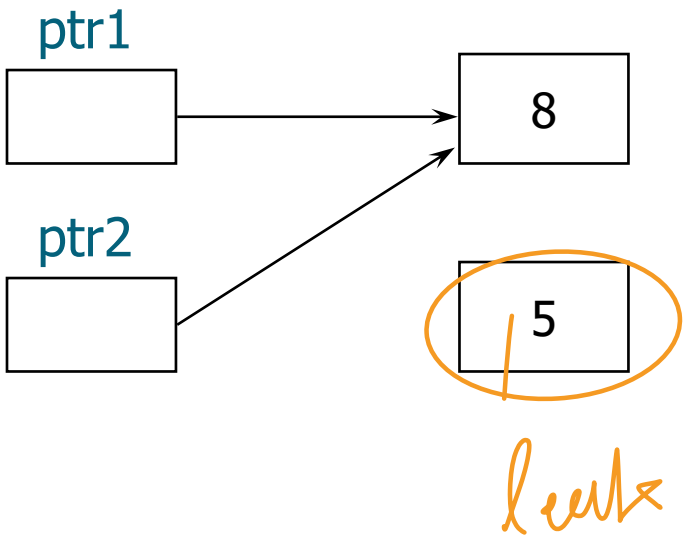
Memory Leaks

- 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

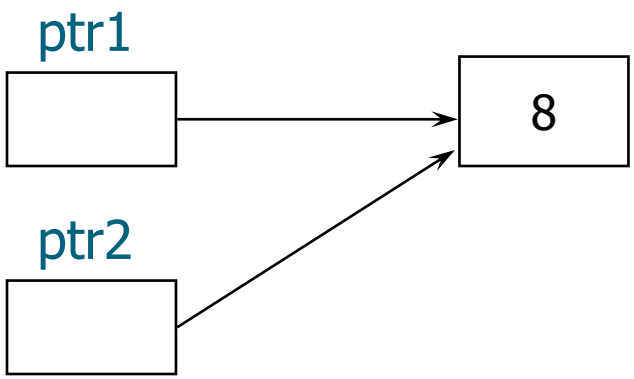
- An inaccessible object is an unnamed object that was created by operator `new` and which a programmer has left without a pointer to it.
- It is a logical error and causes memory leaks.

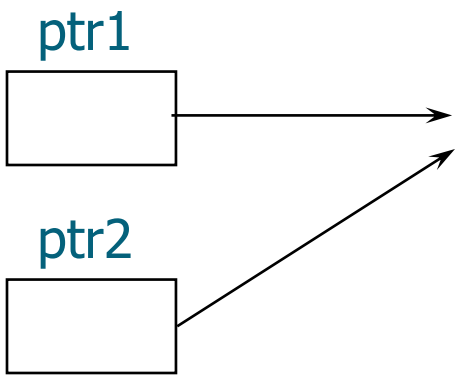
Dangling Pointer :

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

Dangling Pointer

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





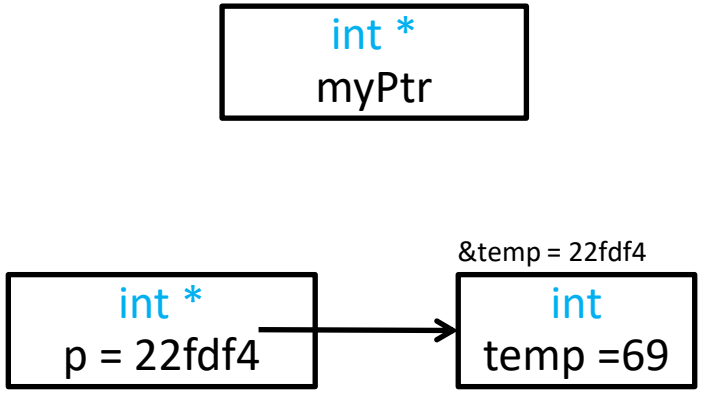
How to avoid?

ptr1 = 0
ptr2 = 0

Dangling Pointer

```
1   #include <iostream>
2   using namespace std;
3
4   int * become69(){
5       int *p;  local
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

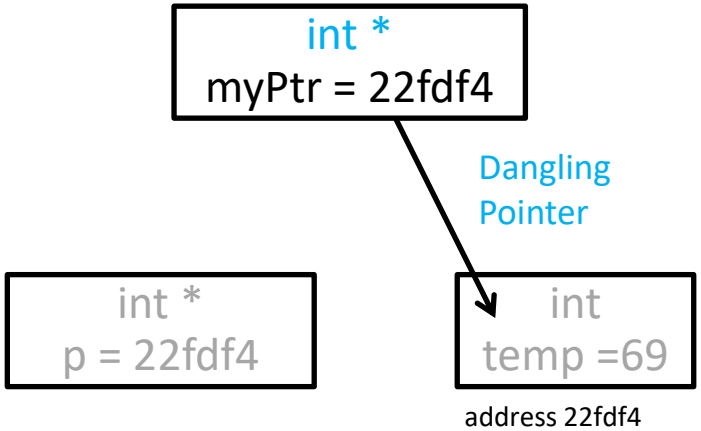



```
p = 0x22fdf4
myPtr = 0x22fdf4
*myPtr = 0
```

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

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

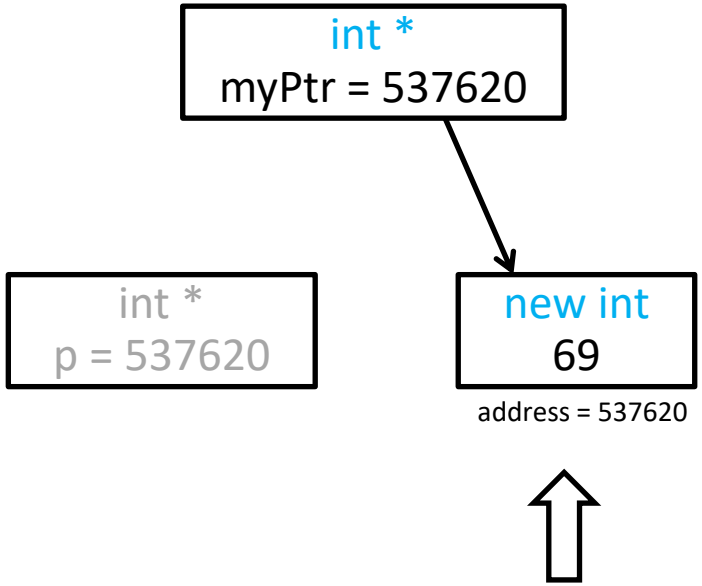
Dangling Pointer

แก้ ?

Dynamic Data

```
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  }
```

ออกมาแล้วไม่ตาย



Before `delete` operator

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

Dangling Pointer

ไม่ 2

Static Data

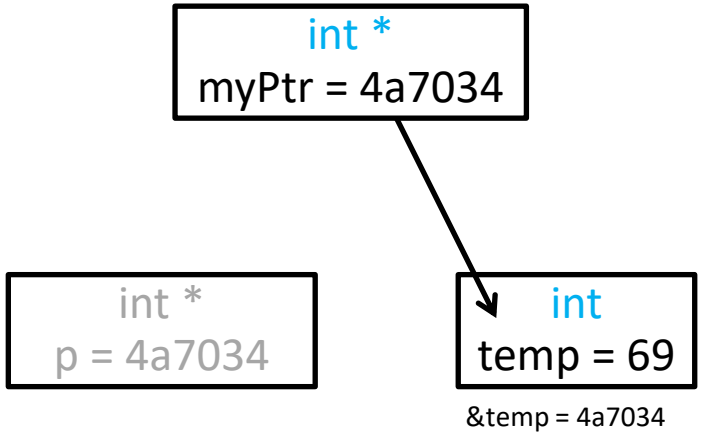
≈ global variable

ใช้ ที่ new

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

int * become69(){
    int *p;
    static int temp;
    p = &temp;
    temp = 69;
    cout << "p = " << p << "\n";
    return p;
}

int main(){
    int *myPtr;
    myPtr = become69();
    cout << "myPtr = " << myPtr << "\n";
    cout << "*myPtr = " << *myPtr << "\n";
}
```

Puay : stack

แม่ temp ไม่ตาย




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

`p` is destroyed after leaving function

`temp` is not destroyed after leaving function

```
void func() {
    static int x = 1;
    cout << x;
    x++;
}

int main() {
    func()
    func()
    func()
}
```

ถ้าไม่มี static

111

ถ้ามี static

123

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 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:

```
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 :

เพิ่ม

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

```
#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;
}
```

คล้าย pointer ชี้ไปตำแหน่ง แรก

แทรกเข้าไปตง. ที่ 2

Output

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

std::Vector

2 hr.

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

```
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  }
```

69 69 69 69 69 69 69

Passed by reference

$$C[i] \overset{*}{\equiv} (C + i)$$

love love