COMPUTER SCIENCE
CITY COLLEGE OF NEW YORK

CSC212
Data Structure
- Section FG

Lectures 6/7
Pointers and Dynamic Arrays

Instructor: Feng HU
Department of Computer Science
City College of New York

Why Pointers and Dynamic Memory

- Limitation of our bag class
  - bag::CAPACITY constant determines the capacity of every bag
  - wasteful (if too big) and hard to reuse (if too small)
    - need to change source code and recompile
- Solution:
  - provide control over size in running time
  - <= dynamic arrays
  - <= pointers and dynamic memory

Outline (Reading Ch 4.1 – 4.2)

- Pointers
  - *(asterisk) and &(ampersand) operators
- Dynamic Variables and new Operator
  - Dynamic Arrays and Dynamic Objects
  - Stack (local) vs. heap (dynamic) memory
- Garbage Collection and delete Operator
- Parameters revisited
  - Pointers and Arrays as Parameters

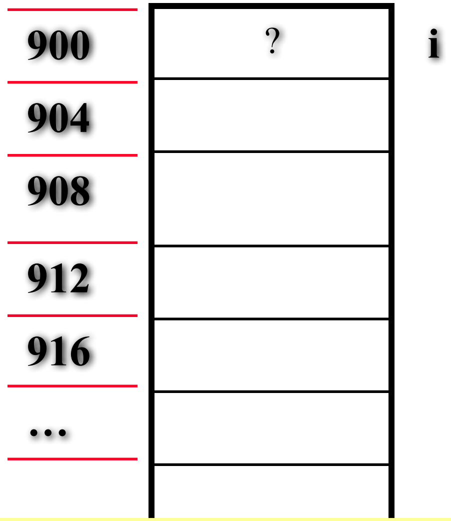
Pointer Variable

- First let’s have a look at local variables

```
int i;
```

By this declaration, a cell of 4 adjacent bytes (in some machines) are allocated in the local memory (called stack memory)

- Q: What’s the value of i?



900	?	i
904		
908		
912		
916		
…		

Address 9## is just for illustration.
Real address may have 64 bits

Pointer Variable

- First let's have a look at local variables

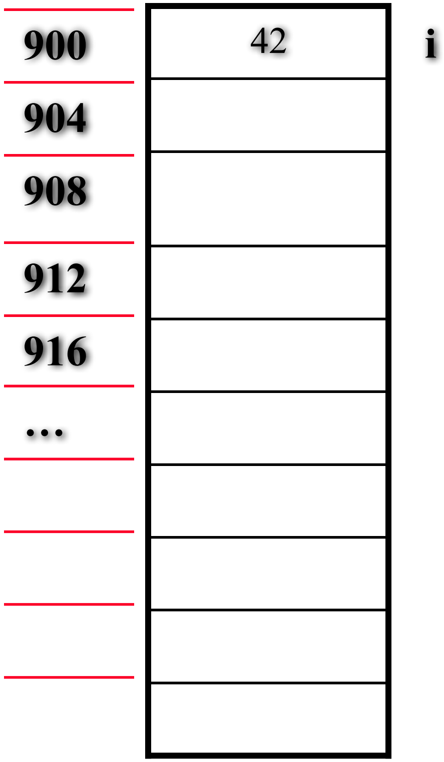
```
int i;

i = 42;
```

The assignment put number 42 in the cell. The memory address of the 1st byte is the address of the variable i

– the pointer to i

- Q: How to get the address?



900	42	i
904		
908		
912		
916		
…		

Pointer Variable

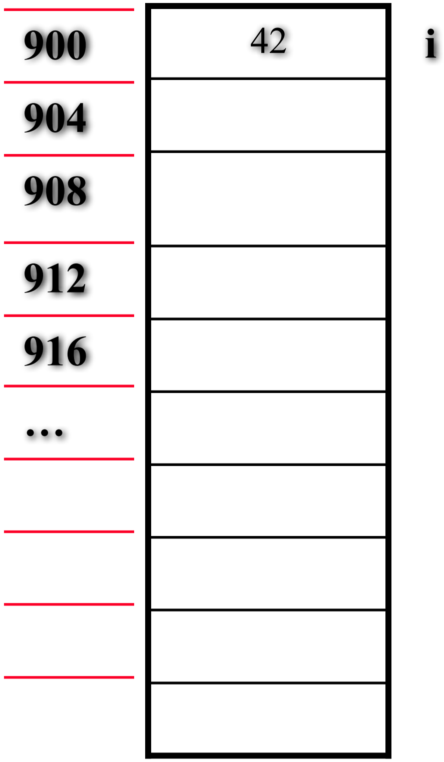
- First let’s have a look at local variables

```
int i;
i = 42;
cout << &i;
```

& (ampersand) operator
- “address of ” operator
- &i is 900 !

-Note: two meanings of &

- Q: Where can we store &i?



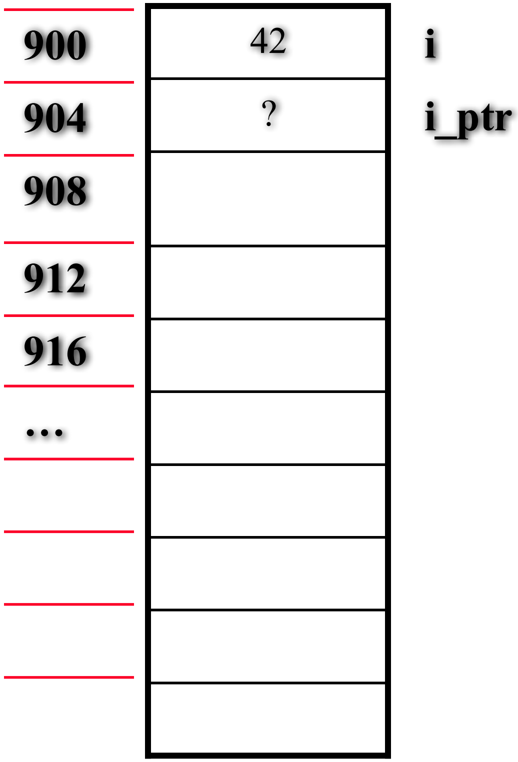
Pointer Variable

- The memory address can be stored a special pointer variable

```
int i=42;
int *i_ptr;
```

1. the type of the data that the pointer points to: int
2. an asterisk (*)
3. the name of the newly declared pointer: i_ptr

- Q: How to point i_ptr to i?



900	42	i
904	?	i_ptr
908		
912		
916		
…		

Pointer Variable

- Assign the address of i to i_ptr

```
int i=42;
int *i_ptr;
i_ptr = &i;
```

900	42	i
904	?	i_ptr
908		
912		
916		
…		

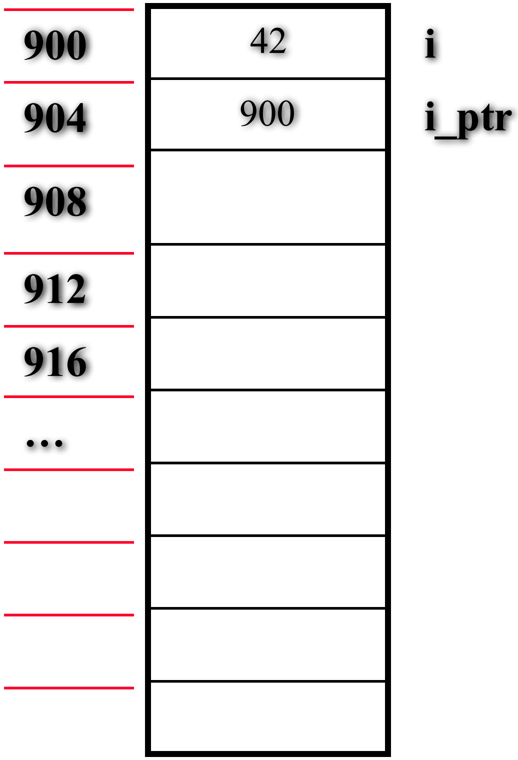
What are the results of

- cout << i;
- cout << i_ptr;
- cout << &i_ptr;

Pointer Variable

- The i_ptr holds the address of an integer, not the integer itself

```
int i=42;
int *i_ptr;
i_ptr = &i;
```



900	42	i
904	900	i_ptr
908		
912		
916		
…		

Two ways to refer to i

- cout << i;
- cout << *i_ptr;

- dereferencing operator *
- two meanings of *

Operators * and &

- Operator *
    - Pointer declaration

      ```
      int *i_ptr;
      ```
    - dereferencing operator

      ```
      cout << *i_ptr;
      ```
- Two different meanings!

- Operator &
    - Reference parameter

      ```
      void funct(int& i);
      ```
    - “address of ” operator

      ```
      i_ptr = &i;
      ```
- Just coincidence?
    - Will see in parameter passing

Syntax and Naming Issues

- How to declare two pointers in a line

  ```
  char *c1_ptr, *c2_ptr;
  ```

  - instead of

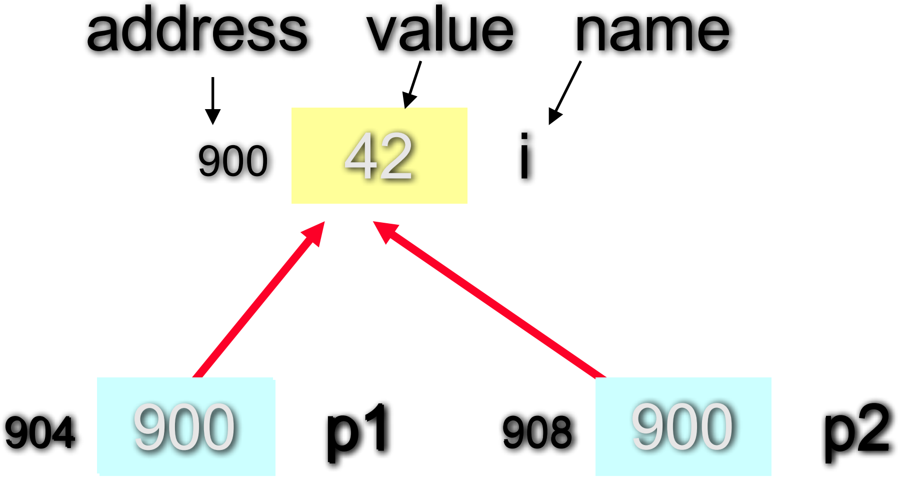
  ```
  char* c1_ptr, c2_ptr;
  ```

- For clarity, use _ptr or cursor for pointer variables

Assignment Operators with Pointers

- p2 = p1

```
int i = 42;
int *p1, *p2;
p1 = &i;
p2 = p1;
```



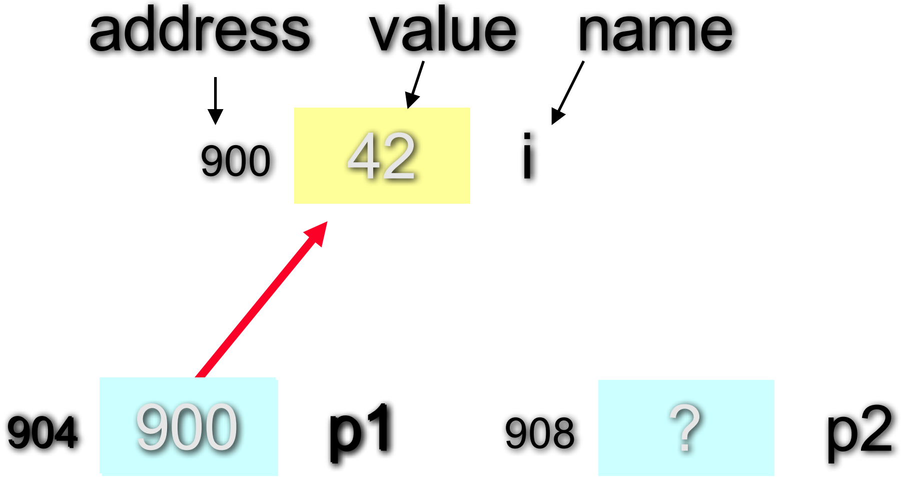
900	42	i
904	900	p1
908	900	p2
912		
916		
…		

Both p1 and p2 point to the same integer

Assignment Operators with Pointers

- *p2 = *p1

```
int i = 42;
int *p1, *p2;
p1 = &i;
*p2 = *p1; X
```



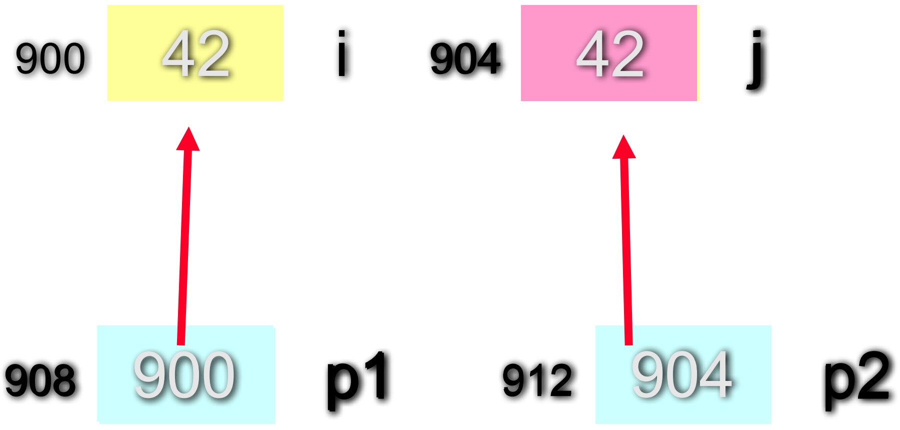
900	42	i
904	900	p1
908	?	p2
912		
916		
…		

p2 doesn't point to anywhere, so assigning value to *p2 will cause a running time error!

Assignment Operators with Pointers

- *p2 = *p1

```
int i = 42;
int j = 20;
int *p1, *p2;
p1 = &i;
p2 = &j;
*p2 = *p1;
```



900	42	i
904	42	j
908	900	p1
912	904	p2
916		
…		

Both i (*p1) and j (*p2) will have the same integer values

Outline (Reading Ch 4.1 – 4.2)

- Pointers
  - *(asterisk) and &(ampersand) operators
- Dynamic Variables and new Operator
  - Dynamic Arrays and Dynamic Objects
  - Stack (local) vs. heap (dynamic) memory
- Garbage Collection and delete Operator
- Parameters revisited
  - Pointers and Arrays as Parameters

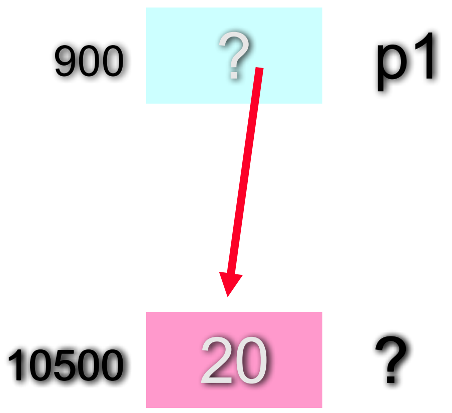
Dynamic Variables

- We cannot use a pointer if not initialized
  - need to point to a declared variable
- How to use a pointer without connecting with a declared ordinary variable?
  - Solution: Dynamic (allocated) variables
    - not declared, therefore no identifier
    - created during execution
  - Real power of pointers is with dynamic variables

The **new** Operator

- allocates memory and return a pointer

```
int *p1;
p1 = new int;
*p1 = 20;
```



900	10500	p1
904		
908		
…		
…		
10492		
10496		
10500	20	?

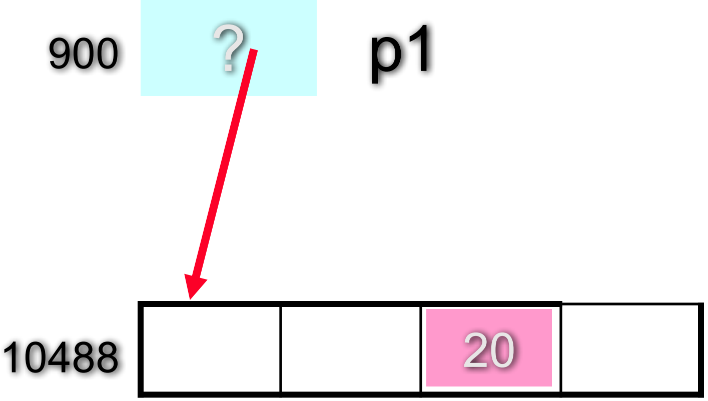
- p1 points to a dynamic integer variable without any identifier (name)

- dynamic memory comes from the programs' heap (free store)

Dynamic Arrays

- new can allocate an entire array all at once

```
int *p1;
p1 = new int[4];
p1[2] = 20;
cout<<*(p1+2);
```



- p1 points to 1st entry of dynamic array
- number of entries in a pair of sq. brackets
- two ways to access p1 (array or pointer)

900	10488	p1
904		
908		
…		
…		
10488		?
10492		
10496	20	
10500		

Accessing Dynamic Array

- Use array notation
  - the 1st entry

    p1[0] = 18;
  - the 3rd entry

    p1[2] = 20;
  - the ith entry

    p1[i-1] = 19;

- Use pointer notation
  - the 1st entry

    *p1 = 18;
  - the 3rd entry

    *(p1+2) = 20;
  - the ith entry

    *(p1+i-1) = 19;

A demo for pointers and dynamic arrays:
test_pointer.cxx

Dynamic Array Example:Quiz

- A program read ages of each of CCNY classes, with varying sizes, calculate the average, and then print out the average.

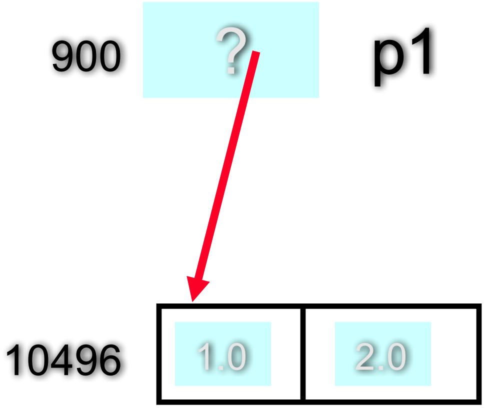
```
size_t size;
int *ages;
float average;

cin >> size;
ages = new int[size];
// input ages of all students
// calculate average
// print average
…
```

Dynamic Objects of a class

- new can also allocate a dynamic object

```
point *p1;
p1 = new point(1.0, 2.0);
cout<< (*p1).get_x();
cout<< p1->get_x();
```



900	10496	p1
904		
908		
…		
…		
10488		
10492		
10496	1.0	?
10500	2.0	

- p1 points to dynamic object without name

- parameters can be used as in declaration

- two ways to access p1 (* and ->)

Dynamic Object Arrays of a class

Q: Are the followings correct? point3 demo

- Ten points with default coordinates?

V
```
p1 = new point[10];
```

- Ten points with the same coordinates?

X
```
p1 = new point(1.0, 2.0)[10];
```

- Ten points on the x axis with interval 1?

V
```
p1 = new point[10];
for (i=0; i<10; i++)  p1[i].set(i, 0);
```

Assume we have a member function

```
void point::set(double x_init, double y_init);
```

Failure of the **new** Operator

- Dynamic memory via new operator comes from heap of a program
- Heap size from several K to 1GB, however fixed
- Could run out of room therefore cause a bad_alloc exception
  - error message and program halts
- Good practice 1: document which functions uses new
- Good practice 2: garbage collection by delete operator

Outline (Reading Ch 4.1 – 4.2)

- Pointers
  - *(asterisk) and &(ampersand) operators
- Dynamic Variables and new Operator
  - Dynamic Arrays and Dynamic Objects
  - Stack (local) vs. heap (dynamic) memory
- Garbage Collection and delete Operator
- Parameters revisited
  - Pointers and Arrays as Parameters

The **delete** Operator

- Release any dynamic memory (heap memory) that is no longer needed

```
int *i_ptr;
double *d_ptr;
point *p_ptr;

i_ptr = new int;
d_ptr = new double[20];
p_ptr = new point(1.0, 2.0);
… …
```

```
…
delete i_ptr;
delete [ ] d_ptr; // empty brackets
delete p_ptr;
```

Questions(true or false):

1. delete resets these pointers X
2. delete removes dynamic objects pointed by the pointers V
3. nothing happens to the pointers themselves V

Outline (Reading Ch 4.1 – 4.2)

- Pointers
  - *(asterisk) and &(ampersand) operators
- Dynamic Variables and new Operator
  - Dynamic Arrays and Dynamic Objects
  - Stack (local) vs. heap (dynamic) memory
- Garbage Collection and delete Operator
- Parameters revisited
  - Pointers and Arrays as Parameters

Pointers and Arrays as Parameters

- Value parameters that are pointers
- Array parameters
- Pointers and arrays as const parameters
- Reference parameters that are pointers

Value parameters that are pointers

- Compare ordinary and pointer variables

```
void print_int_42(int i)
{
    cout << i<<endl ;
    i = 42 ;
    cout << i <<endl;
}
```

```
void set_int_42(int*   i_ptr)
{
    cout << *i_ptr <<endl;
    *i_ptr = 42 ;
    cout << *i_ptr <<endl;
}
```

Calling program:

```
int m = 80;
print_int_42(m);      cout << m<<endl<<endl;
set_int_42(&m);       cout << m<<endl<<endl;
```

```
80
42
80

80
42
42
```

Array Parameters

- Compare ordinary and Dynamic arrays

```
Calling program:

int ages[30];

make_all_20(ages, 30);
```

```
Calling program:

int *ages;
ages = new int[30]
make_all_20(ages, 30);
```

```
void make_all_20(int data[ ], size_t size)
{
   for (int i = 0 ; i< size; i++)
   {
            data[i] = 20;
   }
}
```

- An array parameter automatically treated as pointer to the first entry (– value or reference?)

- In the function prototype and implementation, size of the array is not specified inside bracket but by another parameter

@ Feng HU, 2016

Pointers or Array as const Parameters

- to make sure they will not be changed

```
Protoptyes:
bool is_20(const int* i_ptr);
double  average(const int data[ ], size_t size);
```

```
Calling program:
int *ages, *i_ptr;
double aver_age;
ages = new int [ 30 ];
…
aver_age = average(ages, 30);
i_ptr = &ages[12];  // i_ptr = (ages+12);
if (is_20(i_ptr)) cout <<“Sudent No. 13 is 20!”<<endl;
```

Reference Parameters that are Pointers

- if we want to change the pointer to a new location

```
void allocate_int_arrary(int* i_ptr, size_t size)
{
    i_ptr = new int[size];
}
```

X

```
Calling program:
int *ages;
int  jone = 20;  // assume &jone is 904 now
ages = &jone;
cout << “address that ages points to is ”<< ages<<endl;
allocate_int_array(ages, 30);
cout << “address that ages points to is ”<< ages<<endl;
```

Reference Parameters that are Pointers

- if we want to change the pointer to a new location

```
void allocate_int_arrary(int*& i_ptr, size_t size)
{
    i_ptr = new int[size];
}
```

V

```
Calling  program:
int *ages;
int  jone = 20; // assume &jone is 904 now
ages = &jone;
cout << “address that ages points to is ”<< ages<<endl;
allocate_int_array(ages, 30);
cout << “address that ages points to is ”<< ages<<endl;
```

Reference Parameters that are Pointers

- if we want to change the pointer to a new location

```
typedef int* integer_ptr;
void allocate_int_arrary(integer_ptr& i_ptr, size_t size)
{
    i_ptr = new int[size];
}
```

V

```
Calling  program:
int *ages;
int  jone = 20; // assume &jone is 904 now
ages = &jone;
cout << “address that ages points to is ”<< ages<<endl;
allocate_int_array(ages, 30);
cout << “address that ages points to is ”<< ages<<endl;
```

Reading and Programming Assignments

- Reading before the next lecture
  - Chapter 4. Sections 4.3-4.4

- Programming Assignment 2
  - Detailed guidelines online!
  - Due September 28 (Wednesday)