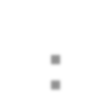
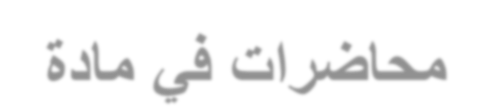
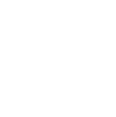
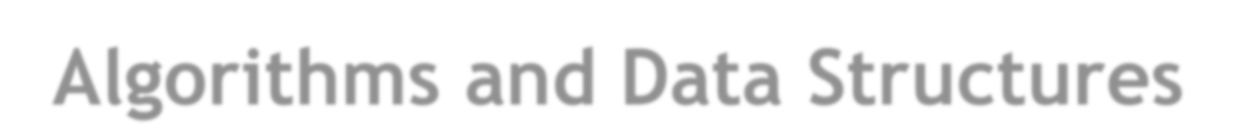
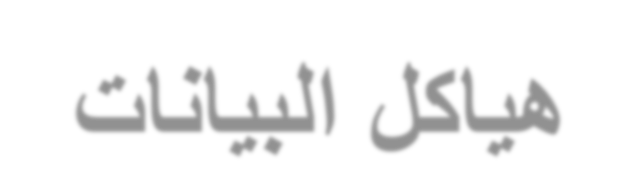
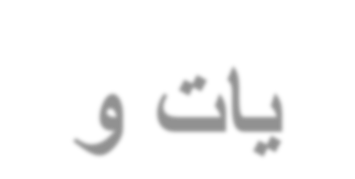
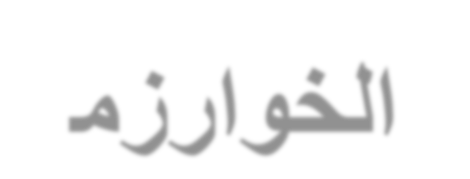
25



**Lecture # 2**

Pointers and Dynamic Objects

Outline

• Pointers

 Memory addresses

 Declaration

 Dereferencing a pointer

 Pointer to pointer

• Static vs. dynamic objects

 **new** and **delete**

Computer Memory

• Each variable is assigned a memory slot (the size depends on the

data type) and the variable’s data is stored there.

**int a = 100;**

Memory ad

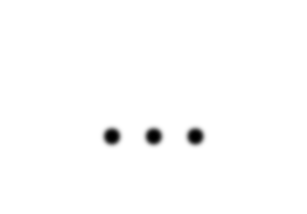
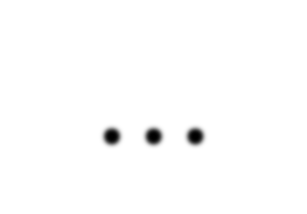
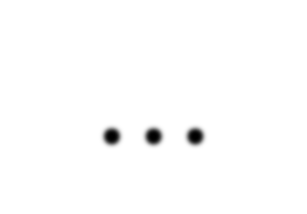
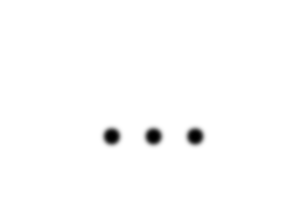
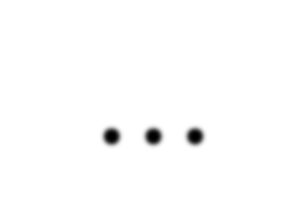
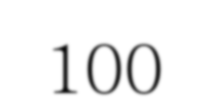
1020

1024 1032

… … 100

ress

a



… … …

Variable a’s value, i.e. 100, is stored at memory location 1024

Pointers

• A pointer is a variable used to store the address of a memory cell.

• We can use the pointer to reference this memory cell.

Memory add

1020

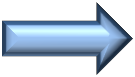
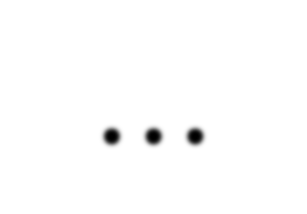
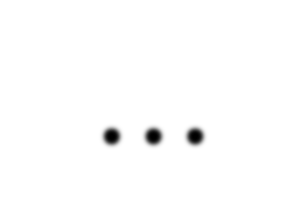
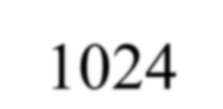
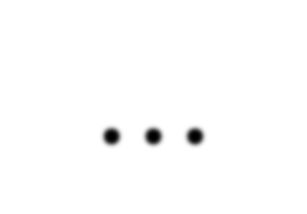
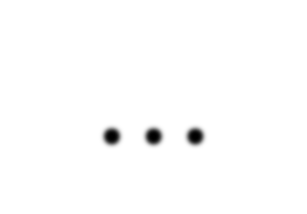
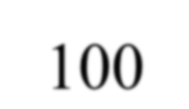
1024 1032

… … 100

ress

… 1024 …

integer



variable pointer

Pointer Variable

• Declaration of Pointer variables:

***type*\* pointer\_name;**

**//or**

***type* \*pointer\_name;**

 where *type* is the type of data pointed to (e.g. **int**, **char**,

**double**, etc).

 Examples:

**int \*n; RationalNumber \*r;**

**int \*\*p; // pointer to pointer**

Address Operator ‘&’

• *The* "*address of* " *operator* ‘&’ gives the memory address of the variable.

 **Usage: &variable\_name**

Memory ad

…

dress

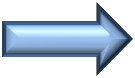
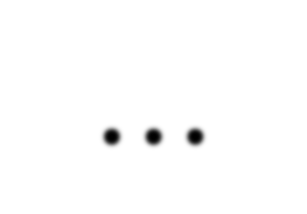
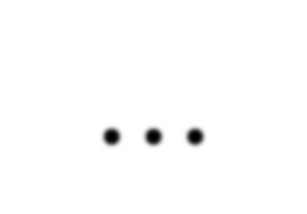
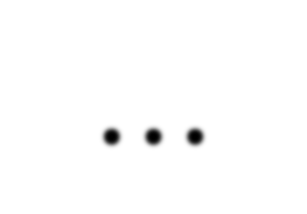
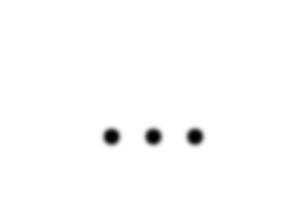
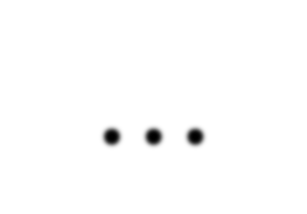
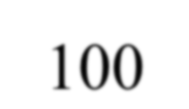
1020

…

1024

100

a



… … …

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **int a =** | **100;** | **//get** | **the** | **value** |
| **cout <<** | **a;** | **//pri** | **nts 1** | **00** |

**//get the memory address**

**cout << &a; //prints 1024**

Address Operator ‘&’

Memory a

1020

1024 1032

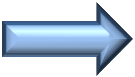
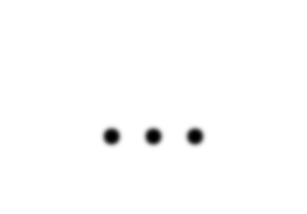
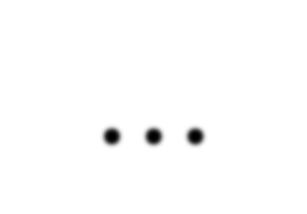
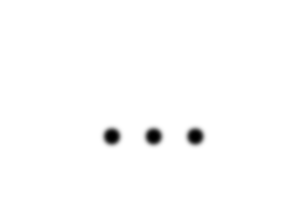
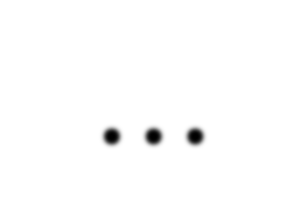
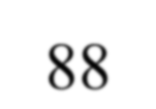
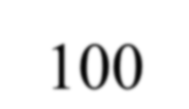
… 88

ddress

a

100

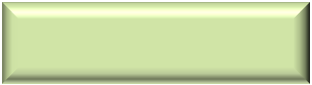
b



… … …

**#include <iostream> using namespace std; void main(){**

**int a = 88, b = 100;**



The Output:

The address of a is: 1020

The address of b is: 1024

**cout << "The address of a is: " << &a << endl;**

**cout << "The address of b is: " << &b << endl;**

**}**

Address Operator ‘&’

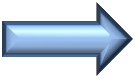
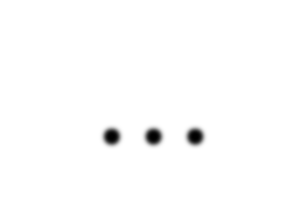
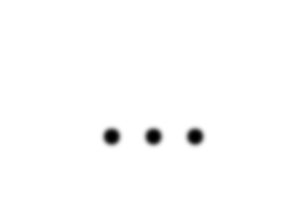
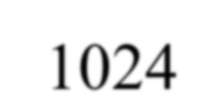
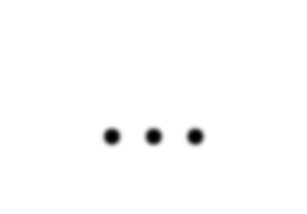
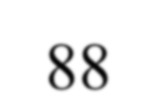
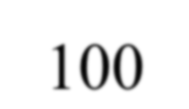
Memory address

1020

1024 1032

… 88

100



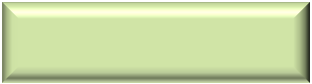
… 1024 …

a p

The Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **int** | **a =** | **100;** |  | |
| **int** | **\*p;** |  |
| **p =** | **&a;** |  |
| **cout <<** | | **a <<** | **" " <<** | **&a << endl;** |
| **cout <<** | | **p <<** | **" " <<** | **&p << endl;** |

100 1024



1024 1032

• The value of pointer *p* is the address of variable *a*.

• A pointer is also a variable, so it has its own memory address.

**Pointers and Dynamic Objects** 33

Dereferencing Operator ‘**\***’

• We can access to the value stored in the variable pointed to by

using the dereferencing operator ‘\*’.

Memory a

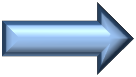
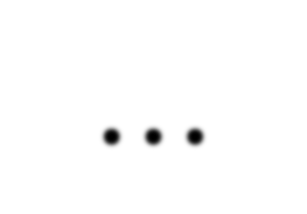
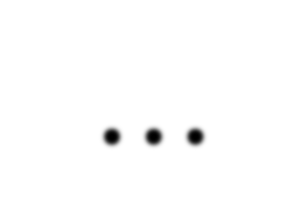
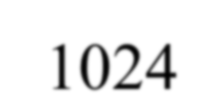
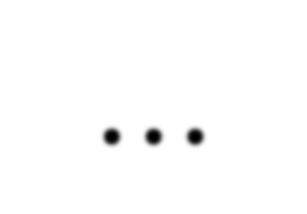
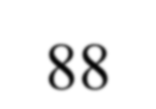
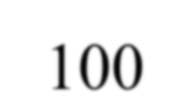
1020

1024 1032

… 88

ddress

100



… 1024 …

a p

**int a = 100;**

**int \*p = &a;**

**cout << a << endl;**

**cout << &a << endl;**

**cout << p << " " << \*p << endl;**

**cout << &p << endl;**

The Output:

100

1024



1024 100

1032

Don’t get confused

 Declaring a pointer means only that it is a pointer:

**int \*p;**

 Don’t be confused with the dereferencing operator, which is also

written with an asterisk ‘\*’.

 They are simply two different tasks represented with the same sign.

**int a = 100, b = 88, c = 8;**

**int \*p1 = &a, \*p2, \*p3 = &c;**

**p2 = &b; // p2 points to b p2 = p1; // p2 points to a b = \*p3; //assign c to b**

**\*p2 = \*p3; //assign c to a**

**cout << a << b << c;**

The Output:

8 8 8



A Pointer Example

**#include <iostream> using namespace std; int main (){**

**int value1 = 5, value2 = 15;**

**int \*p1, \*p2;**

What is the output?

**p1 = &value1; *// p1 = address of value1***

**p2 = &value2; *// p2 = address of value2***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **\*p1 = 10;** | ***//*** | ***value pointed*** | ***to*** | ***by p1=10*** |
| **\*p2 = \*p1;** | ***//*** | ***value pointed*** | ***to*** | ***by p2= value*** |
|  | ***//*** | ***pointed to by*** | ***p1*** |  |

**p1 = p2; *// p1 = p2 (pointer value copied)***



**\*p1 = 20; *// value pointed to by p1 = 20***

**cout << "value1= " << value1 << endl**

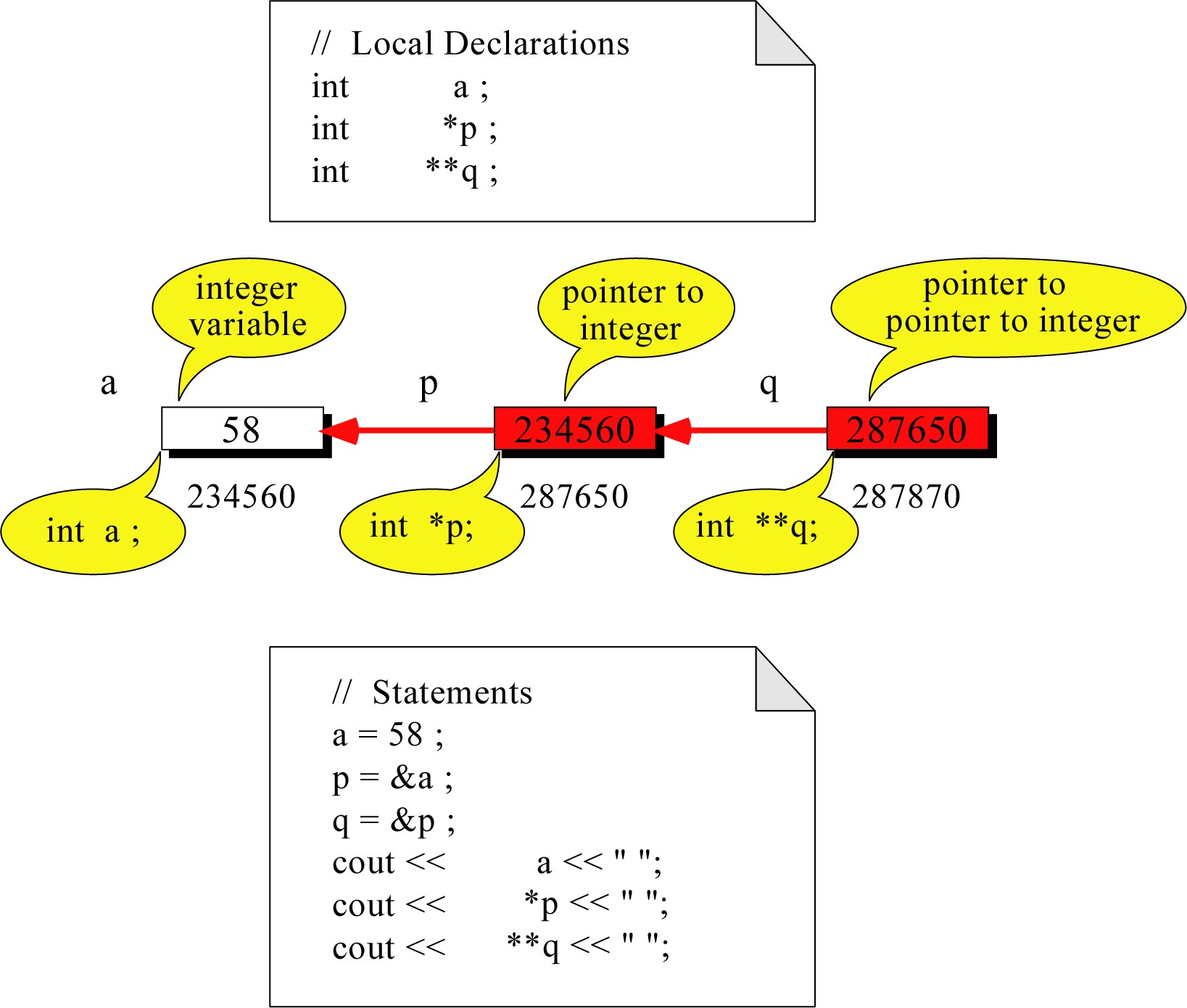
**<< "value2= " << value2 << endl**

**<< "\*p1= " << \*p1 << " \*p2= " << \*p2;**

|  |  |
| --- | --- |
|  | **return 0;** |
| **}** |  |

Pointer to Pointer

The Output:



58 58 58

Traditional Pointer Usage

**void Swap(char \*Ptr1, char \*Ptr2){**

**char temp = \*Ptr1;**

**\*Ptr1 = \*Ptr2;**

**\*Ptr2 = temp;**

**}**

**int main()**

**{**

**char a = 'y';**

**char b = 'n';**

**cout << a << b << endl; Swap(&a, &b);**

**cout << a << b << endl;**

**return 0;**

**}**

The Output:

**yn ny**



Pass by Reference

**void Swap(char& m, char& n) {**

**char temp = m;**

**m = n;**

**n = temp;**

**}**

**int main()**

**{**

**char a = 'y';**

**char b = 'n';**

**cout << a << b << endl;**

**Swap(a, b);**

**cout << a << b << endl;**

**return 0;**

**}**

The Output:

**yn ny**



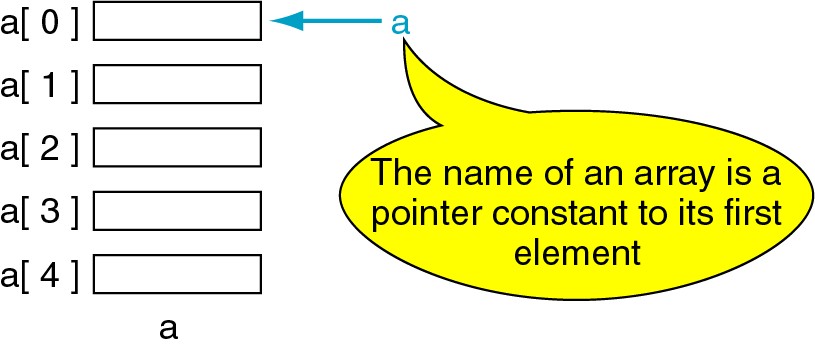
Pointers and Arrays

• The name of an array points only to the first element not the whole array.

1000

1004

1008



1012

1016

Array Name is a pointer constant

**#include <iostream>**

**using namespace std;**

**void main ()**

**{**

**int a[5];**

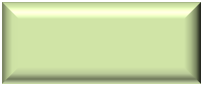
**cout << "Address of a[0]: " << &a[0] << endl;**

**<< "Name as pointer: " << a << endl;**

**}**

Result:

**Address of a[0]: 0x0065FDE4**



**Name as pointer: 0x0065FDE4**

Dereferencing An Array Name

This element is called a[0] or \*a

a[0] a[1] a[2] a[3] a[4]

|  |
| --- |
| 2 |
| 4 |
| 6 |
| 8 |
| 22 |

**a**

**#include <iostream> using namespace std; void main(){**

**int a[5] = {2,4,6,8,22};**

**cout << \*a << " "**

**<< a[0];**

**} //main**

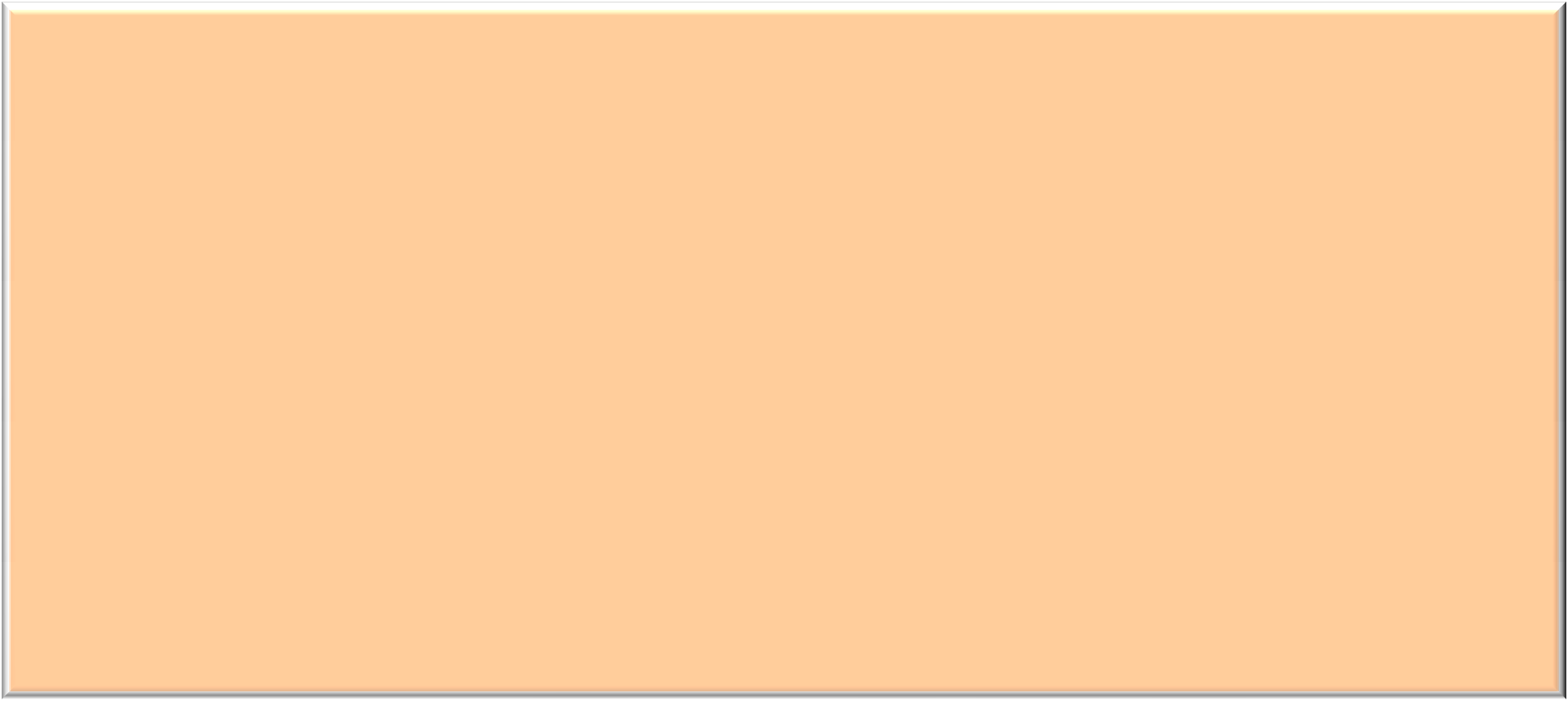
Array Names as Pointers

 To access an array, any pointer to the first element can be used instead of the name of the array.



|  |  |  |
| --- | --- | --- |
| **p**  **a #include <iostream>**  **using namespace std;**  **void main(){** The Output:  a[0] 2 **int a[5] = {2,4,6,8,22};** 2 2  a[1] 4 **int \*p = a;**  a[2] 6 **cout << a[0] << " "**  **<< \*p;**  a[3] 8 **}**  a[4] 22  **a** | | |
|  | We could replace **\*p** by **\*a** |  |

Dereferencing Array Pointers



|  |
| --- |
| 2 |
| 4 |
| 6 |
| 8 |
| 22 |

|  |  |  |
| --- | --- | --- |
| **a** |  | |
| **a** | **+** | **1** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a[0]** | **or** | **\*(a** | **+** | **0)** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a[1]** | **or** | **\*(a** | **+** | **1)** | **a + 2** | | |
| **a[2]** | **or** | **\*(a** | **+** | **2)** |
| **a[3]** | **or** | **\*(a** | **+** | **3)** | **a** | **+** | **3** |
| **a[4]** | **or** | **\*(a** | **+** | **4)** | **a** | **+** | **4** |

**\*(a+n)** is identical to **a[n]**

44

Dynamic Objects

Memory Management

• Static Memory Allocation:

 Memory is allocated at compilation time.

• Dynamic Memory Allocation:

 Memory is allocated at running time.

Static vs. Dynamic Objects

• Static object:

 Memory is acquired

automatically.

 Memory is returned automatically when object goes out of scope.

• Dynamic object:

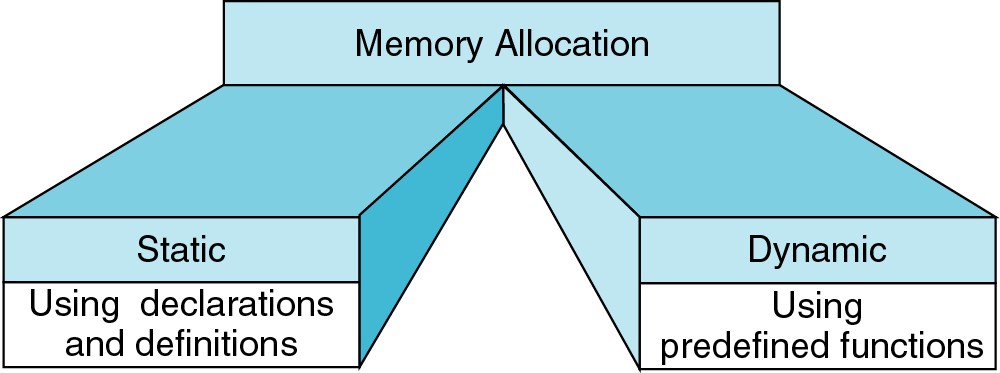
 Memory is acquired by program with an allocation request:

 **new** operation.

 Object memory is returned by a deallocation request:

 **delete** operation.

Memory Allocation



**{**

**int a[200];**

**…**

**}**

**int\* ptr;**

**ptr = new int[200];**

**…**

**delete [] ptr;**

Object (variable) creation: **new**

**Syntax:**

**ptr = new SomeType;**

 where **ptr** is a pointer of type **SomeType**.

Example:

**int\* p = new int;**

Uninitialized int variable

p

Object (variable) destruction: **delete**

**Syntax:**

**delete p;**

 Storage pointed to by **p** is returned to free store and **p** is now undefined.

**Example:**

**int\* p = new int;**

**\*p = 10;**

**delete p;**

**p** 10

Array of **new**:

**dynamic arrays**

• Syntax:

**P = new SomeType[Size];**

 where:

 P is a pointer of type SomeType.

 **Size** is the number of objects to be constructed.

• Because of the flexible pointer syntax, **P** can be considered to be an array.

Example

• Dynamic Memory Allocation:

 Request for “unnamed” memory from the Operating System.

 **int \*p, n=10;** p

|  |  |  |  |
| --- | --- | --- | --- |
| **p** | **=** | **new** | **int;** |
|  **p**   **p** | **=**  **=** | **new**  **new** | **int[100];**  **int[n];** |

new

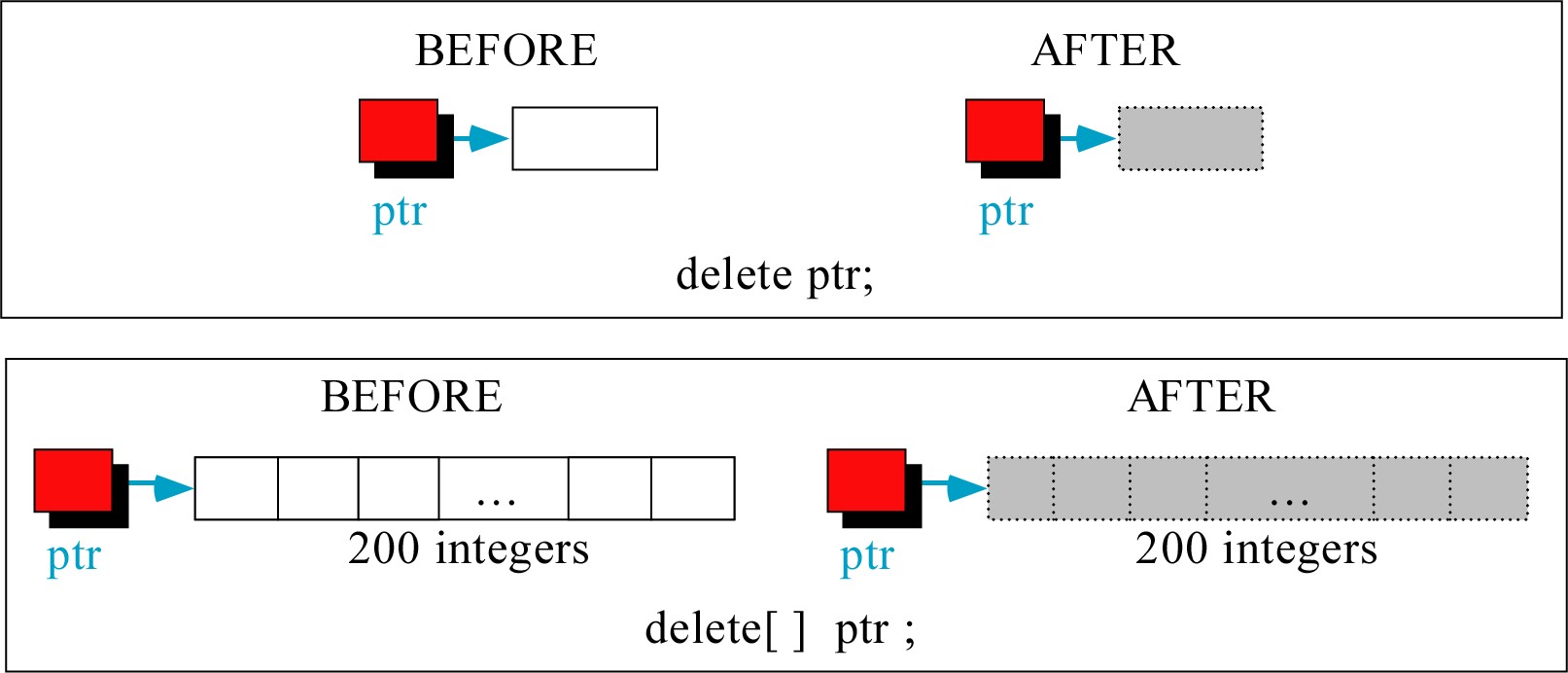
new

p

new

p

Freeing (or deleting) Memory



Memory Allocation Example

 Want an array of unknown size

**#include <iostream> using namespace std; void main()**

**{**

**int n;**

**cout << "How many students? ";**

**cin >> n;**

**int \*grades = new int[n];**

**for(int i=0; i < n; i++){**

**cout << "Input Grade for Student "**

**<< (i+1) << " ? : ";**

**cin >> grades[i]; }**

**for(int i=0; i < n; i++){**

**cout << "Grade[" << (i+1) << "]="**

**<< grades[i] << endl; }**

**delete [] grades;**

**}**