FUNDAMENTALS OF STRUCTURED PROGRAMMING

Lecture 10

Pointers II and Dynamic Arrays

Course Coordinator: Prof. Zaki Taha Fayed

Presented by: Dr. Sally Saad

SallySaad@gmail.com

DropBox folder link

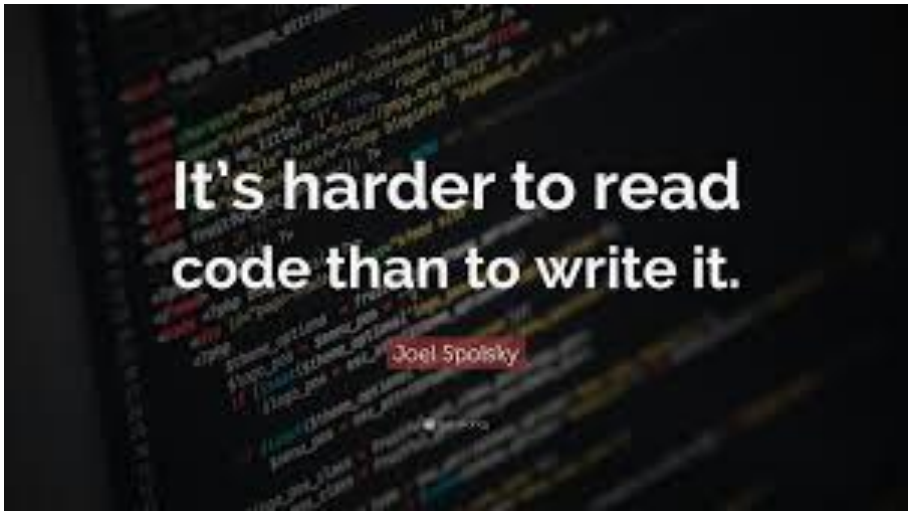
https://www.dropbox.com/sh/85vnrgkfqgrzhwn/AABdwKLJZqZs26a7u-y0AFwia?dl=0

Credits to Dr.Salma Hamdy for content preparation

Quote of the Day!

"FIRST, SOLVE THE PROBLEM.
THEN, WRITE THE CODE."

- JOHN JOHNSON



Pointers II
And Dynamic Arrays

Contents

1. Pointers (Revisited)

- A pointer is the memory address of a variable.
- A pointer variable holds a pointer value. A pointer value is the address of a variable in memory.

Declaration

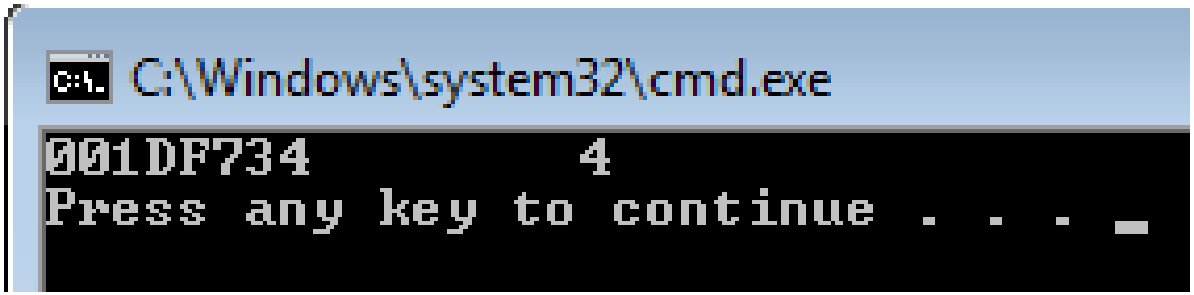
```
int count = 7;
int *p1, *p2, v1, v2;   double *p;
```

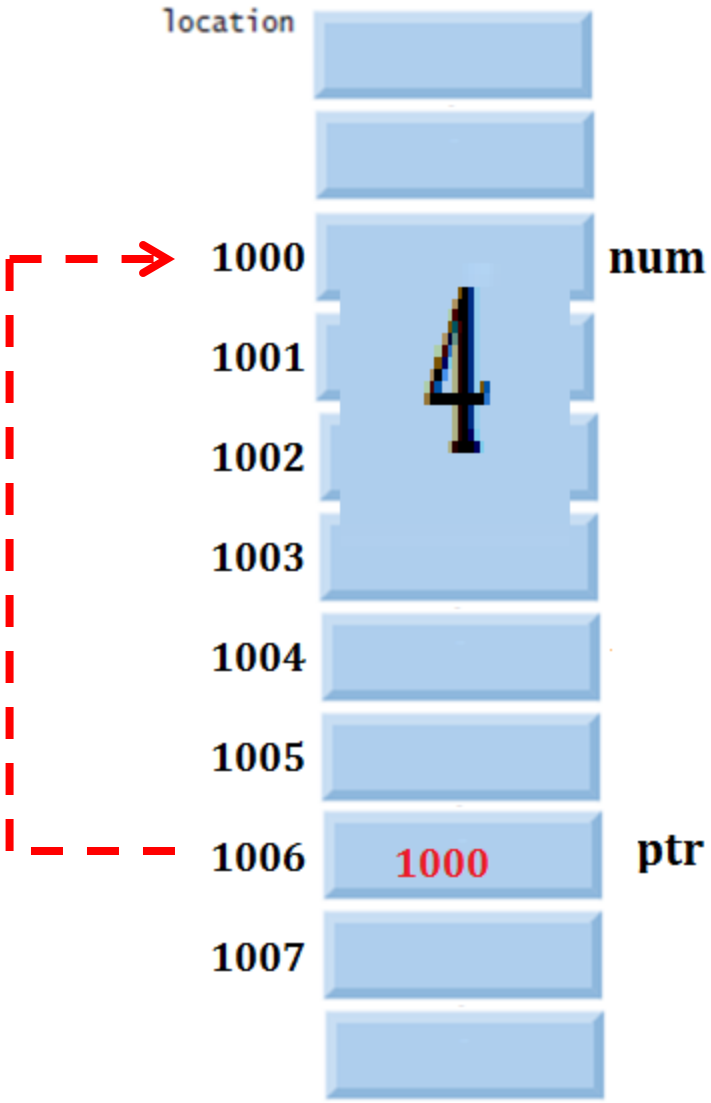
1. Pointers (Revisited) – (cont.)

Initialization (1)

Address operator

```
int num = 4;
int *ptr = &num;
cout<<ptr<<"\t"<<*ptr<<endl;
```

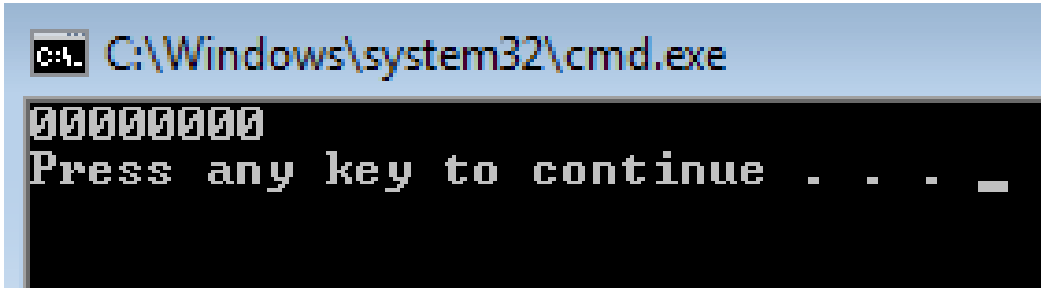




1. Pointers (Revisited) – (cont.)

Initialization (2)

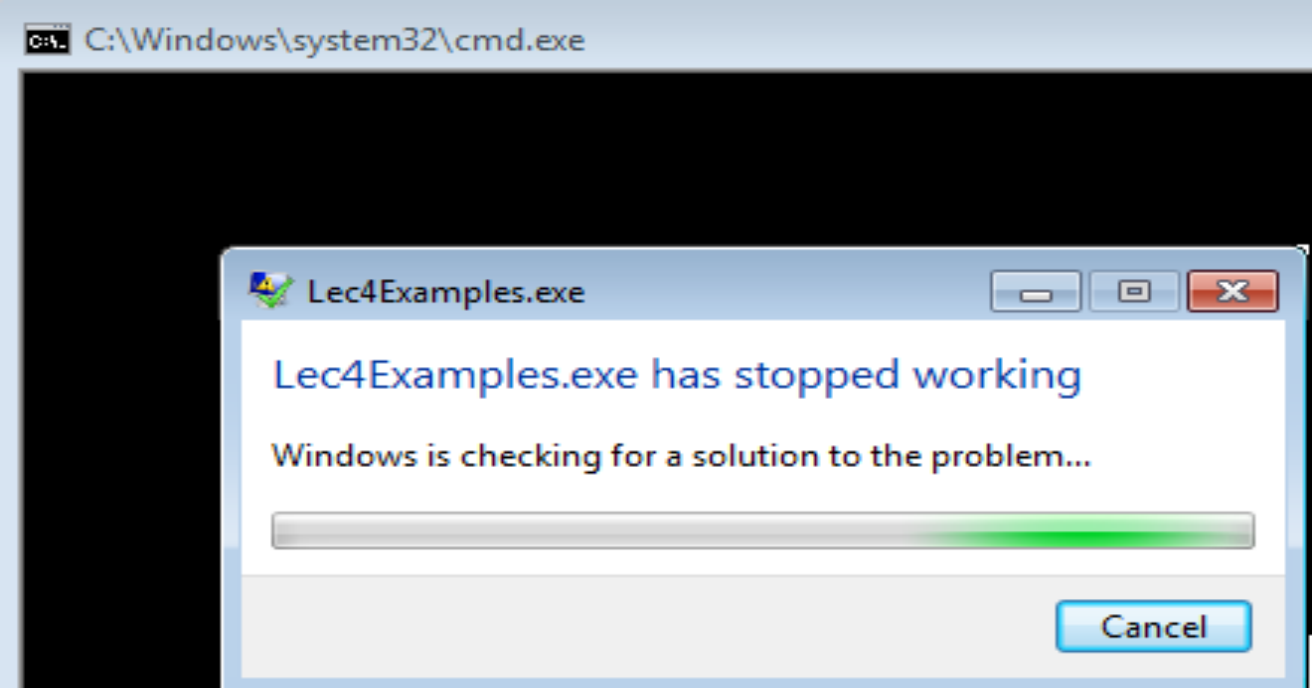
```
int num = 4;
int *ptr = NULL;
cout<<ptr<<endl;
```



1. Pointers (Revisited) – (cont.)

Initialization (2)

```
int num = 4;
int *ptr = NULL;
cout<<ptr<<"\t"<<*ptr<<endl;
```

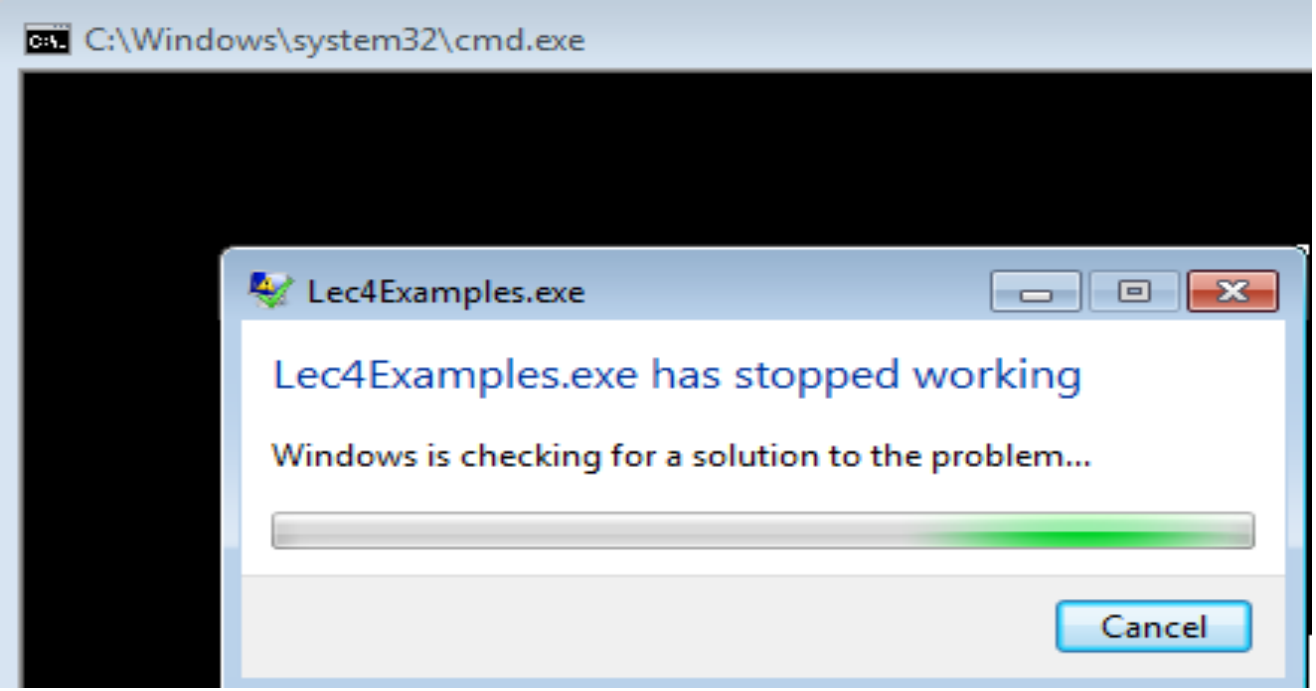


You cannot dereference a NULL pointer.

1. Pointers (Revisited) – (cont.)

Initialization (3)

```
int num = 4;
int *ptr = 0;
cout<<ptr<<"\t"<<*ptr<<endl;
```



You cannot dereference a NULL pointer.

1. Pointers (Revisited) – (cont.)

Dereferencing Operator (indirection)

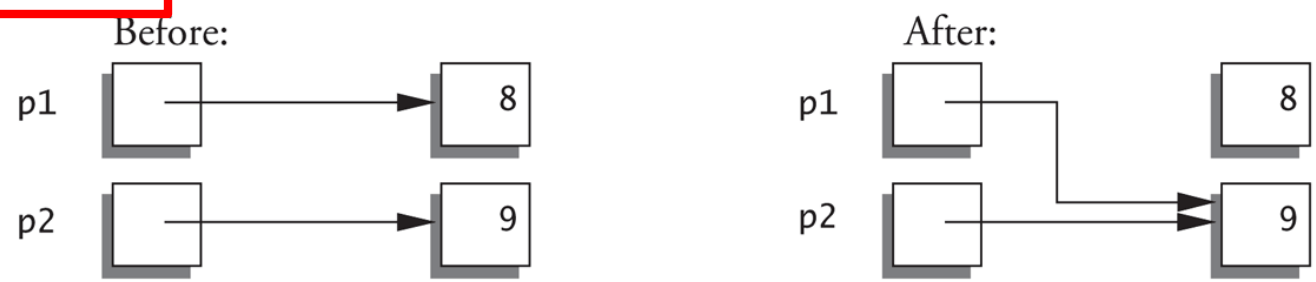
```
int *p1, *p2, v1, v2;
p1 = &v1;
v1 = 7;
cout<<v1;
cout<<*p1;
```

1. Pointers (Revisited) – (cont.)

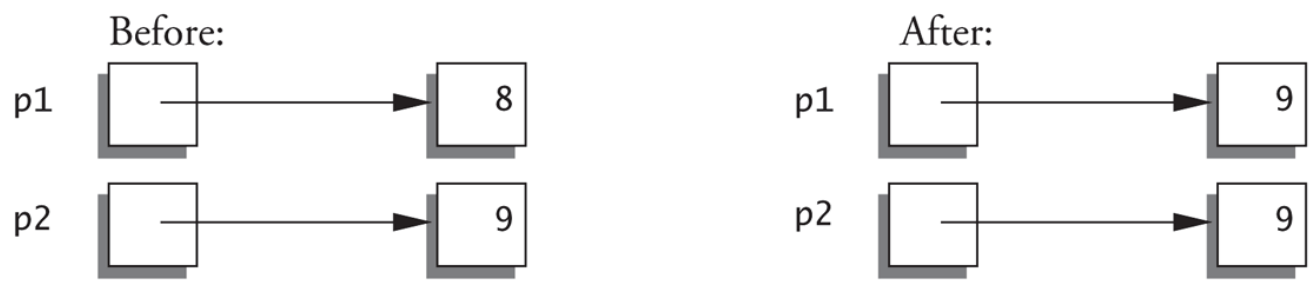
Pointers in Assignments

Display 10.1 Uses of the Assignment Operator with Pointer Variables

```
p1 = p2;
```




```
*p1 = *p2;
```



1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- Pointer is **NOT** an integer and cannot be used as a number.

```
int add = ptr+1;
```

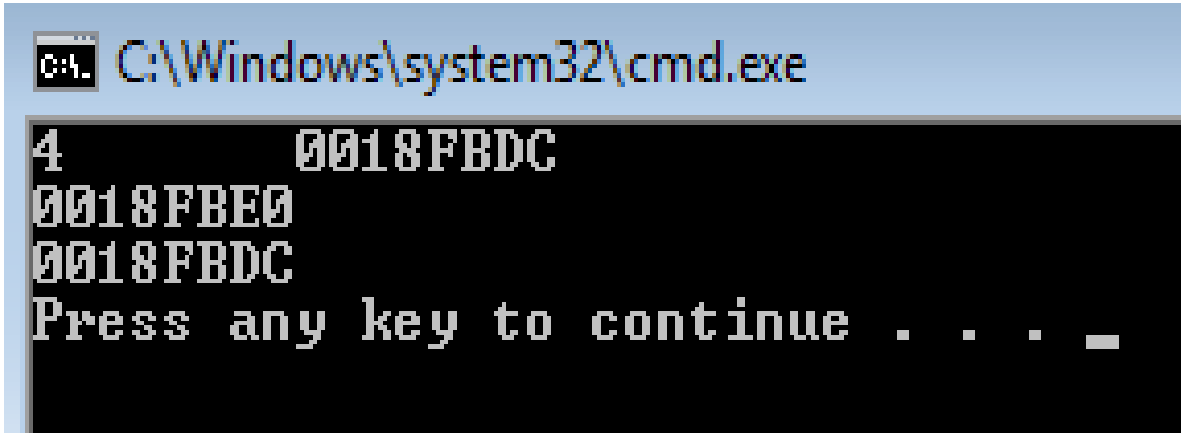
1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
cout<<num<<"\t"<<ptr<<endl;

ptr = ptr + 1; cout<<ptr<<endl;
ptr = ptr - 1; cout<<ptr<<endl;
```

Here, the address will be altered by four bytes.

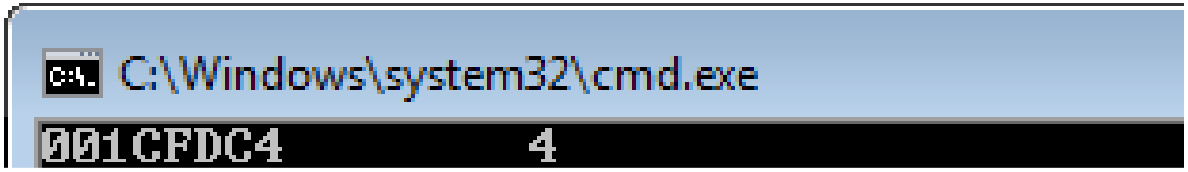


1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
cout<<ptr<<"\t"<<*ptr<<endl;
```

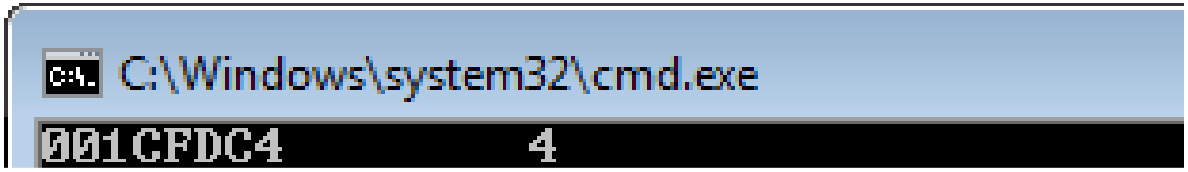


1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
cout<<ptr<<"\t"<<*ptr<<endl;

ptr = ptr + 1;
cout<<ptr<<"\t"<<*ptr<<endl;
```



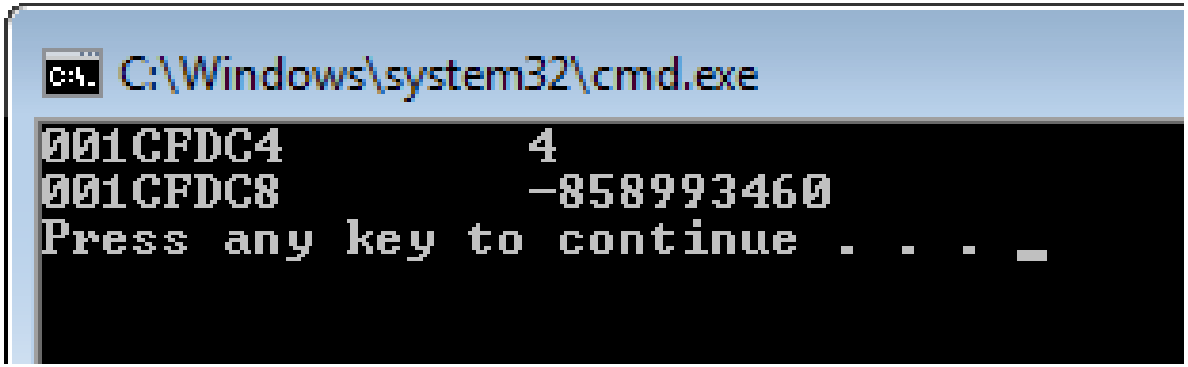
1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
cout<<ptr<<"\t"<<*ptr<<endl;

ptr = ptr + 1;
cout<<ptr<<"\t"<<*ptr<<endl;
```

Here, the address will be advanced by four bytes and you can VIEW the RANDOM value in that new address.

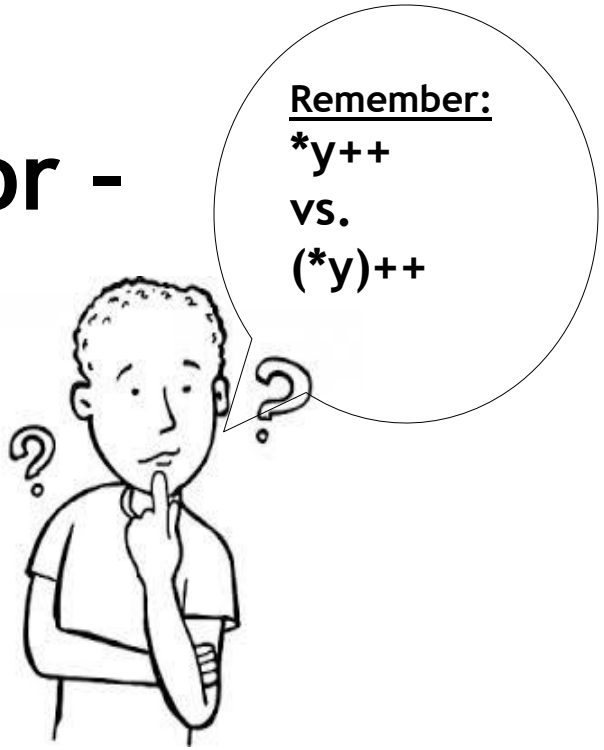


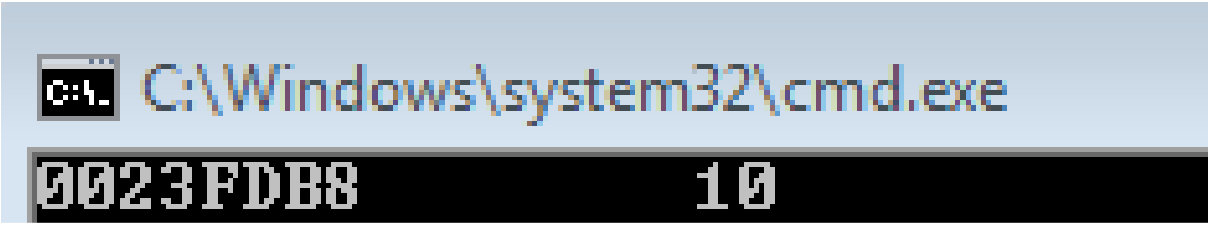
1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
*ptr +=6;
cout<<ptr<<"\t"<<*ptr<<endl;
```





1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
*ptr +=6;
cout<<ptr<<"\t"<<*ptr<<endl;

ptr = ptr + 1;
cout<<ptr<<"\t"<<*ptr<<endl;
```

C:\Windows\system32\cmd.exe

```
0023FDB8        10
```

1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
*ptr +=6;
cout<<ptr<<"\t"<<*ptr<<endl;

ptr = ptr + 1;
cout<<ptr<<"\t"<<*ptr<<endl;
```

C:\Windows\system32\cmd.exe

```
0023FDB8        10
0023FDBC        -858993460
```

1. Pointers (Revisited) – (cont.)

Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
*ptr +=6;
cout<<ptr<<"\t"<<*ptr<<endl;

ptr = ptr + 1;
cout<<ptr<<"\t"<<*ptr<<endl;
*ptr += 6;
```

C:\Windows\system32\cmd.exe

```
0023FDB8        10
0023FDBC        -858993460
```

1. Pointers (Revisited) – (cont.)

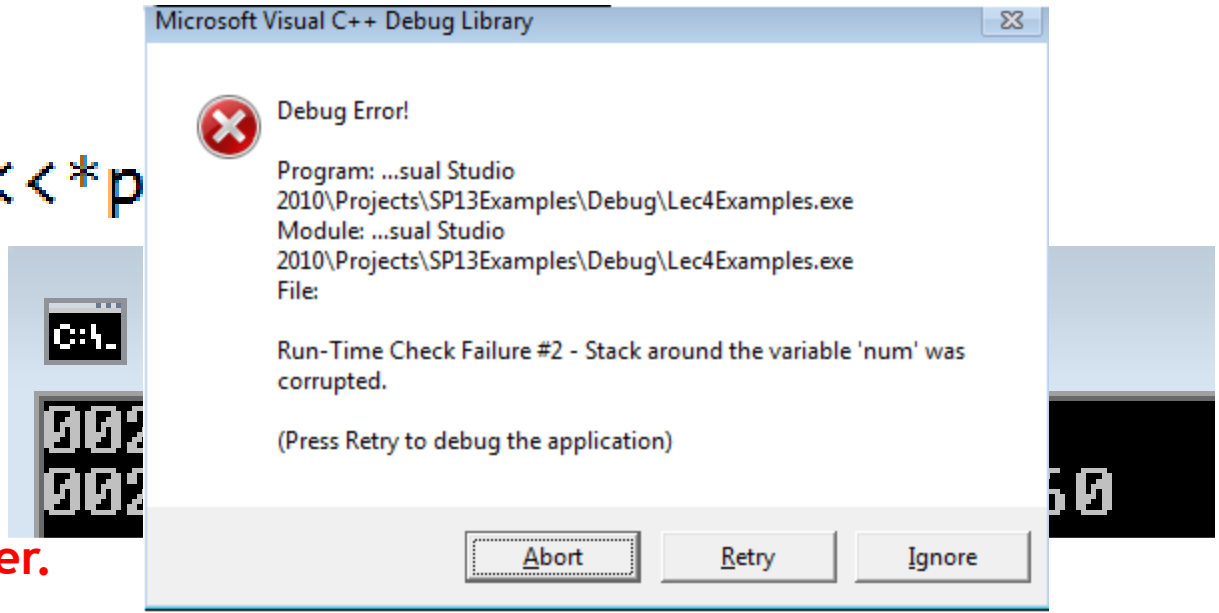
Pointer Arithmetic

- But you can use it with + or -

```
int num = 4, *ptr = &num;
*ptr +=6;
cout<<ptr<<"\t"<<*ptr<<endl;

ptr = ptr + 1;
cout<<ptr<<"\t"<<*p
*ptr += 6;
```

You cannot CHANGE values at locations that are not yours!

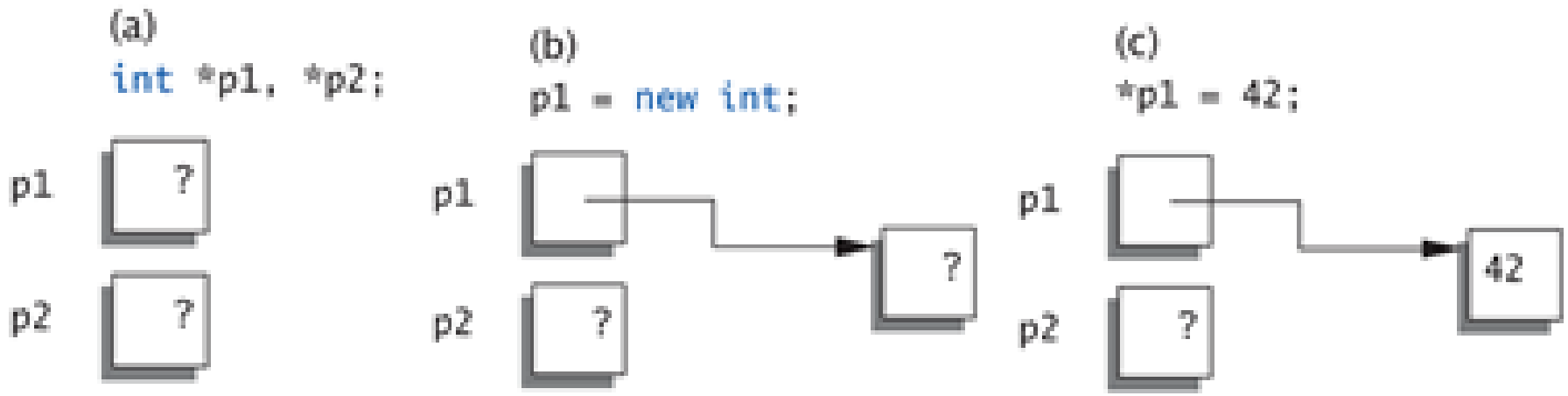
Could depend on the compiler.



2. Dynamic Memory (Revisited)

Dynamic Variables Allocation

```
p1 = new int;
```



2. Dynamic Memory – (cont.)

Dynamic Variables Initialization

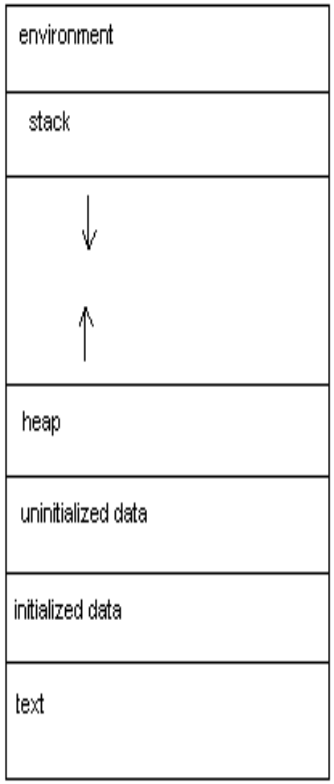
```
int *iPtr = new int(17);     //Initializes *iPtr to 17

double *dPtr;
dPtr = new double(98.6); // Initializes *dPtr to 98.6.


int num(5);  // exactly as int num = 5;
int* ptr = new int(num);
```

Automatic or local variable

2. Dynamic Memory – (cont.)

Insufficient Memory Test

```
int* ptr = new int;
if(ptr==NULL)
{
    cout<<"Failed to allocate memory.\n";
    exit(1);
} // end if
else
    cout<<"successful new allocation.\n";
```



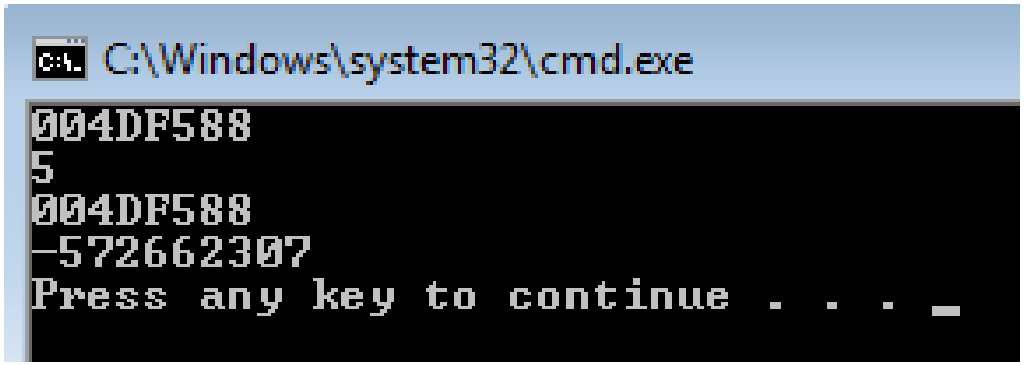
Virtual memory organization

2. Dynamic Memory – (cont.)

Dynamic Variables De-allocation

- Destroys dynamic memory but `ptr` still points there! Called **dangling pointer**.

```
int* ptr = new int(5);
// processing
cout<<ptr<<endl;
cout<<*ptr<<endl;

delete ptr;

cout<<ptr<<endl;
cout<<*ptr<<endl;
```



2. Dynamic Memory – (cont.)

Dynamic Variables De-allocation

- Avoid dangling pointers by assigning pointer to NULL after delete.

```
int* ptr = new int(5);
// processing
cout<<ptr<<endl;
cout<<*ptr<<endl;

delete ptr;
ptr = NULL;

cout<<ptr<<endl;
cout<<*ptr<<endl;
```

You cannot dereference a null pointer

2. Dynamic Memory – (cont.)

- A pointer variable can be assigned to any variable type.
- Pointer to integer, to float, to double, to struct, to pointer, to array.
- Example: dynamic allocation of a struct variable.

3. Pointers and Arrays

- Our previous Array variables
  - Really pointer variables!
- Recall: arrays stored in memory addresses, sequentially
  - Array variable "refers to" first indexed variable
  - So array variable is a kind of pointer variable!
- Example:

```
int arr[10];
int *p;
```

  - `arr` and `p` are both pointer variables!

3. Pointers and Arrays – (cont.)

- So can they be assigned to each other?

```
int num = 4;
int arr[5] = {0};
int *ptr;

ptr = arr;
cout<<arr<<"\t"<<ptr<<endl;
```

```
arr = ptr;
cout<<arr<<"\t"<<ptr<<endl;
```

3. Pointers and Arrays – (cont.)

- So can they be assigned to each other?

```
int num = 4;
int arr[5] = {0};
int *ptr;

ptr = arr;
cout<<arr<<"\t"<<ptr<<endl;
```

```
arr = ptr;
cout<<arr<<"\t"<<ptr<<endl;
```

C:\Windows\system32\cmd.exe

```
0038FB50        0038FB50
Press any key to continue . . .
```

3. Pointers and Arrays – (cont.)

- So can they be assigned to each other?

```
int num = 4;
int arr[5] = {0};
int *ptr;

ptr = arr;
cout<<arr<<"\t"<<ptr<<endl;
```

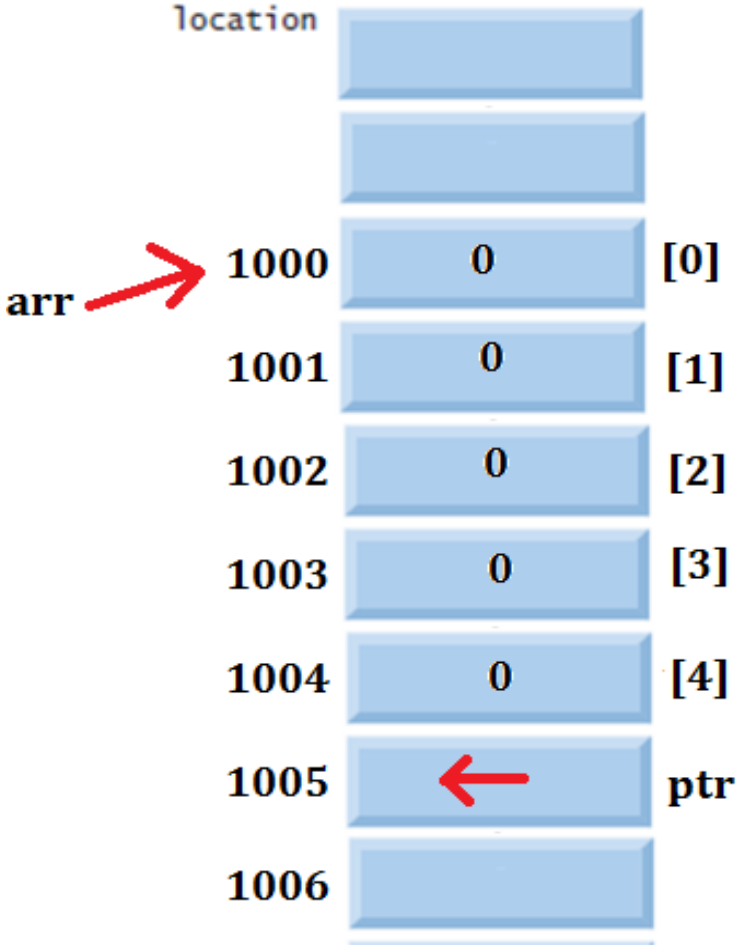
```
C:\Windows\system32\cmd.exe
0038FB50        0038FB50
Press any key to continue . . .
```

```
arr = ptr;
cout<<arr<<"\t"<<ptr<<endl;
```

ILLEGAL !
Note that an array name is exactly as a const int * type.

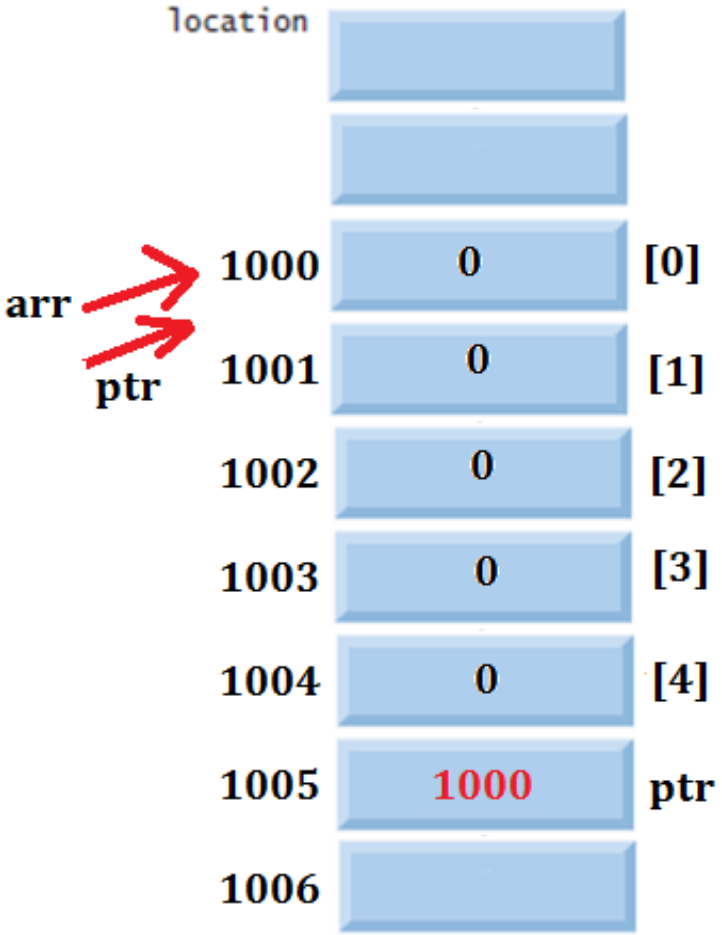
3. Pointers and Arrays – (cont.)

```
int arr[5] = {0};
int *ptr;
```



3. Pointers and Arrays – (cont.)

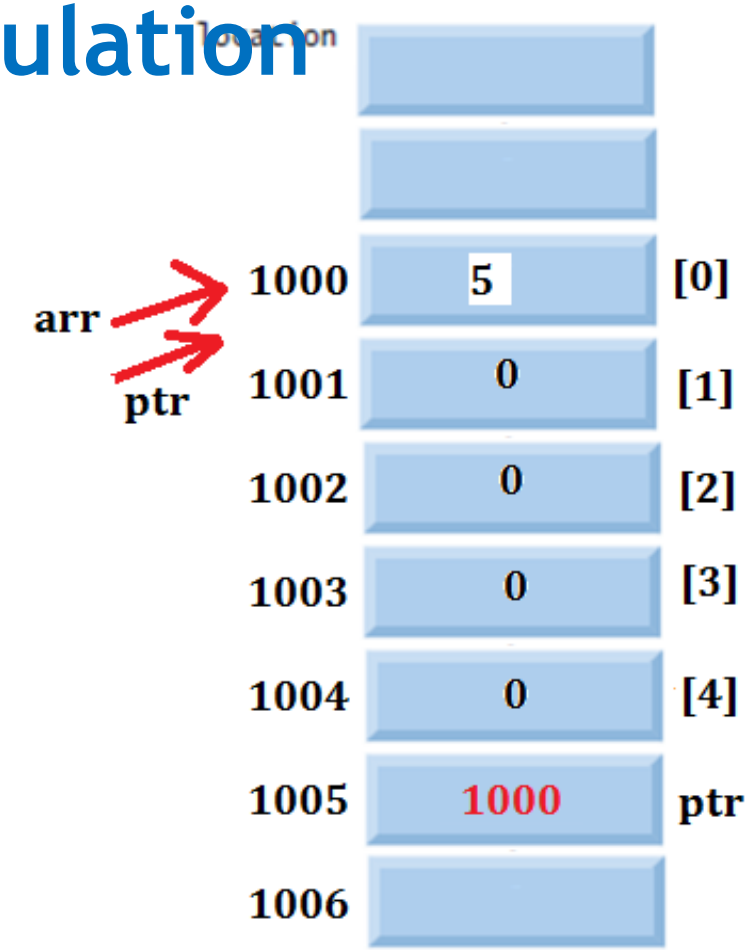
```
int arr[5] = {0};
int *ptr;

ptr = arr;
```

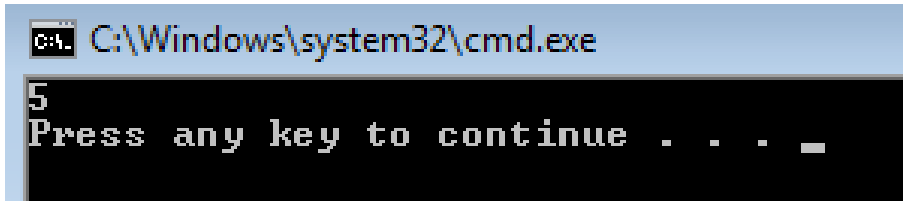


3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

```
int arr[5] = {0};
int *ptr;

ptr = arr;

*ptr = 5;
cout<<arr[0]<<endl;
```





3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- **Using address arithmetic.**

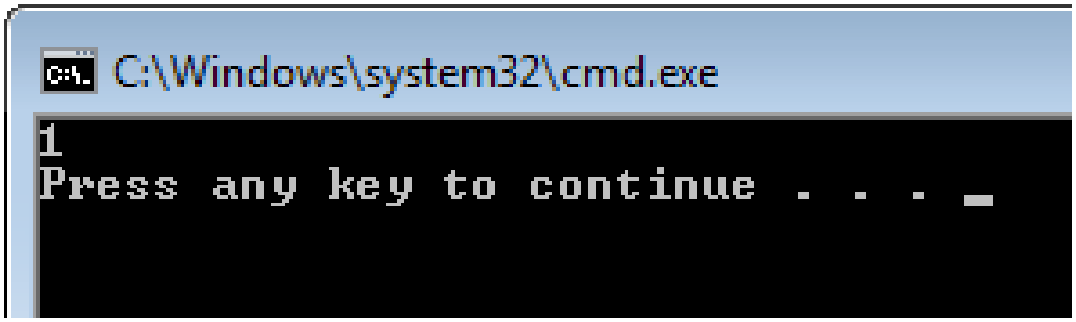
```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
ptr++;
cout<<*ptr<<endl;
```

3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- **Using address arithmetic.**

```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
ptr++;
cout<<*ptr<<endl;
```



This is equivalent to

```
ptr = &arr[1];
```

3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- To display all elements

```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
for(int i=0; i<5; i++)
    cout<<   ?   <<endl;
```

C:\Windows\system32\cmd.exe

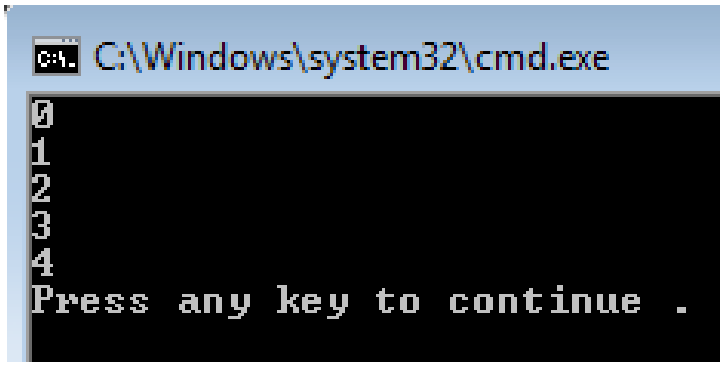
```
0
1
2
3
4
Press any key to continue .
```

3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- To display all elements

```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
for(int i=0; i<5; i++)
    cout<<*(ptr+i)<<endl;
```



3. Pointers and Arrays – (cont.)

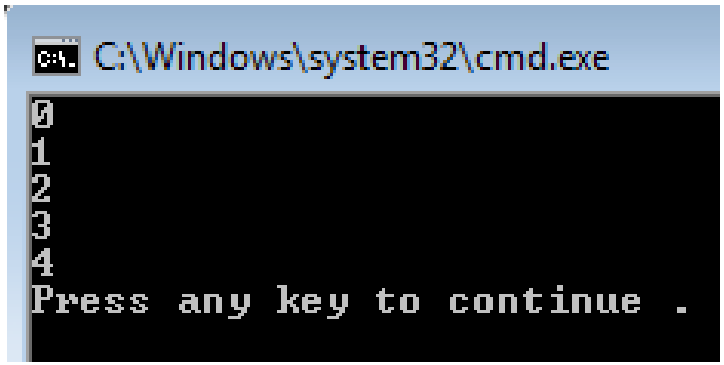
Alternative Arrays Manipulation

- To display all elements


```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
for(int i=0; i<5; i++)
    cout<<*(ptr+i)<<endl;
```

This is equivalent to

```
cout<<*(arr+i)<<endl;
```

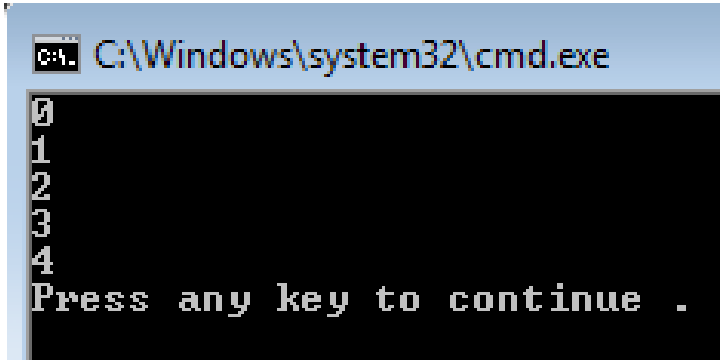


3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- To display all elements

```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
for(int i=0; i<5; i++)
    cout<<   ?   <<endl;
```

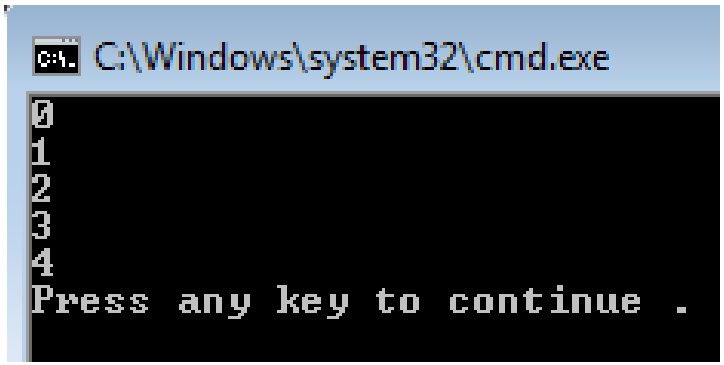


3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- To display all elements

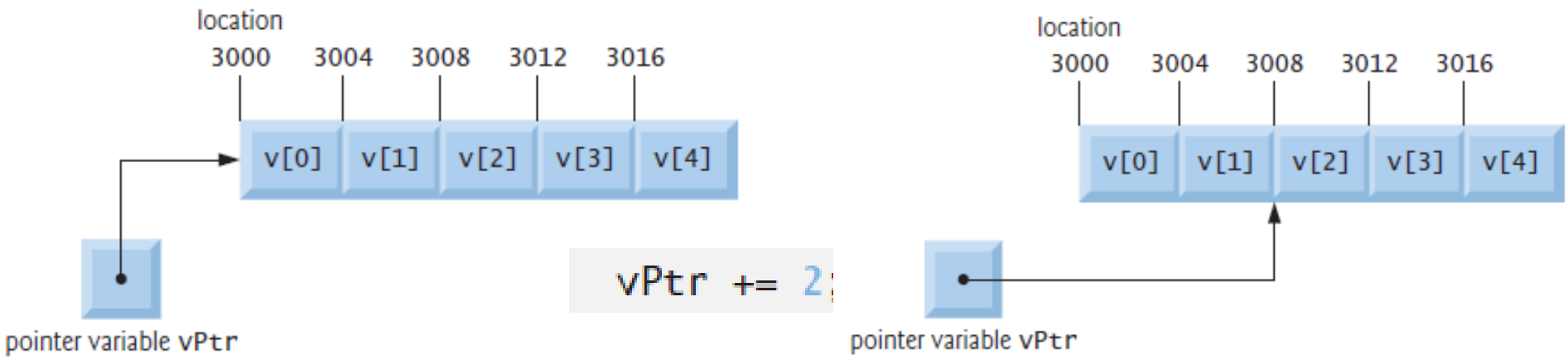
```
int arr[5] = {0, 1, 2, 3, 4};
int *ptr;

ptr = arr;
for(int i=0; i<5; i++)
    cout<<ptr[i]<<endl;
```



3. Pointers and Arrays – (cont.)

Alternative Arrays Manipulation

- Only addition/subtraction on pointers
  - No multiplication, division
- Can use ++ and -- on pointers



4. Dynamic Arrays

Standard array

- Must specify size first - Fixed size
- May not know actual size until program runs!
- Must "estimate" maximum size needed
  - "Wastes" memory

Dynamic array

- Size not specified at programming time
- Determined while program running
- Can grow and shrink as needed

4. Dynamic Arrays – (cont.)

Allocating Dynamic Array Variable

- Use `new` operator
  - Dynamically allocates memory with pointer variable at run time.

```
double *d = new double[10];
```

  - Creates dynamically allocated array variable *d*, with ten elements, base type double.

4. Dynamic Arrays – (cont.)

Allocating Dynamic Array Variable

- The advantage here is that we can assign a <u>variable as the array size.</u> Something we couldn't do in automatic array variables.

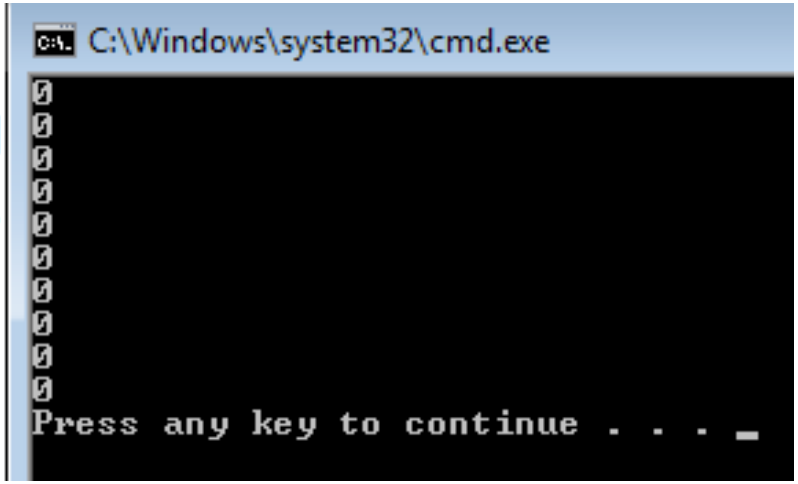
```
int size  = 4; // or cin>>size from user
int arr[size];
int *dArr = new int[size];
```

4. Dynamic Arrays – (cont.)

Initializing Dynamic Array Variable

- Recall array variable zero initialization

```
int arr[5] = {}; // same as = {0}
for(int i=0; i<5; i++)
    cout<<arr[i]<<endl;
```



4. Dynamic Arrays – (cont.)

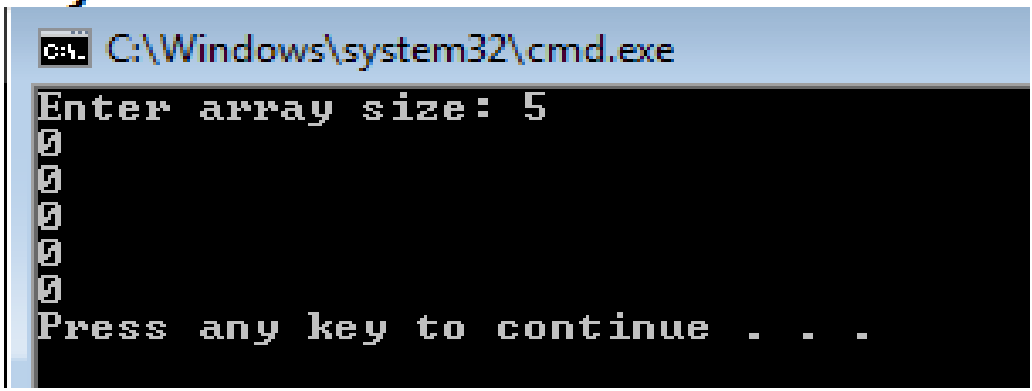
Initializing Dynamic Array Variable

- Dynamic arrays are zero initialized only

```
int size;
cout<<"Enter array size: "; cin>>size;

int *dArr = new int[size]();
for(int i=0; i<size; i++)
    cout<<dArr[i]<<endl;
```

Remember you can also write

```
cout<<*(dArr+i)<<endl;
```



4. Dynamic Arrays – (cont.)

Processing Dynamic Array Variable

- Treated like any standard array.
- `dArr = new double[10];`
  `dArr` contains address of `dArr[0]`
  `dArr+1` evaluates to address of `dArr[1]`
  `dArr+2` evaluates to address of `dArr[2]`
  and so on.

4. Dynamic Arrays – (cont.)

De-allocating Dynamic Array Variable

```
delete [] dArr;
```

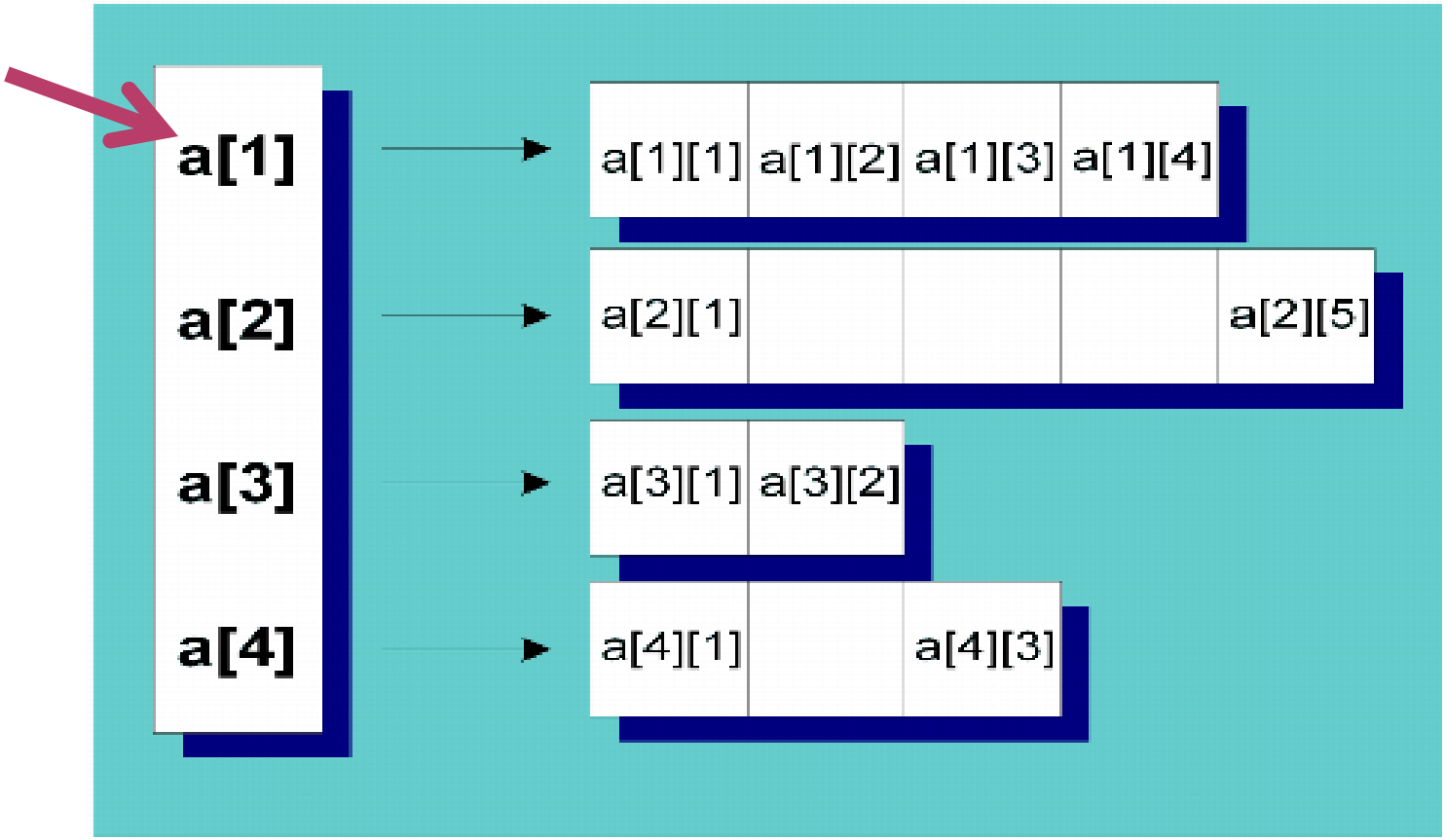
- Returns memory to OS.
- Brackets indicates array;
- Remember `dArr` pointer still exists
  - Should set `dArr = NULL;`

5. Examples

Example 1: 2D dynamic array Lec10Ex1.cpp

- Create a dynamic 2D array with fixed number of rows and columns entered by the user.
- *What if we have different number of columns entered by the user for each row ?*
- *How to de-allocate 2d array.*

5. Examples



5. Examples – (cont.)

Example 1.1:

Repeat Example1 using 2 Functions

1. **Input()**
2. **Display()**

5. Examples – (cont.)

Example 2: Lec10Ex2.cpp

- Creating an array of student structures with variable size and computing average of grades for each student

5. Examples – (cont.)

Example 2.1:

- Modify Example 2 to use 3 functions:

1. FillArray()
2. ComputeAvg()
3. Display()

Class Accumulative Project: Employees Salary for Companies

Tasks 1,2,3,4,5,6 **(DONE☺)**

Solution of Emps tasks 5 and 6

TASK 7 (NEW* BONUS): *Last chance☹*

- Modify your code to have ***dynamic array*** of Employees whose size is read by the user.
- **Add to your Program the following functions:** ***(c is an object of struct Company)***

```
void InputEmpsData( Emp* c.Emps)
void DisplayEmpsInfo(Emp* c.Emps)
```

- Submit your code as text in this form, **from Thursday 19/4/2018** till **due Date Friday 27/4/2018 at 11:59 pm**


Thank
you.