Computer Programming (b) - E1124

(Spring 2021-2022)

Lecture 1

Introduction - Pointers and References

INSTRUCTOR

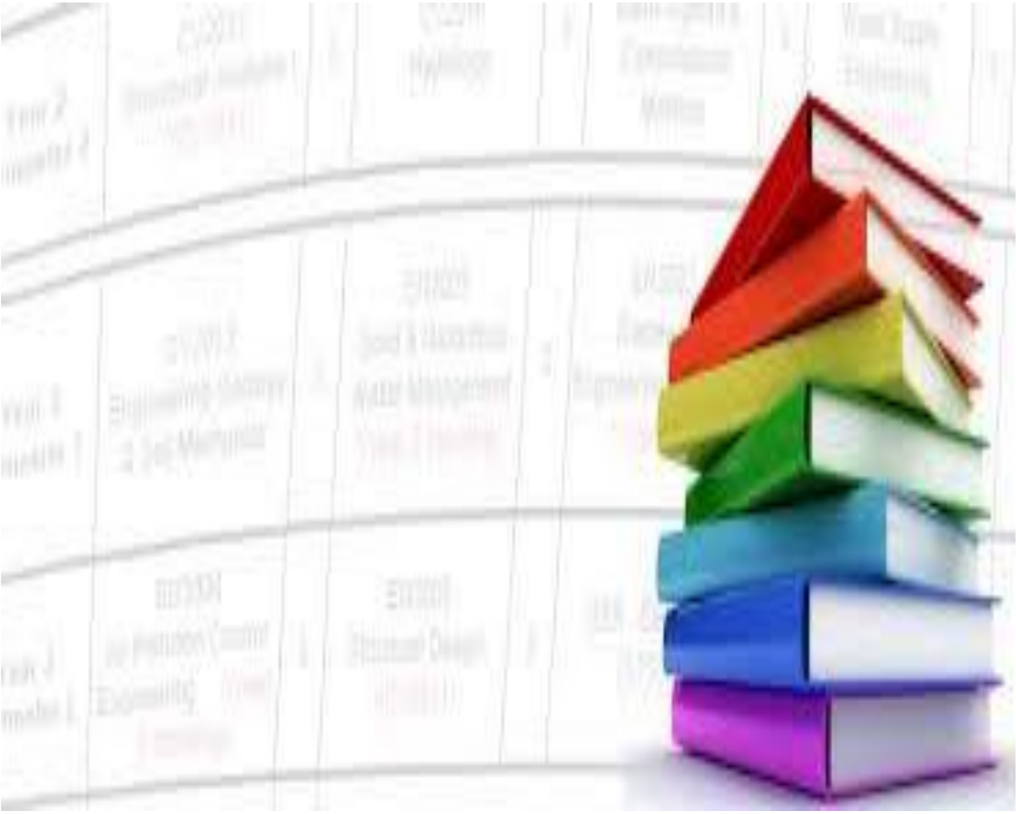
Dr / Ayman Soliman

➢ Contents

1) Course Contents.
2) Grading System & distribution.
3) Course Information.
4) Course Policy.
5) Objectives.
6) Pointers and References.

1) Course Contents.

- Introduction
- Pointers and References.
- Files manipulation
- Searching and sorting algorithms
- Object-oriented design
- Encapsulation and information hiding
- Problem solving with objects.
- Project.



2) Grading System & distribution.

Total score (100)

- Lab. (10)
- Lec. (10)
- Project (10)
- Reports (10)
- Midterm exam (20)
- Final exam (40)

3) Course Information.

Lectures: Tuesday, (9:00 - 9:45 AM)

Office Hours: Saturday, Tuesday, Thursday.

Prerequisite: E1123

References:

- C++ Programming: From Problem Analysis to Program Design, Fifth Edition D.S. Malik
- Object-Oriented Programming Using C++, Fourth Edition Joyce Farrell
- www.learncpp.com

Instructor:

Dr. Ayman Soliman

Ayman.mohamed01@bhit.bu.edu.eg

TAs:

Eng. Enas Mohamed
Eng. Rehab Ibrahim
Eng. Nora Ahmed

Eng. Ahmed Ragab
Eng. Sameh Gamal

4) Course Policy.

- Any forms of **cheating or plagiarism** will result in a **Zero grade** for the required task, report or exam (No discussion nor excuses).

- Students are expected to **respect** Instructors, TAs, and their colleagues.

- Be **on time** and cell phones should be silent or off during the lecture.

- Your grades is based on **merit only** nothing else.

5) Objectives

- Analyze a problem and construct a solution using C++ programming language.
- Explain how an existing C++ program works, discovering errors and fix them.
- Critique a C++ program and describe ways to improve it.
- Follow up intermediate and advanced level of C++ programming language.

Pointers and References

➢ Outlines

- ➢ Objectives
- ➢ Introduction
- ➢ Pointer Variables
- ➢ Initialize and assign a value to a pointer
- ➢ Dereferencing Operator (*)
- ➢ Address of Operator (&)
- ➢ Pointers and Arrays

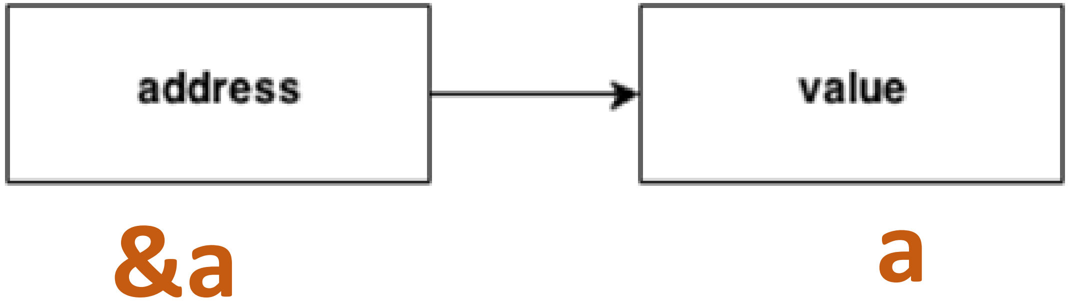
Objectives

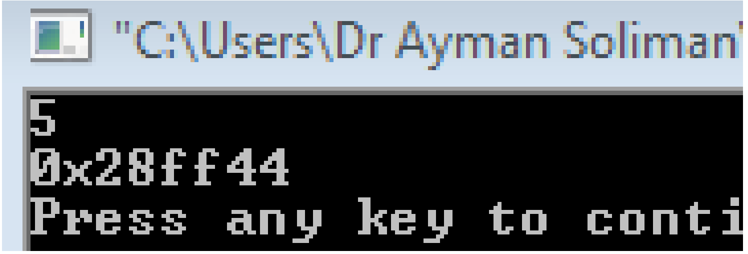
- Learn about the pointer data type and pointer variables
- Explore how to declare and manipulate pointer variables
- Learn about the address of operator and the dereferencing operator
- Learn about pointer Arithmetic
- Pointers and its relations with Arrays

Introduction

- Variable is a name for a piece of memory that holds a value.
- A memory address is automatically assigned to the variable, and any value we assign to the variable is stored in this memory address.
- **Example:-**

```
#include <iostream.h>
int main()
{
            int a=5;
            cout<<a<<endl;     // print the content of a variable
            cout<<&a<<endl;    // print the address of a variable
            return 0;
}
```

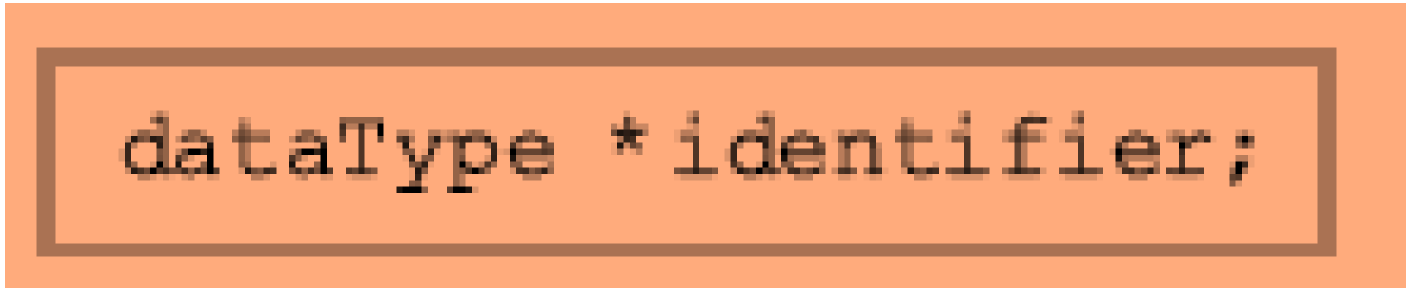




```
5
0x28ff44
Press any key to conti
```

Pointer Variables

- **Pointer variable:** content is a memory address
- Declaring Pointer Variables: Syntax



```
dataType *identifier;
```

- **Examples**:
- int *p;
- char *ch;
- int* fun_1();   // returning a pointer from a function

➢ Initialize and assign a value to a pointer

➢ initialize pointer with address of variable value

```
int value = 10;
int *ptr = & value;
```

➢ assigning pointer

```
int x = 10;
int *ptr ;
ptr=& x;
```

➢ Not allowed initialization

```
int *ptr = 5;
```

or

```
double *ptr = 0x006ffe44;
```

or

```
double value = 10;
int *ptr = & value;
```

//data types must be same

➢ Pointer Variables (cont.)

- ➢ These statements are equivalent

```
int *p;
int* p;
int * p;
```

- ➢ The character * can appear anywhere between type name and variable name

- ➢ In the statement

```
int* p, q;
```

only p is the pointer variable, not q; here q is an int variable

- ➢ The following statement declares both p and q to be pointer variables of the type int

```
int    *p, *q;
```

Dereferencing Operator (*)

- C++ uses * as the binary multiplication operator and as a unary operator
- When used as a unary operator, *
  - ✓ Called dereferencing operator or indirection operator
  - ✓ The dereference operator (*) used to access the value at a particular address:

```
Int x=25;
Int *p;
P = &x;     // store the address of x in p
```

- The following statement prints the value stored in the memory space pointed to by p, which is the value of x. `Cout << *p << endl;`
- The following statement stores 55 in the memory location pointed to by p—that is, in x.

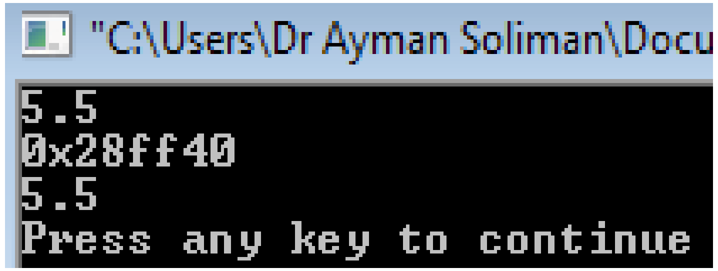
```
*p = 55;
```

Address of Operator (&)

- The ampersand, &, is called the address of operator
- The address of operator is a unary operator that returns the address of its operand

- **Example:-**

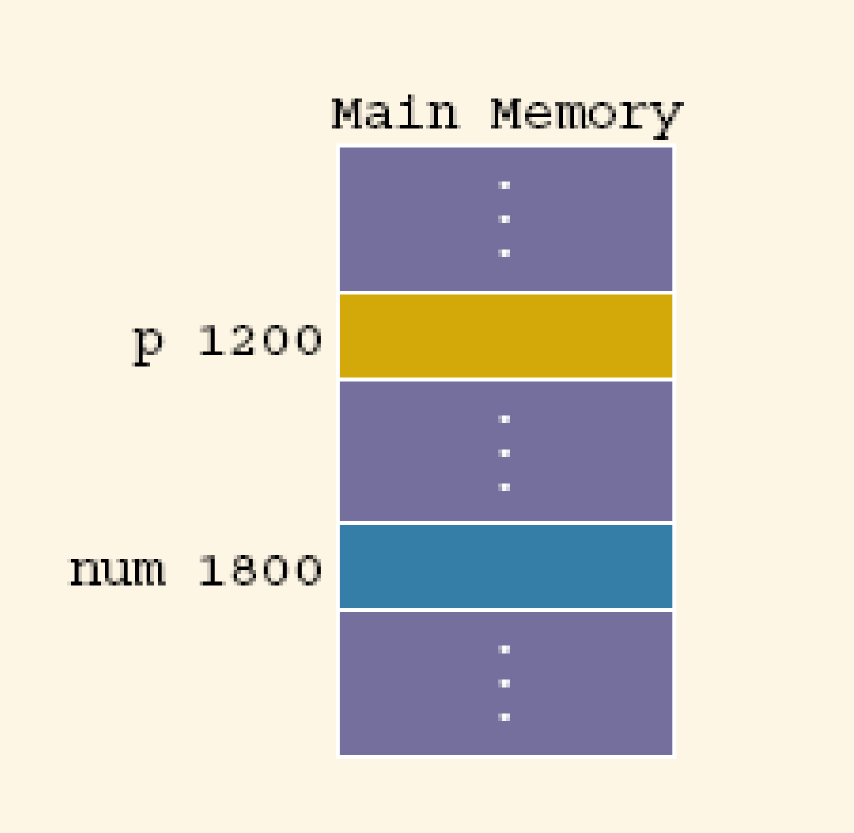
```
#include <iostream.h>
int main()
{
    double b=5.5;
    cout<<b<<endl;      // print the content of b variable
    cout<<&b<<endl;     // print the address of b variable
    cout<<*&b<<endl;    // print the content of of address of b variable
    return 0;
}
```



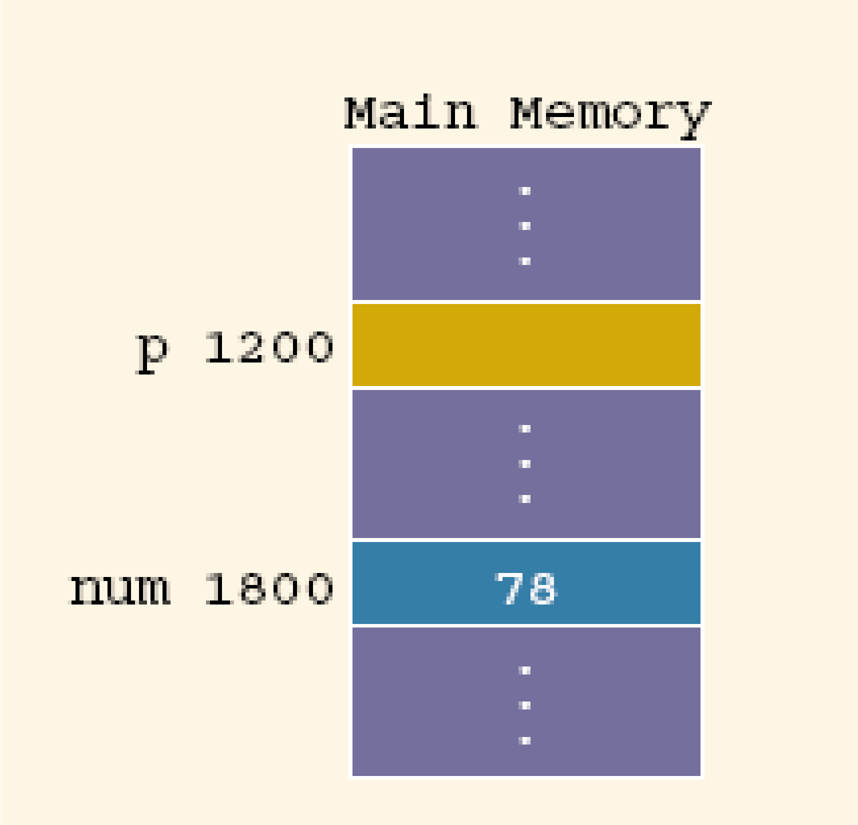
```
5.5
0x28ff40
5.5
Press any key to continue
```

Example 1

- Int *p;
- Int num;

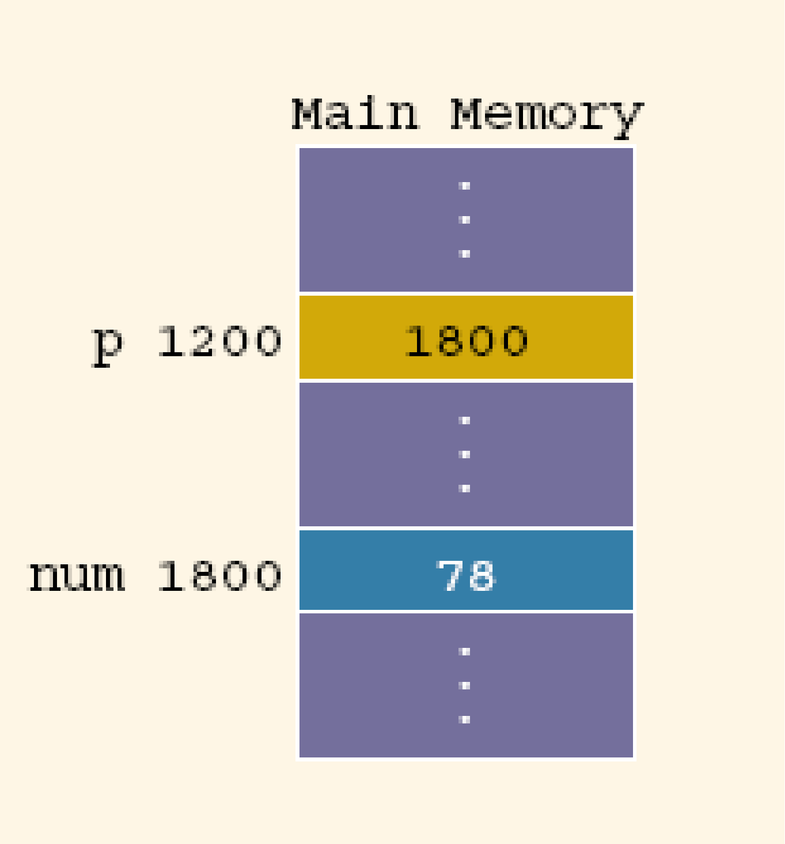


- Num = 78;

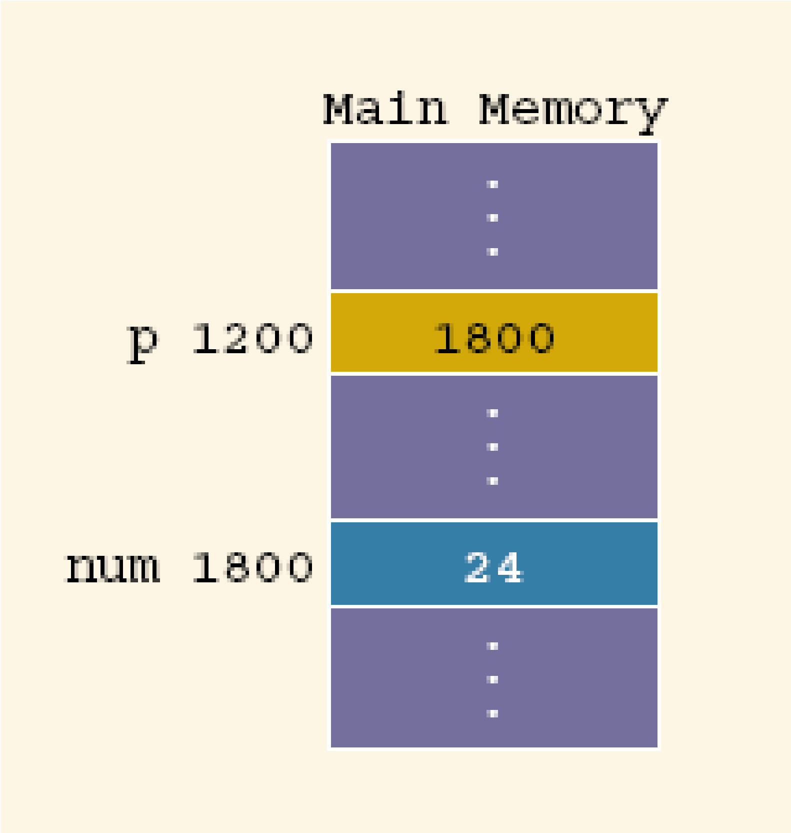


➢ Example 1 (cont.)

- ➢ P = &num;



- ➢ *p = 24;



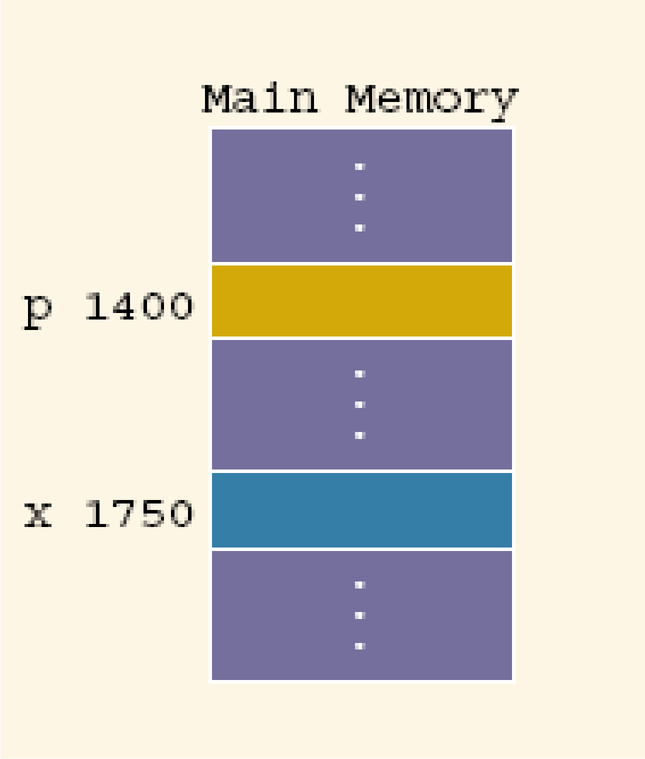
Example 1 (cont.)

- **&p, p, and *p all have different meanings.**

- &p means the address of p.
- p means the content of p.
- *p means the content of the memory location pointed to by p.

Example 2

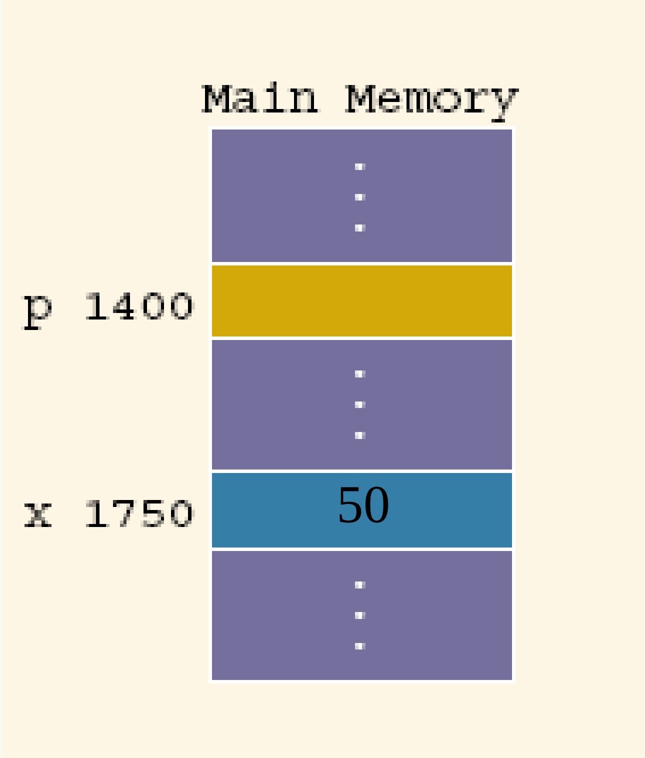
- Int *p;
- Int x;



	value
&p	1400
P	??? (unknown)
*p	does not exist (undefined)
&x	1750
X	??? (unknown)

Example 2 (cont.)

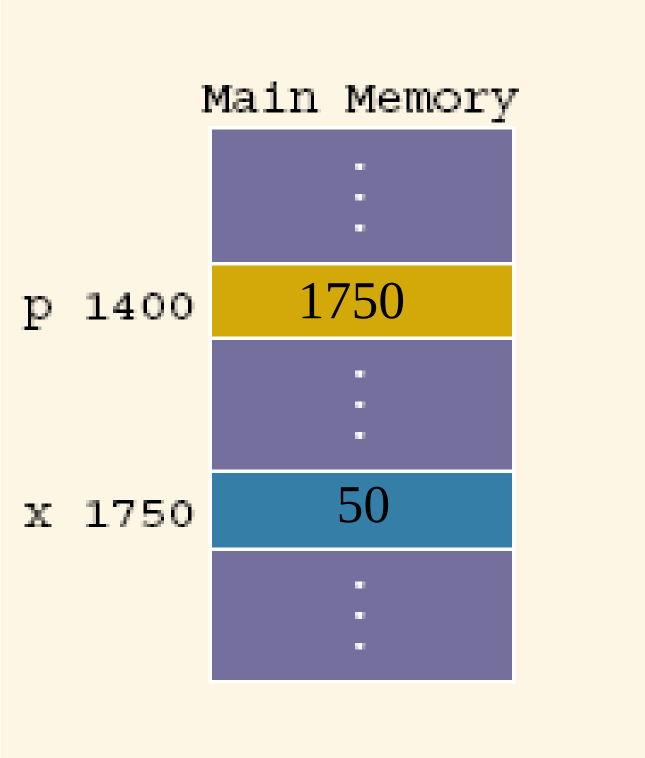
- x = 50;



	value
&p	1400
P	??? (unknown)
*p	does not exist (undefined)
&x	1750
X	50

➢ Example 2 (cont.)

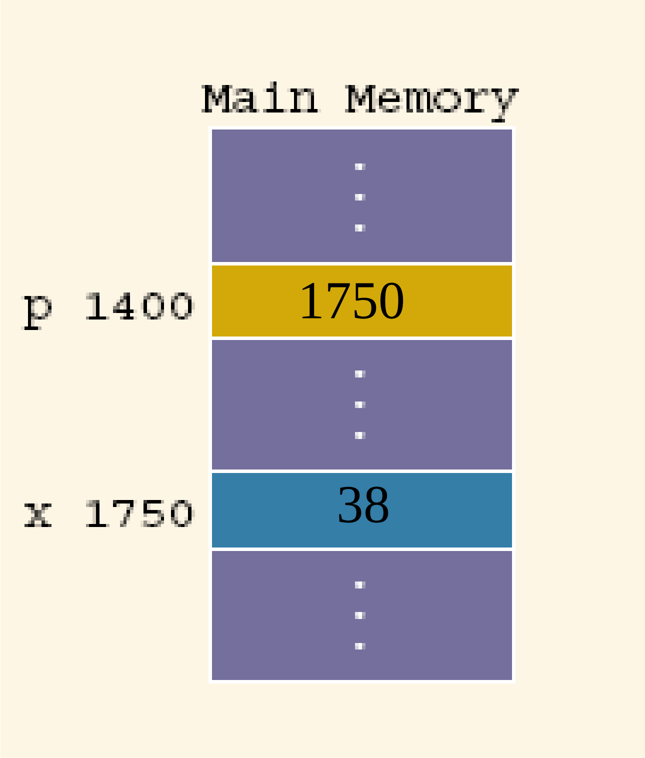
- ➢ p = &x;



	value
&p	1400
P	1750
*p	50
&x	1750
X	50

Example 2 (cont.)

- *p = 38;



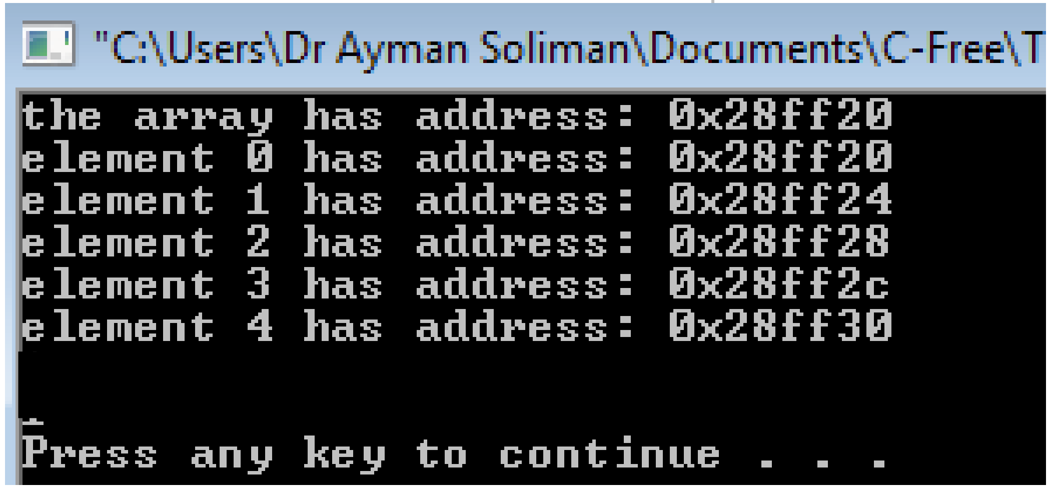
	value
&p	1400
P	1750
*p	38
&x	1750
X	38

Pointers and Arrays

```
#include <iostream.h>
int main()
{
    int array[5]={1,3,5,7,9};
    cout<<"the array has address: "<<array<<endl;      // prints the array address
    cout<<"element 0 has address: "<<&array[0]<<endl;  // prints element 0 address
    cout<<"element 1 has address: "<<&array[1]<<endl;  // prints element 1 address
    cout<<"element 2 has address: "<<&array[2]<<endl;  // prints element 2 address
    cout<<"element 3 has address: "<<&array[3]<<endl;  // prints element 3 address
    cout<<"element 4 has address: "<<&array[4]<<endl;  // prints element 4 address
                                                       // dereferencing an array returns element 0



    return 0;
}
```



"C:\Users\Dr Ayman Soliman\Documents\C-Free\T

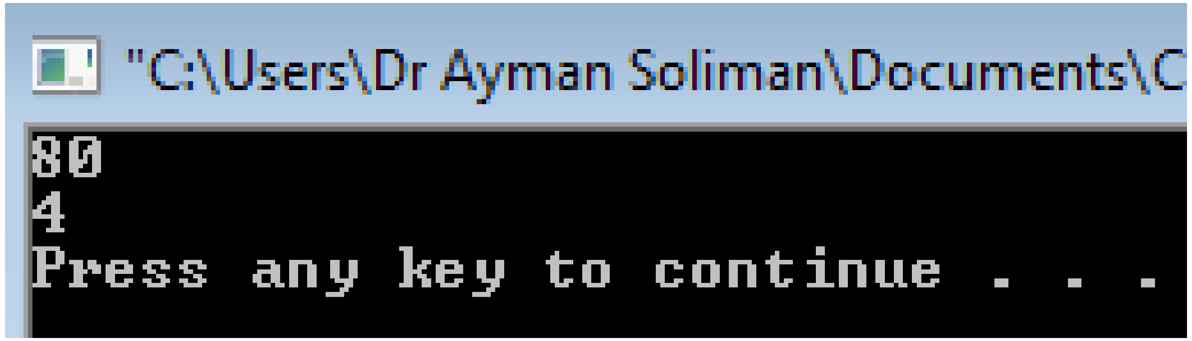
```
the array has address: 0x28ff20
element 0 has address: 0x28ff20
element 1 has address: 0x28ff24
element 2 has address: 0x28ff28
element 3 has address: 0x28ff2c
element 4 has address: 0x28ff30


Press any key to continue . . .
```

➢ Pointers and Arrays (cont.)

```
#include <iostream.h>
// implicity convert parameter array[] to *array
void size(double array[])          // void size(int *array)
{ // array is treated as pointer here, not a fixed array
    cout<<sizeof(array)<<endl;
}

int main()
{
    double array[]={1,3,5,7,9,11,13,15,17,19};
    cout<<sizeof(array)<<endl;      // size of data type * array length
    size(array);
    return 0;
}
```

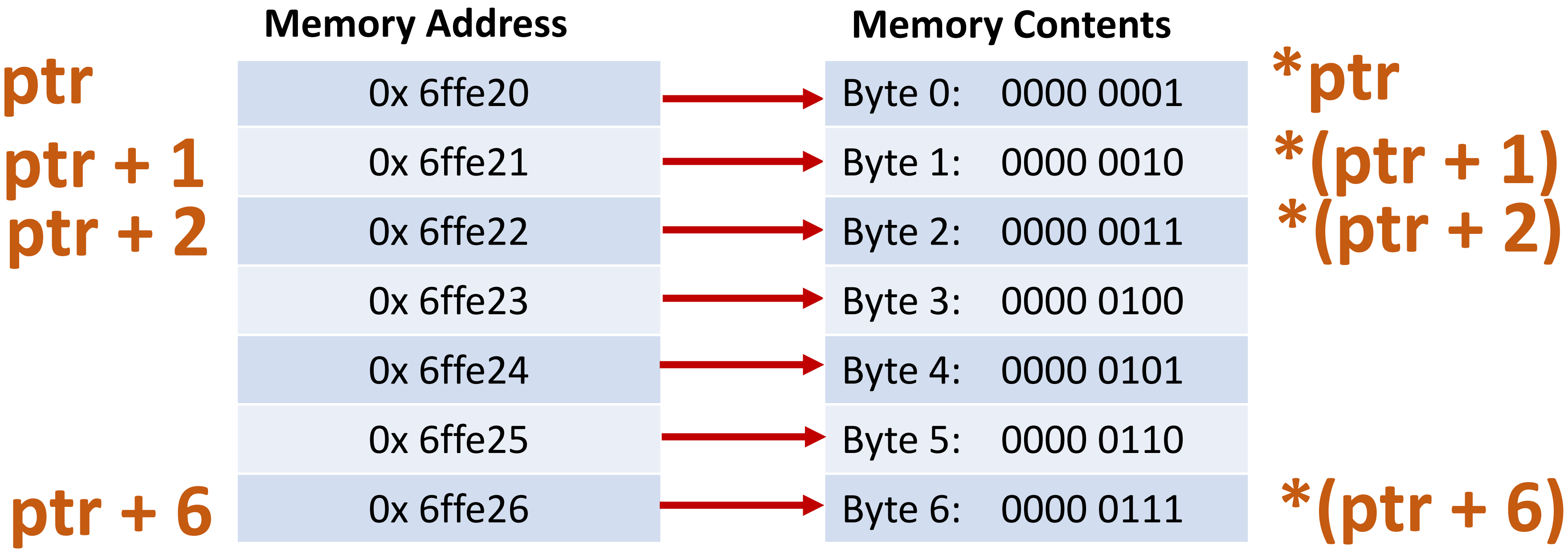


"C:\Users\Dr Ayman Soliman\Documents\C

```
80
4
Press any key to continue . . .
```

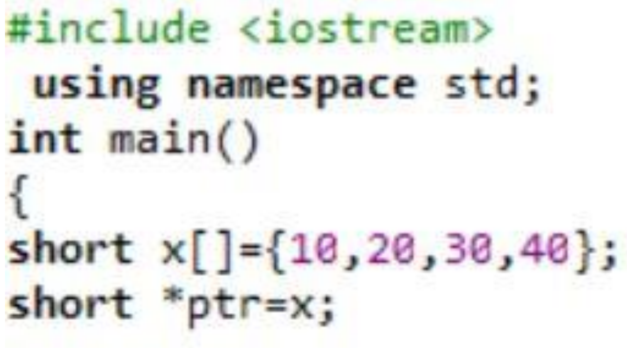
Pointers and Arrays (cont.)

- The C++ allows to perform integer addition or subtraction operations on pointers.

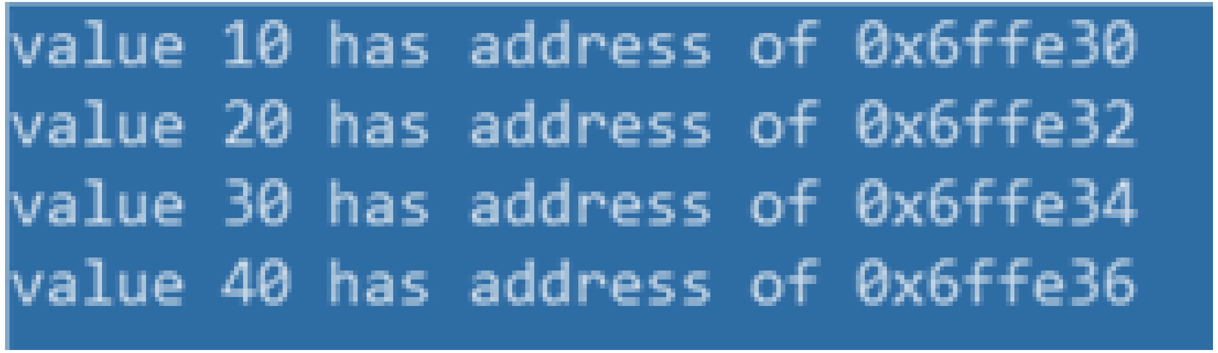


	Memory Address	Memory Contents	
ptr	0x 6ffe20	Byte 0: 0000 0001	*ptr
ptr + 1	0x 6ffe21	Byte 1: 0000 0010	*(ptr + 1)
ptr + 2	0x 6ffe22	Byte 2: 0000 0011	*(ptr + 2)
	0x 6ffe23	Byte 3: 0000 0100	
	0x 6ffe24	Byte 4: 0000 0101	
	0x 6ffe25	Byte 5: 0000 0110	
ptr + 6	0x 6ffe26	Byte 6: 0000 0111	*(ptr + 6)

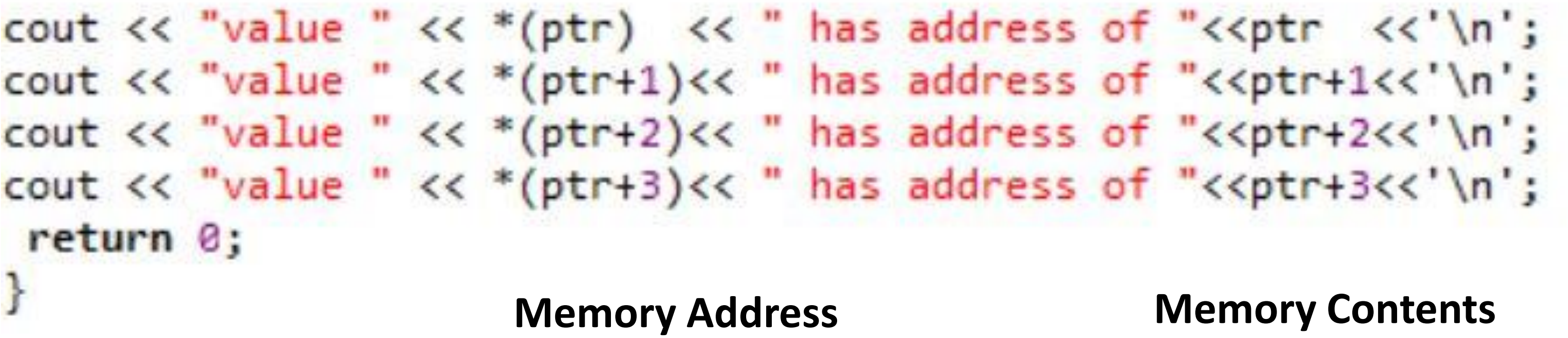
➢ Pointers and Arrays (cont.)



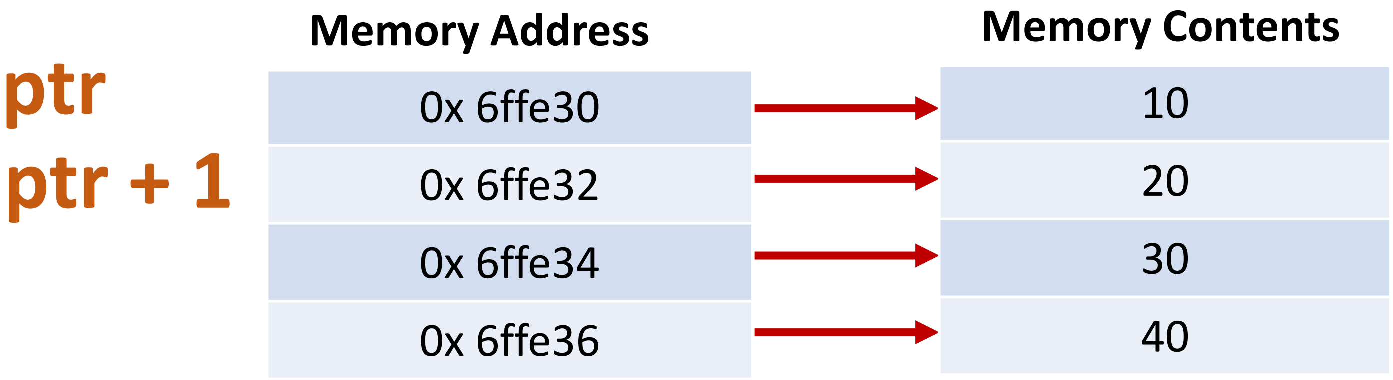
```
#include <iostream>
 using namespace std;
int main()
{
short x[]={10,20,30,40};
short *ptr=x;
```



```
value 10 has address of 0x6ffe30
value 20 has address of 0x6ffe32
value 30 has address of 0x6ffe34
value 40 has address of 0x6ffe36
```



```
cout << "value " << *(ptr)  << " has address of "<<ptr  <<'\n';
cout << "value " << *(ptr+1)<< " has address of "<<ptr+1<<'\n';
cout << "value " << *(ptr+2)<< " has address of "<<ptr+2<<'\n';
cout << "value " << *(ptr+3)<< " has address of "<<ptr+3<<'\n';
 return 0;
}
```

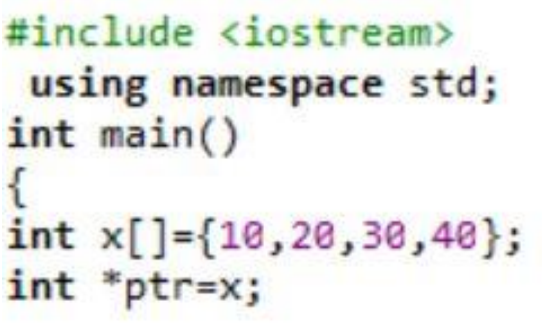


ptr

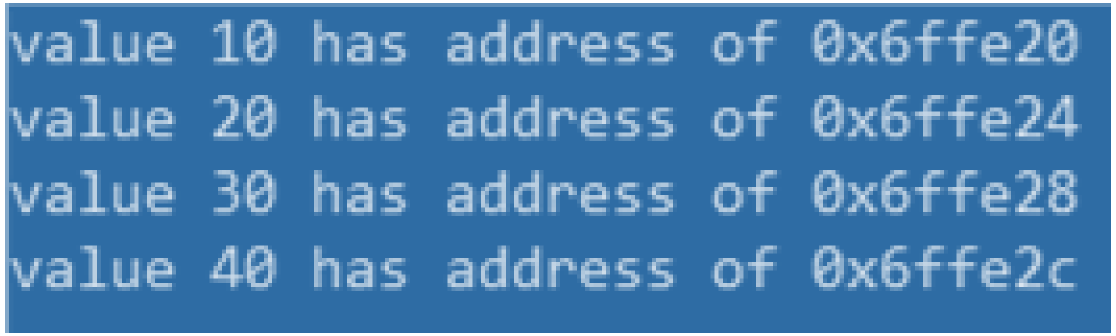
ptr + 1

Memory Address	Memory Contents
0x 6ffe30	10
0x 6ffe32	20
0x 6ffe34	30
0x 6ffe36	40

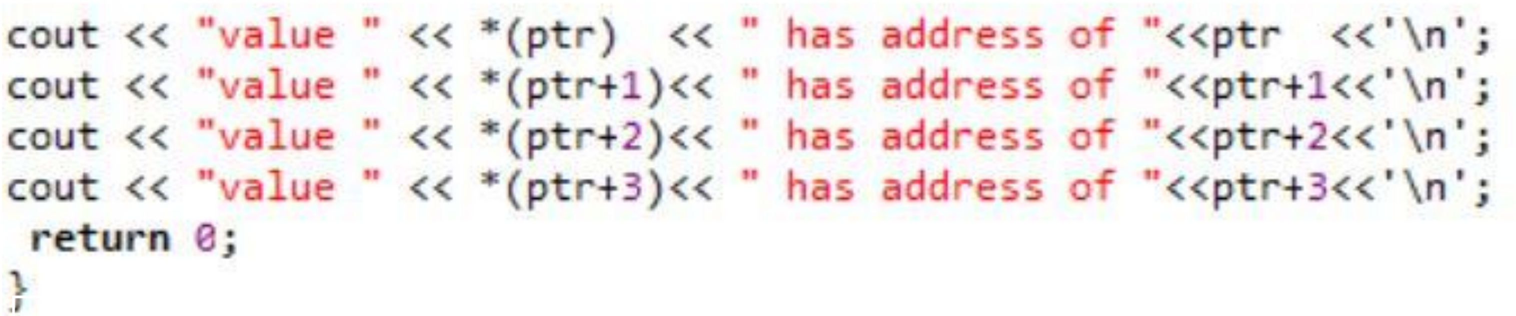
➢ Pointers and Arrays (cont.)



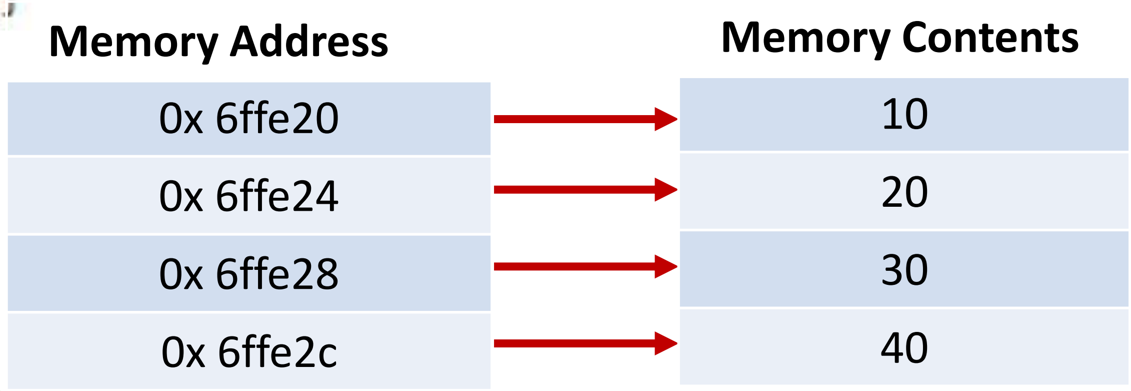
```
#include <iostream>
 using namespace std;
int main()
{
int x[]={10,20,30,40};
int *ptr=x;
```



```
value 10 has address of 0x6ffe20
value 20 has address of 0x6ffe24
value 30 has address of 0x6ffe28
value 40 has address of 0x6ffe2c
```



```
cout << "value " << *(ptr)  << " has address of "<<ptr  <<'\n';
cout << "value " << *(ptr+1)<< " has address of "<<ptr+1<<'\n';
cout << "value " << *(ptr+2)<< " has address of "<<ptr+2<<'\n';
cout << "value " << *(ptr+3)<< " has address of "<<ptr+3<<'\n';
 return 0;
}
```



Memory Address	Memory Contents
0x 6ffe20	10
0x 6ffe24	20
0x 6ffe28	30
0x 6ffe2c	40

The result of a pointer arithmetic expression always multiplies the integer operand by the size of the object being pointed to (scaling).

Thank you