Pointers & Structures

SESSION 9

Pointers

A special variable that contains the “address” of the memory location of another variable

Declaring a pointer variable in C

```
int *p
```

Introduction to Pointers

When we declare a variable some memory is allocated for it. The memory location can be referenced through the identifier “i”. Thus, we have two properties for any variable : its address and its data value. The address of the variable can be accessed through the referencing operator “&”. “&i” gives the memory location where the data value for “i” is stored.

A pointer variable is one that stores an address. We can declare pointers as follows int* p; .This means that p stores the address of a variable of type int.

Introduction to Pointers

Q: Why is it important to declare the type of the variable that a pointer points to? Aren’t all addresses of the same length?

A: It’s true that all addresses are of the same length, however when we perform an operation of the type “p++” where “p” is a pointer variable, for this operation to make sense the compiler needs to know the data type of the variable “p” points to. If “p” is a character pointer then “p++” will increment “p” by one byte (typically), if “p” were an integer pointer its value on “p++” would be incremented by 2 bytes (typically).

Introduction to Pointers

Summary of what was learnt so far:

- Pointer is a data type that stores addresses, it is declared as follows:

  int* a;

  char* p;   etc.
- The value stored in a pointer p can be accessed through the dereferencing operator "*".
- The address of a memory location of a variable can be accessed through the reference operator "&".
- Pointer arithmetic: only addition and subtraction are allowed.

Pointers and Arrays

The concept of array is very similar to the concept of pointer. The identifier of an array actually a pointer that holds the address of the first element of the array.

Therefore if you have two declarations as follows:

- "int a[10];" "int* p;" then the assignment "p = a;" is perfectly valid
- Also "*(a+4)" and "a[4]" are equivalent as are "*(p+4)" and "p[4]" .
- The only difference between the two is that we can change the value of "p" to any integer variable address whereas "a" will always point to the integer array of length 10 defined.

Character Pointers, Arrays and Strings

What is a String?

- A string is a character array that is ‘\0’ terminated.
- E.g. “Hello”

What is a Character array?

- It is an array of characters, not necessarily ‘\0’ terminated
- E.g. char test[4] = {‘a’, ‘b’, ‘c’, ‘d’}; <this char array is not zero terminated>

What is a character pointer?

- It is a pointer to the address of a character variable.
- E.g. char* a; <this pointer is not initialized>

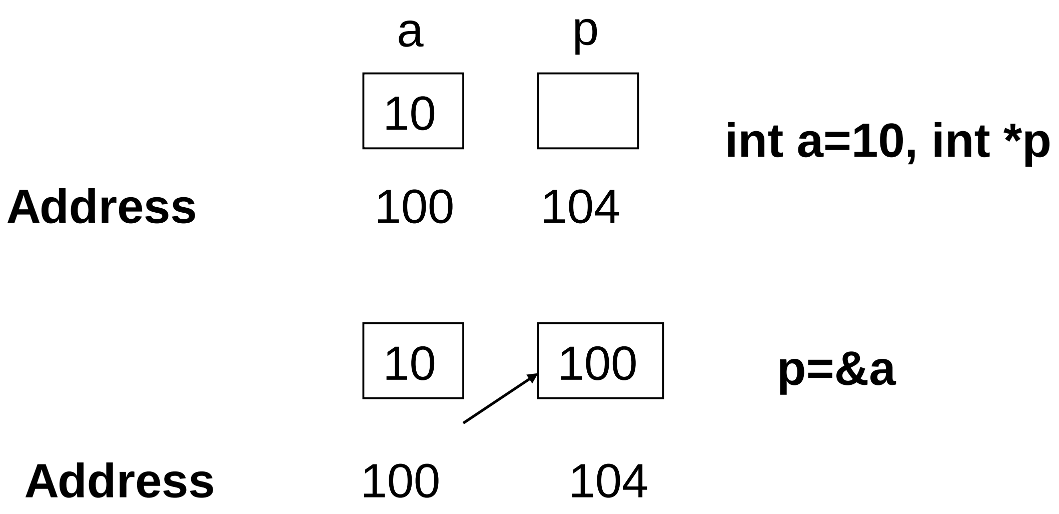
Examples

char* a = “Hello”;

- a -> gives address of ‘H’
- *a -> gives ‘H’
- a[0] -> gives ‘H’
- a++ -> gives address of ‘e’
- *a++ -> gives ‘e’
- a = &b; where b is another char variable is perfectly LEGAL. However “char a[100];” “a =&b;” where b is another char variable is ILLEGAL.

Assigning an Address to a pointer

```
int a = 10
int *p
p = &a
```



Pointers (Contd.)

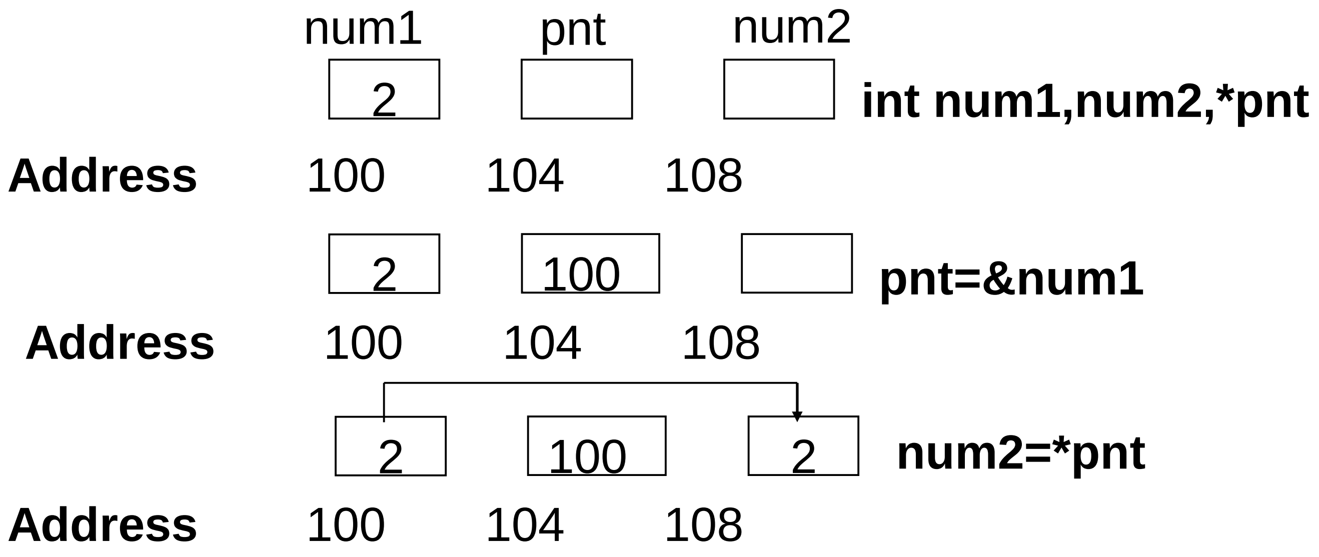
```
char c = ‘s’, *cp
cp = &c
```

c is a variable of type character

cp is a pointer that points to c

Retrieving Values from a Pointer

```
int num1=2,num2,*pnt
pnt=&num1
num2=*pnt
```



Pointers in C#

POINTER

A **pointer** is a programming language object, whose value refers to (or "points to") another value stored elsewhere in the computer memory using its memory address.

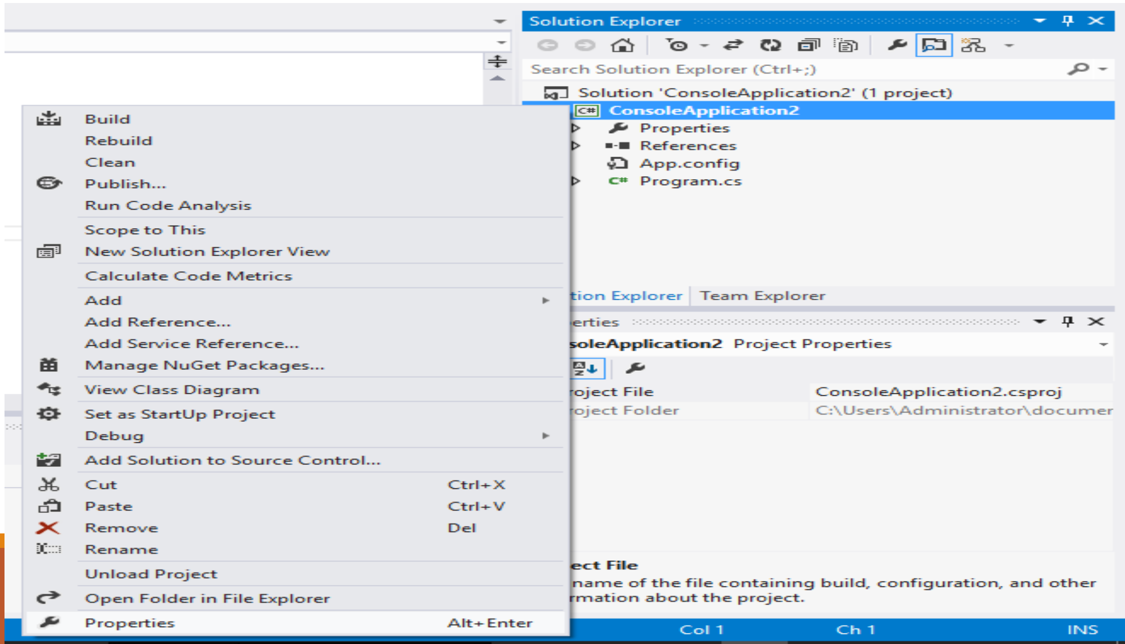
A **pointer** is a variable whose value is the address of another variable i.e., the direct address of the memory location. similar to any variable or constant, you must declare a pointer before you can use it to store any variable address.

Pointer types are not tracked by the default garbage collection mechanism.

Unsafe Codes

The C# statements can be executed either as in a safe or in an unsafe context. The statements marked as unsafe by using the keyword unsafe runs out side the control of Garbage Collector. Remember that in C# any code involving pointers requires an unsafe context.

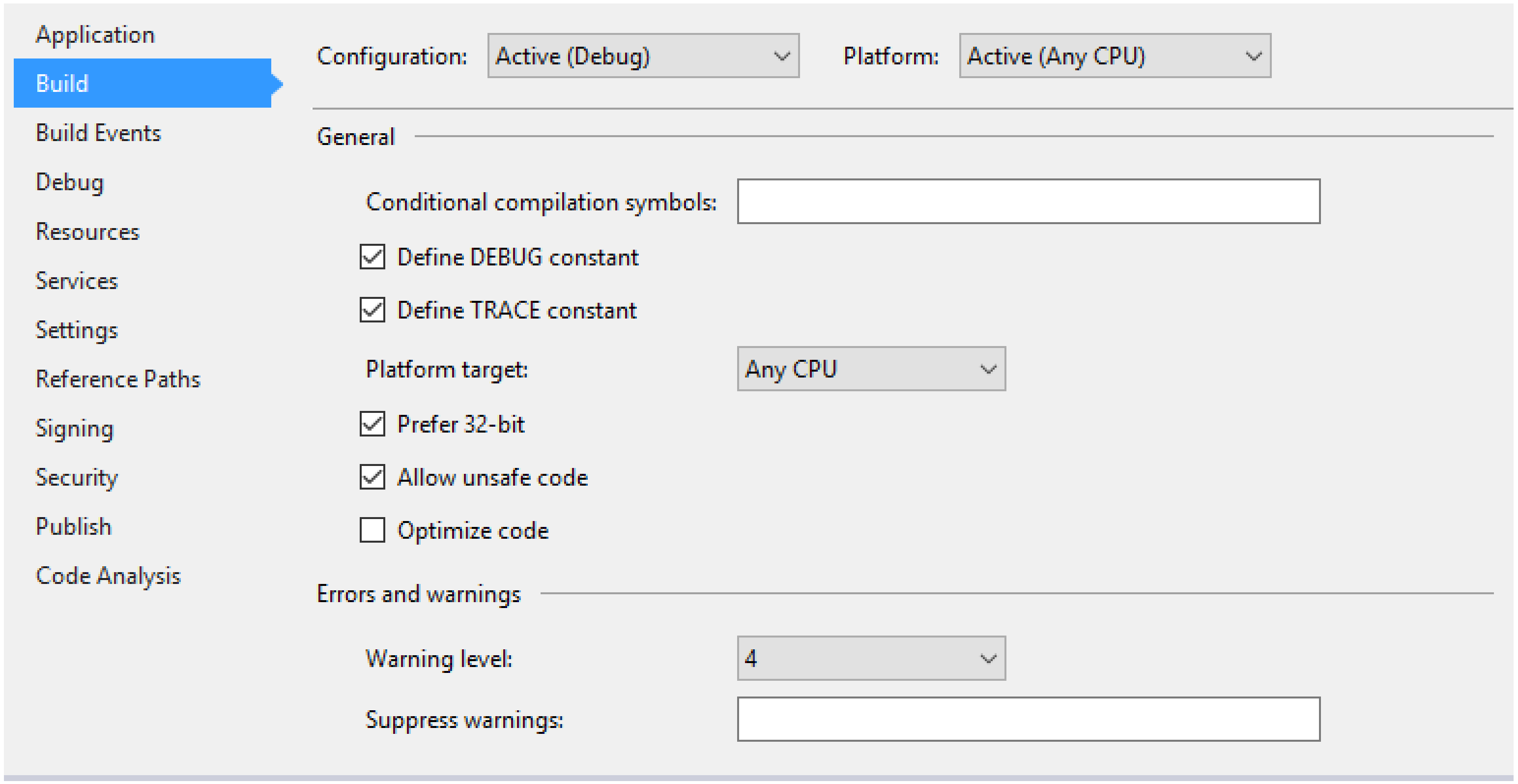
Remember to enable unsafe code in the **Project Designer**; choose **Project**, **Properties** on the menu bar, and then select **Allow unsafe code** in the **Build** tab.



roject File	ConsoleApplication2.csproj
roject Folder	C:\Users\Administrator\documer

name of the file containing build, configuration, and other rmation about the project.

Allow unsafe code



Configuration: Active (Debug)

Platform: Active (Any CPU)

General

Conditional compilation symbols:

- [✓] Define DEBUG constant
- [✓] Define TRACE constant

Platform target: Any CPU

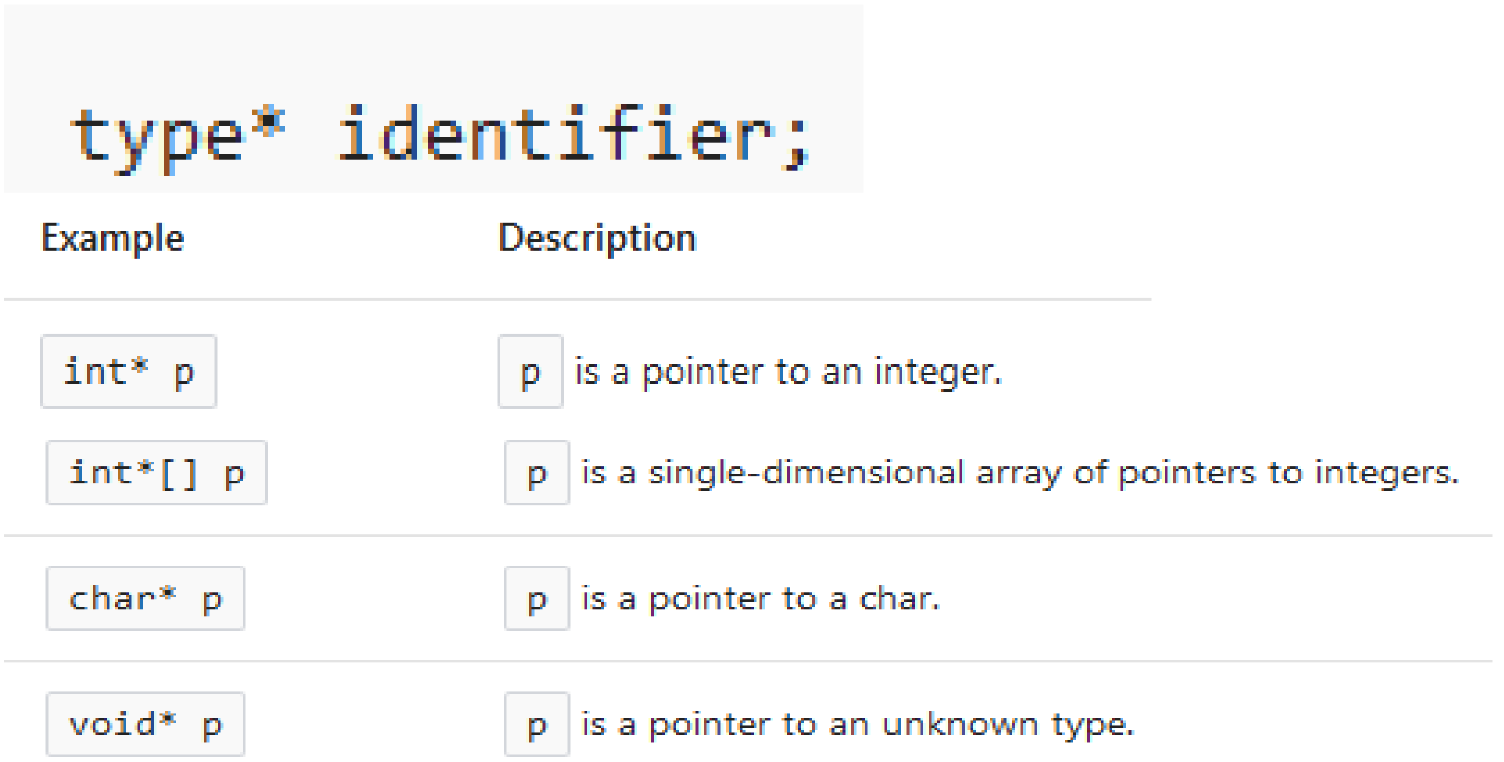
- [✓] Prefer 32-bit
- [✓] Allow unsafe code
- [ ] Optimize code

Errors and warnings

Warning level: 4

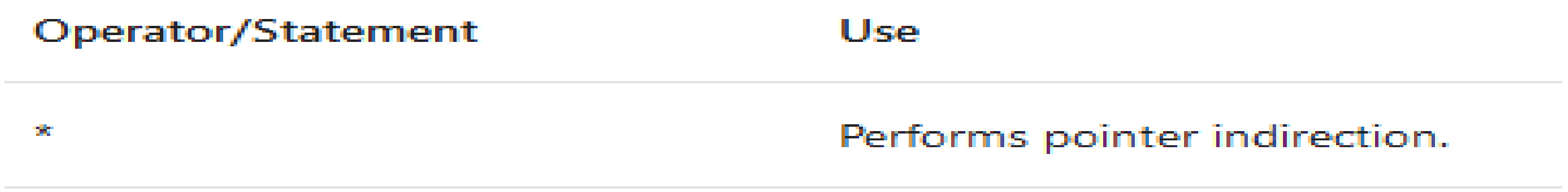
Suppress warnings:

representation



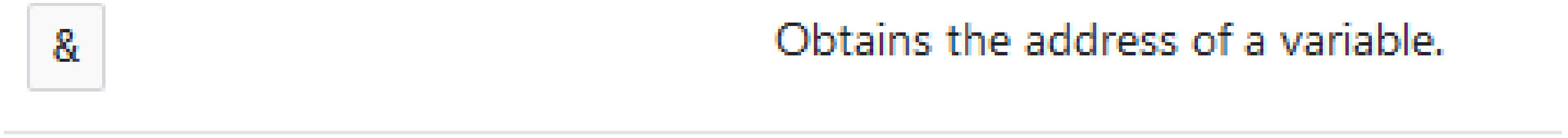
```
type* identifier;
```

Example	Description
`int* p`	`p` is a pointer to an integer.
`int*[] p`	`p` is a single-dimensional array of pointers to integers.
`char* p`	`p` is a pointer to a char.
`void* p`	`p` is a pointer to an unknown type.

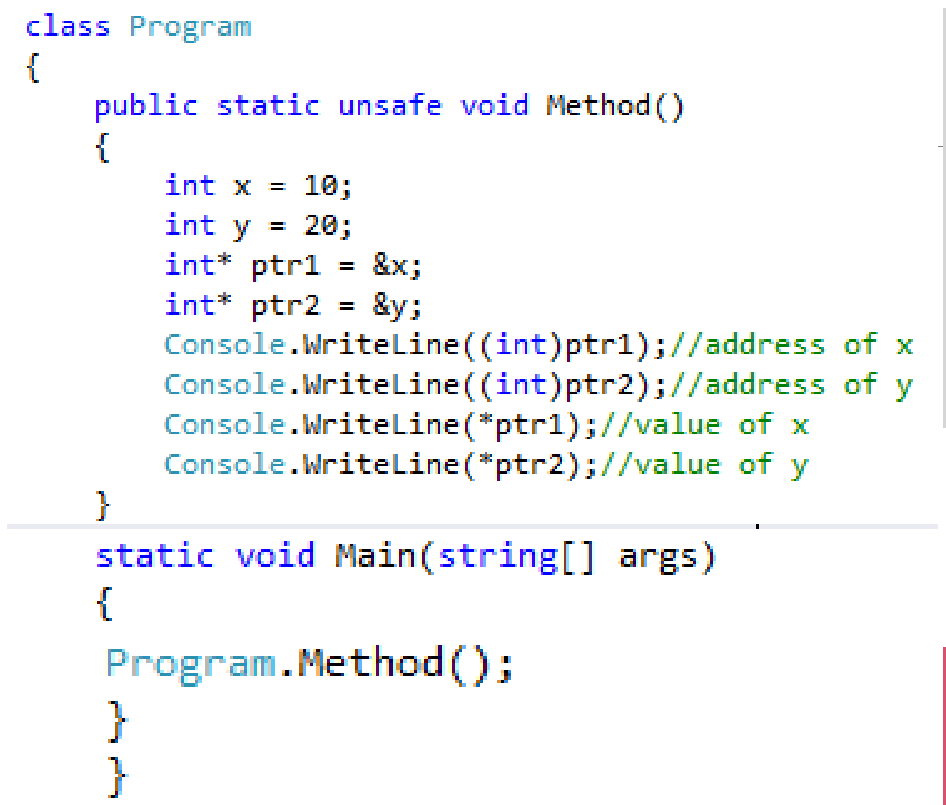


Operator/Statement	Use
*	Performs pointer indirection.

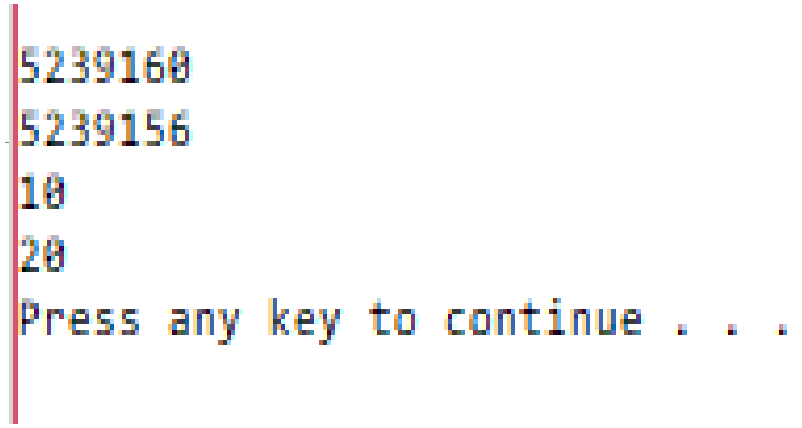
The pointer indirection operator * can be used to access the contents at the location pointed to by the pointer variable.



&	Obtains the address of a variable.

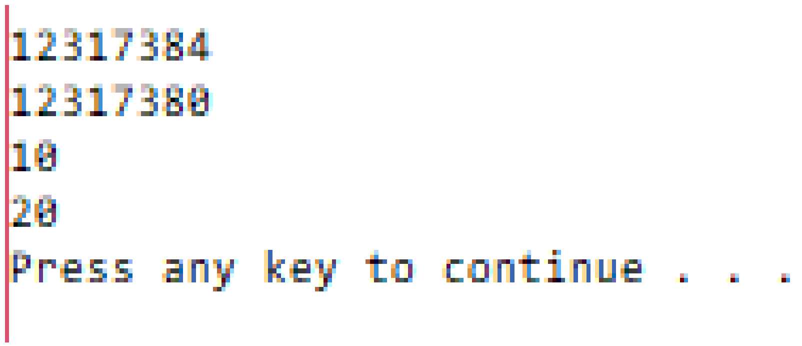


```
class Program
{
    public static unsafe void Method()
    {
        int x = 10;
        int y = 20;
        int* ptr1 = &x;
        int* ptr2 = &y;
        Console.WriteLine((int)ptr1);//address of x
        Console.WriteLine((int)ptr2);//address of y
        Console.WriteLine(*ptr1);//value of x
        Console.WriteLine(*ptr2);//value of y
    }
    static void Main(string[] args)
    {
     Program.Method();
     }
     }
```



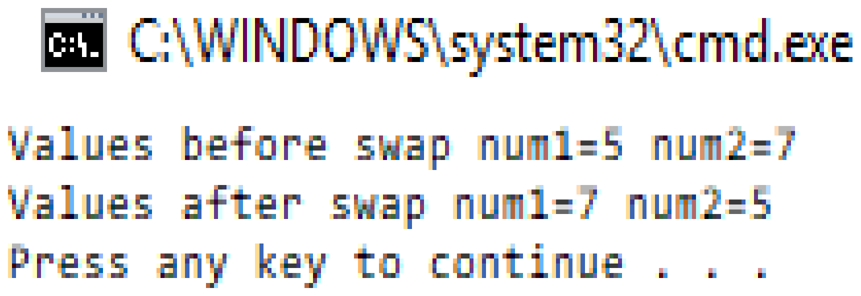
```
5239160
5239156
10
20
Press any key to continue . . .
```

ON RUNNING THE CODE AGAIN



```
12317384
12317380
10
20
Press any key to continue . . .
```

```
public static unsafe void Swap(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```



C:\WINDOWS\system32\cmd.exe

```
Values before swap num1=5 num2=7
Values after swap num1=7 num2=5
Press any key to continue . . .
```

```
static unsafe void Main(string[] args)
{

int num1 = 5;
int num2 = 7;
int* x = &num1;
int* y = &num2;
Console.WriteLine("Values before swap num1=" +*x + " num2=" +*y);
Program.Swap(x,y);
Console.WriteLine("Values after swap num1=" + *x + " num2=" + *y);

}
}
```

Structure

A data type that holds different types of data within a single group

```
struct Books {
   public string title;
   public string author;
   public string subject;
   public int book_id;
};
```

Defining a Structure

```
Start
Structure employee
{
char name[10]
char address[20]
float salary
}
End
```

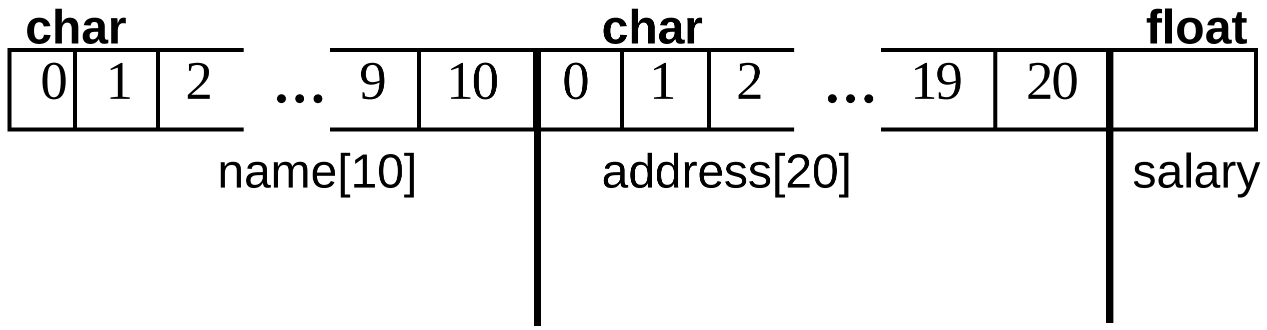
Structure keyword declares a structure in algorithms

name, address and salary are the members of the structure

employee is the structure name

Defining a Structure (Contd.)

- The members within the structure **employee** can be visualized as:



Defining a Structure (Contd.)

```
Start
Structure employee
{
char name[10]
char address[20]
float salary
}
Structure employee e1
End
```

- Structure **e1** of the type employee is created

Accessing the Members of a Structure

Members of structure are accessed as:

Structure variable.Member variable

Example: To access members of structure **e1**

```
e1.name

e1.address

e1.salary
```

Structure variables can be assigned values

```
e1.name = “Jackson”
e1.address = “15/2, New York”
e1.salary = 500,000
```

Structures (Example)

To calculate the area of a rectangle:

```
Start
Structure rectangle
{
int length
int breadth
}
Structure rectangle rect
declare area as integer
rect.length = 10
rect.breadth = 2
area = rect.length * rect.breadth
End
```

Variant of a Structure

```
Start
Structure rectangle
{
int length
int breadth
}
Structure rectangle rect = {10,2}
End
```

Variable **rect** is defined and the values 10 and 2 is assigned for its members

```
struct Books {

  public string title;

  public string author;

  public string subject;

  public int book_id;

};

public class testStructure {

  public static void Main(string[] args) {

    Books Book1;

    Books Book2;

Book1.title = "C# Programming";

    Book1.author = "Nuha Ali";

    Book1.subject = "C# Programming Tutorial";

    Book1.book_id = 6495407;

    Book2.title = "Telecom Billing";

     Book2.author = "Zara Ali";

     Book2.subject =  "Telecom Billing Tutorial";

     Book2.book_id = 6495700;

      Console.WriteLine( "Book 1 title : {0}", Book1.title);

     Console.WriteLine("Book 1 author : {0}", Book1.author);

     Console.WriteLine("Book 1 subject : {0}", Book1.subject);

    Console.WriteLine("Book 1 book_id :{0}", Book1.book_id);

     Console.WriteLine("Book 2 title : {0}", Book2.title);

     Console.WriteLine("Book 2 author : {0}", Book2.author);

     Console.WriteLine("Book 2 subject : {0}", Book2.subject);

     Console.WriteLine("Book 2 book_id : {0}",
Book2.book_id);

      Console.ReadLine();

  }

}
```

Structures

- Custom data Types
- Can have methods
- Can have constructors
- Cannot implement inheritance

```
…
struct TestStruct
{
public TestStruct()
       {
       //Constructor Implementation
       }

public Method1()
       {
       //Method1 Implementation
       {

public int dataMember;
}
…
```

```
 struct Books
    {
        public string title;
        public string author;
        public string subject;
        public int book_id;
        public Books(string t, string a, string s, int id)
        {
            title = t;
            author = a;
            subject = s;
            book_id = id;
        }
        public void Display()
        {
            Console.WriteLine("Title : {0}", title);
            Console.WriteLine("Author : {0}", author);
            Console.WriteLine("Subject : {0}", subject);
            Console.WriteLine("Book_Id :{0}\n",
book_id);
        }
    };
```

```
class Program
    {
        public static void Main(string[] args)
        {
            Books Book1 = new Books("C#
Programming", "Nuha Ali", "C# Programming
Tutorial", 6495407);

            Books Book2 = new Books("Telecom Billing",
"Zara Ali", "Telecom Billing Tutorial", 6495700);

            Book1.Display();
            Book2.Display();
           Console.ReadLine();
        }
    }
```

Enumerators (1)

- They are a set of named constants.

```
using System;
public class food
{
 public enum foodType
        {
        Pizza,
        Pasta,
        Spaghetti,
        }

 public void GetFoodOrder(String CustName, foodType order)
        {
        //Process FoodOrder
        }

 static void Main()
 {
 food myDinner = new food();

 myDinner.GetFoodOrder(“Scooby”, food.foodType.Pizza);
 }
}
```

Enumerators (2)

- Enumerators in C# have numbers associated with the values.

- By default, the first element of enum is assigned a value of 0 and is incremented for each subsequent enum element.

Enumerators (3)

- The default value can be overridden during initialization.

```
…
public enum foodType
      {
      Pizza = 1,
      Pasta = 3,
      Spaghetti = 5,
      }
…
```