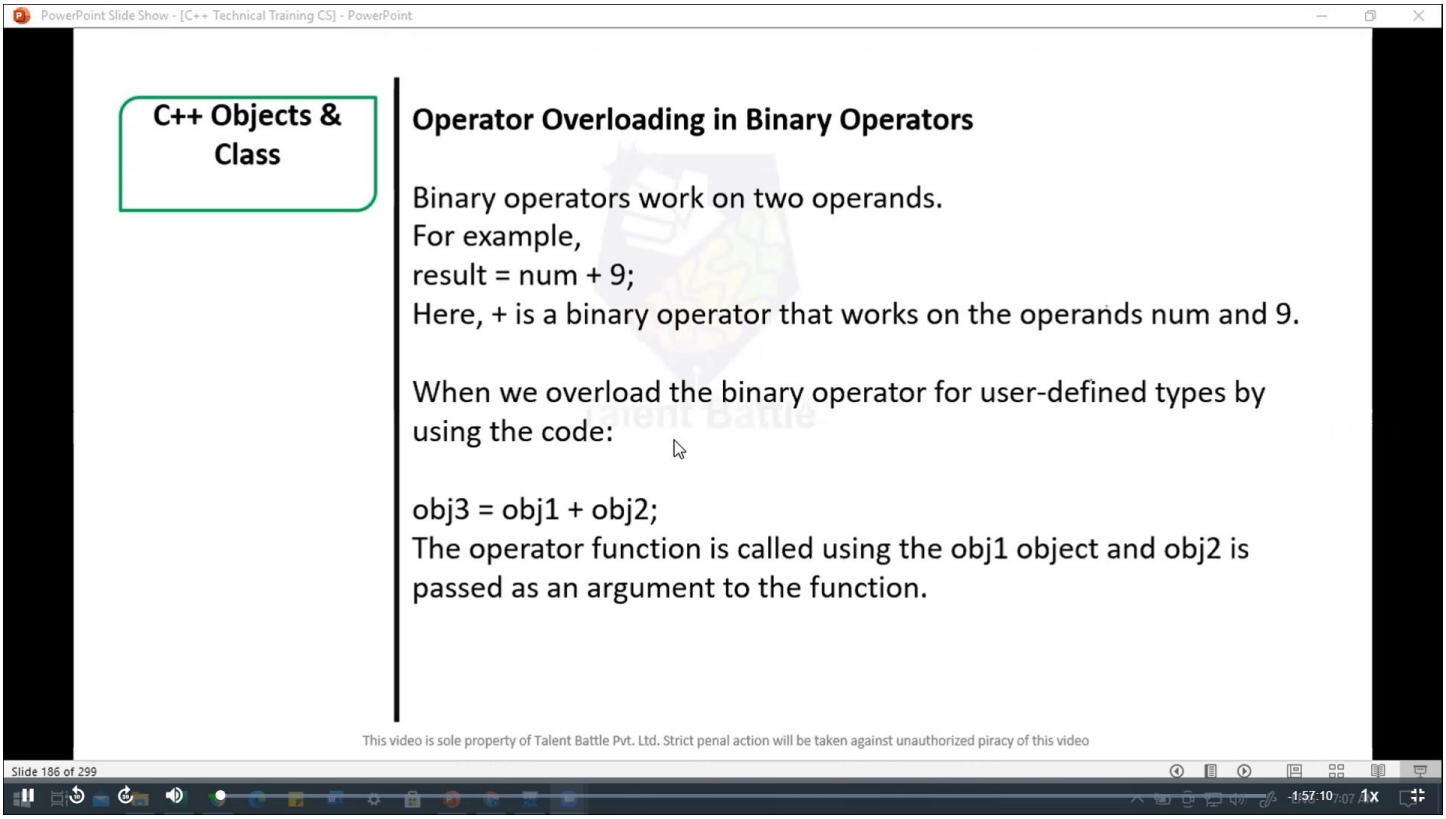
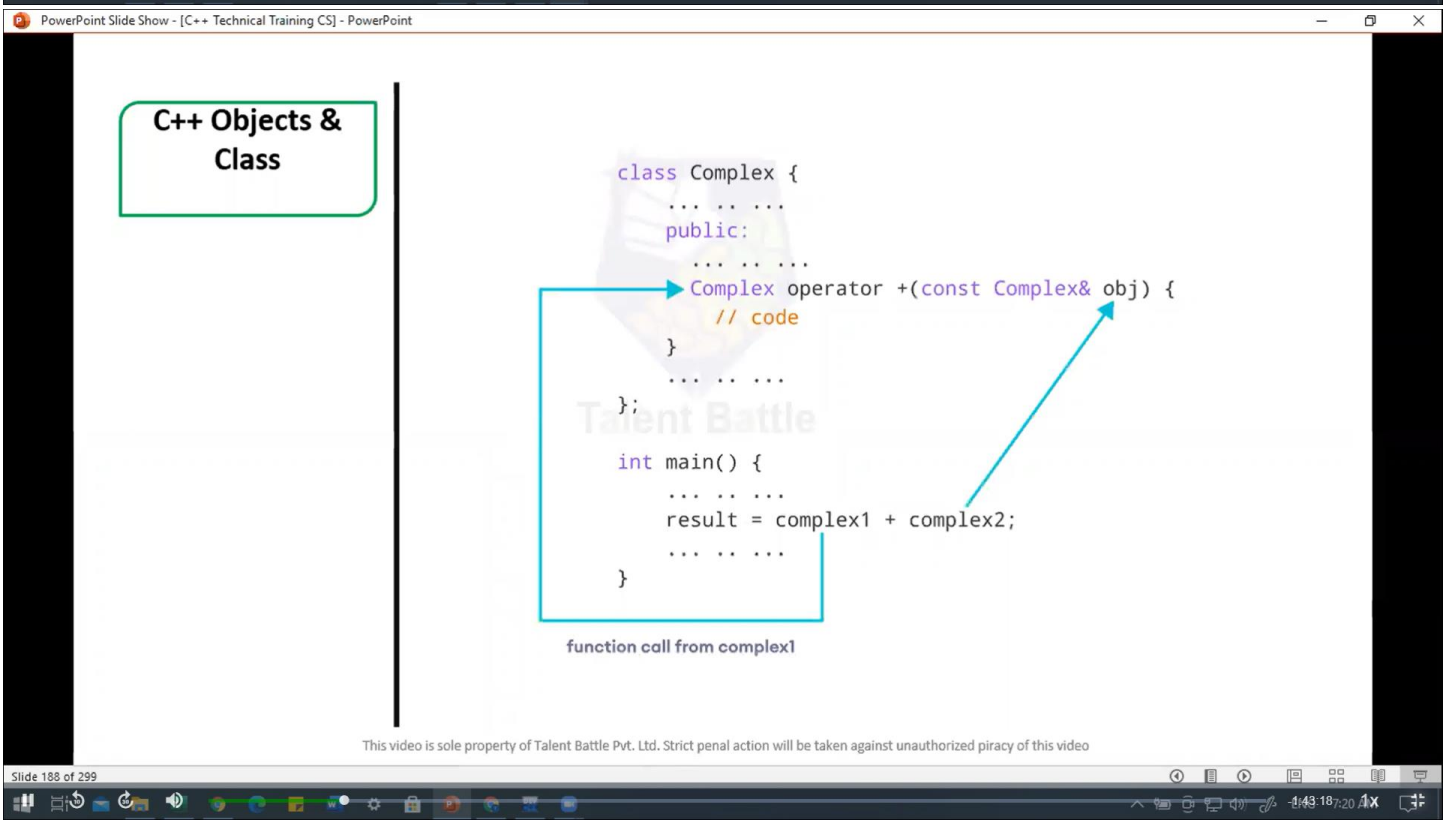

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint
C++ Objects & Class
Operator Overloading in Binary Operators
Binary operators work on two operands.
For example,
result = num + 9;
Here, + is a binary operator that works on the operands num and 9.
When we overload the binary operator for user-defined types by using the code:
obj3 = obj1 + obj2;
The operator function is called using the obj1 object and obj2 is passed as an argument to the function.
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Slide 186 of 299


PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint
C++ Objects & Class
class Complex {
... .
public:
... .
Complex operator +(const Complex& obj) {
// code
}
... .
};
int main() {
... .
result = complex1 + complex2;
... .
}
function call from complex1
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Slide 188 of 299

```cpp
// C++ program to overload the binary operator +
//This program adds two complex numbers

#include<iostream>
using namespace std;

class Complex{
  private:
    float real;
    float imag;

  public:
    // Constructor to initialize real and image to 0
    Complex() : real(0), imag(0){}

    void input(){
      cout << "Enter real and imaginary parts respectively: ";
      cin >> real;
      cin >> imag;
    }

    // Overload the + operator
    Complex operator +(const Complex& obj){
      Complex temp;
      temp.real = real + obj.real;
      temp.imag = imag + obj.imag;
      return temp;
    }

    void output(){
      if(imag < 0){
        cout << "Output Complex number: " << real << imag << "i";
      }
      else{
        cout << "Output Complex number: " << real << " + " << imag << "i";
      }
    }

};

int main(){
  Complex complex1, complex2, result;

  cout << "Enter first complex number:\n";
  complex1.input();

  cout << "Enter second complex number:\n";
  complex2.input();
```

```
  // complex1 call the operator function
  // complex2 is passed as an argument to the function
  result = complex1 + complex2;
  result.output();

  return 0;
}



/*
Enter first complex number:
Enter real and imaginary parts respectively: 4
3
Enter second complex number:
Enter real and imaginary parts respectively: 3
5
Output Complex number: 7 + 8i
--------------------------------
Process exited after 4.714 seconds with return value 0
Press any key to continue . . .
*/
```

C++ Objects & Class

Things to Remember in C++ Operator Overloading

Two operators = and & are already overloaded by default in C++. For example, to copy objects of the same class, we can directly use the = operator. We do not need to create an operator function.

Operator overloading cannot change the precedence and associativity of operators. However, if we want to change the order of evaluation, parentheses should be used.

There are 4 operators that cannot be overloaded in C++. They are:

1. :: (scope resolution)
2. . (member selection)
3. .* (member selection through pointer to function)
4. ?: (ternary operator)

C++ Pointers

C++ Pointers

In C++, pointers are variables that store the memory addresses of other variables.

Address in C++

If we have a variable var in our program, &var will give us its address in the memory.

```cpp
//================================================
// C++ pointer
#include <iostream>
using namespace std;

int main(){
  // declare variiables
  int var1 = 3;
  int var2 = 24;
  int var3 = 17;

  // print address of var1
  cout << "Address of var1: " << &var1 << endl;

  // print address of var2
  cout << "Address of var2: " << &var2 << endl;

  // print address of var3
  cout << "Address of var3: " << &var3 << endl;

  return 0;
}



/*
Address of var1: 0x6ffe1c
Address of var2: 0x6ffe18
Address of var3: 0x6ffe14

--------------------------------
Process exited after 0.06599 seconds with return value 0
*/
```

C++ Pointers

C++ Pointers

As mentioned above, pointers are used to store addresses rather than values.

Here is how we can declare pointers.

```
int *pointVar;
```

Here, we have declared a pointer pointVar of the int type.

We can also declare pointers in the following way.

```
int* pointVar; // preferred syntax
```

Let's take another example of declaring pointers.

```
int* pointVar, p;
```

Here, we have declared a pointer pointVar and a normal variable p.

Note: The * operator is used after the data type to declare pointers.

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers

Assigning Addresses to Pointers

Here is how we can assign addresses to pointers:

```
int* pointVar, var;
var = 5;

// assign address of var to pointVar pointer
pointVar = &var;
```

Here, 5 is assigned to the variable var. And, the address of var is assigned to the pointVar pointer with the code pointVar = &var.

-1:27:12 1x

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers

Get the Value from the Address Using Pointers

To get the value pointed by a pointer, we use the * operator. For example:

```
int* pointVar, var;
var = 5;
// assign address of var to pointVar
pointVar = &var;
// access value pointed by pointVar
cout << *pointVar << endl;   // Output: 5
```

In the above code, the address of var is assigned to pointVar. We have used the *pointVar to get the value stored in that address.

When * is used with pointers, it's called the dereference operator. It operates on a pointer and gives the value pointed by the address stored in the pointer. That is, *pointVar = var.

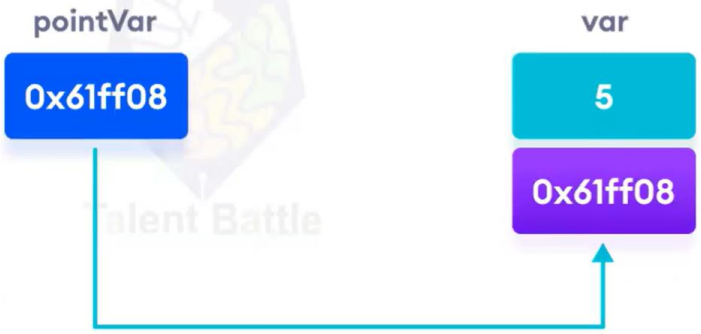
Note: In C++, pointVar and *pointVar is completely different. We cannot do something like *pointVar = &var;

-1:24:06 1x

```cpp
//=========================================
// c++ pointer

#include<iostream>
using namespace std;

int main(){
  int var = 5;

  // declare pointer variable
  int* pointVar;

  // store address of var
  pointVar = &var;

  // print value of var
  cout << "var = " << var << endl;

  // print address of var
  cout << "Address of var (&var) = " << &var << endl << endl;

  // print pointer pointVar
  cout << "pointVar = " << pointVar << endl;

  // print the content of the address pointVar points to
  cout << "Content of the address pointed to by pointVar (*pointVar) = "<< *pointVar
<< endl;
  return 0;
}


/*
var = 5
Address of var (&var) = 0x6ffe04

pointVar = 0x6ffe04
Content of the address pointed to by pointVar (*pointVar) = 5

--------------------------------
Process exited after 0.07561 seconds with return value 0
*/
```

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers



ENG 7:48 AM

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers

Changing Value Pointed by Pointers

If pointVar points to the address of var, we can change the value of var by using *pointVar.

For example,

```
int var = 5;
int* pointVar;
// assign address of var
pointVar = &var;
// change value at address pointVar
*pointVar = 1;
cout << var << endl; // Output: 1
```

Here, pointVar and &var have the same address, the value of var will also be changed when *pointVar is changed.

-1:14:41 1x

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers

C++ Pointers and Arrays

In C++, Pointers are variables that hold addresses of other variables. Not only can a pointer store the address of a single variable, it can also store the address of cells of an array.

Consider this example:

```
int *ptr;
int arr[5];
// store the address of the first
// element of arr in ptr
ptr = arr;
```

Here, ptr is a pointer variable while arr is an int array. The code ptr = arr; stores the address of the first element of the array in variable ptr.

Notice that we have used arr instead of &arr[0]. This is because both are the same. So, the code below is the same as the code above.

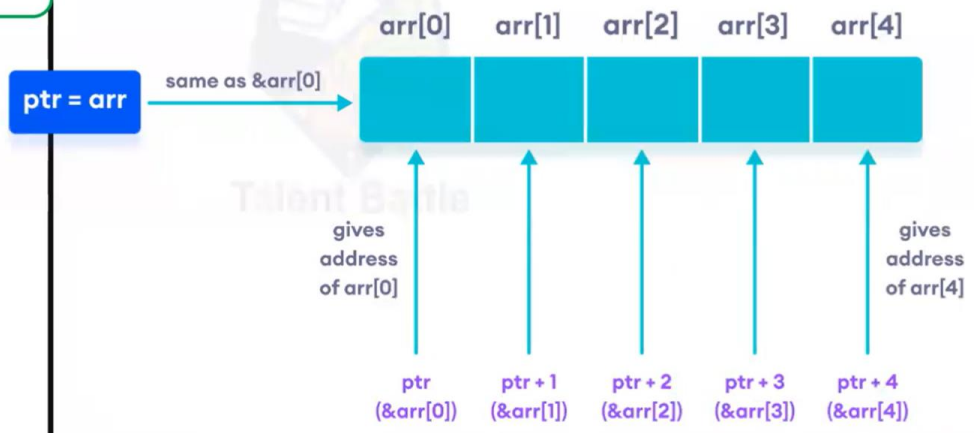
```
int *ptr;
int arr[5];
ptr = &arr[0];
```

The addresses for the rest of the array elements are given by &arr[1], &arr[2], &arr[3], and &arr[4].

-1:03:53 1x

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers



ENG 8:09 AM

C++ Pointers

Point to Every Array Elements

Suppose we need to point to the fourth element of the array using the same pointer ptr.

Here, if ptr points to the first element in the above example then ptr + 3 will point to the fourth element. For example,

```
int *ptr;
int arr[5];
ptr = arr;
ptr + 1 is equivalent to &arr[1];
ptr + 2 is equivalent to &arr[2];
ptr + 3 is equivalent to &arr[3];
ptr + 4 is equivalent to &arr[4];
```

Similarly, we can access the elements using the single pointer. For example,

```
// use dereference operator
*ptr == arr[0];
*(ptr + 1) is equivalent to arr[1];
*(ptr + 2) is equivalent to arr[2];
*(ptr + 3) is equivalent to arr[3];
*(ptr + 4) is equivalent to arr[4];
```

Suppose if we have initialized ptr = &arr[2]; then

```
ptr - 2 is equivalent to &arr[0];
ptr - 1 is equivalent to &arr[1];
ptr + 1 is equivalent to &arr[3];
ptr + 2 is equivalent to &arr[4];
```


C++ Pointers

C++ Call by Reference: Using pointers [With Examples]

```
// function that takes value as parameter
void func1(int numVal) {
   // code
}
// function that takes reference as parameter
// notice the & before the parameter
void func2(int &numRef) {
   // code
}
int main() {
   int num = 5;
   // pass by value
   func1(num);
   // pass by reference
   func2(num);
   return 0;
}
```

numVal = ? 5

numRef = ? 1000

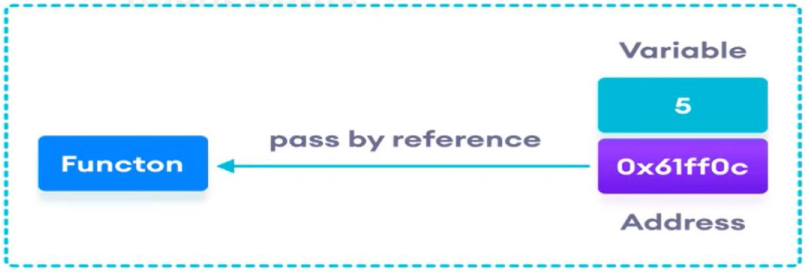
num

5

1000

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers



38:48 1x

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

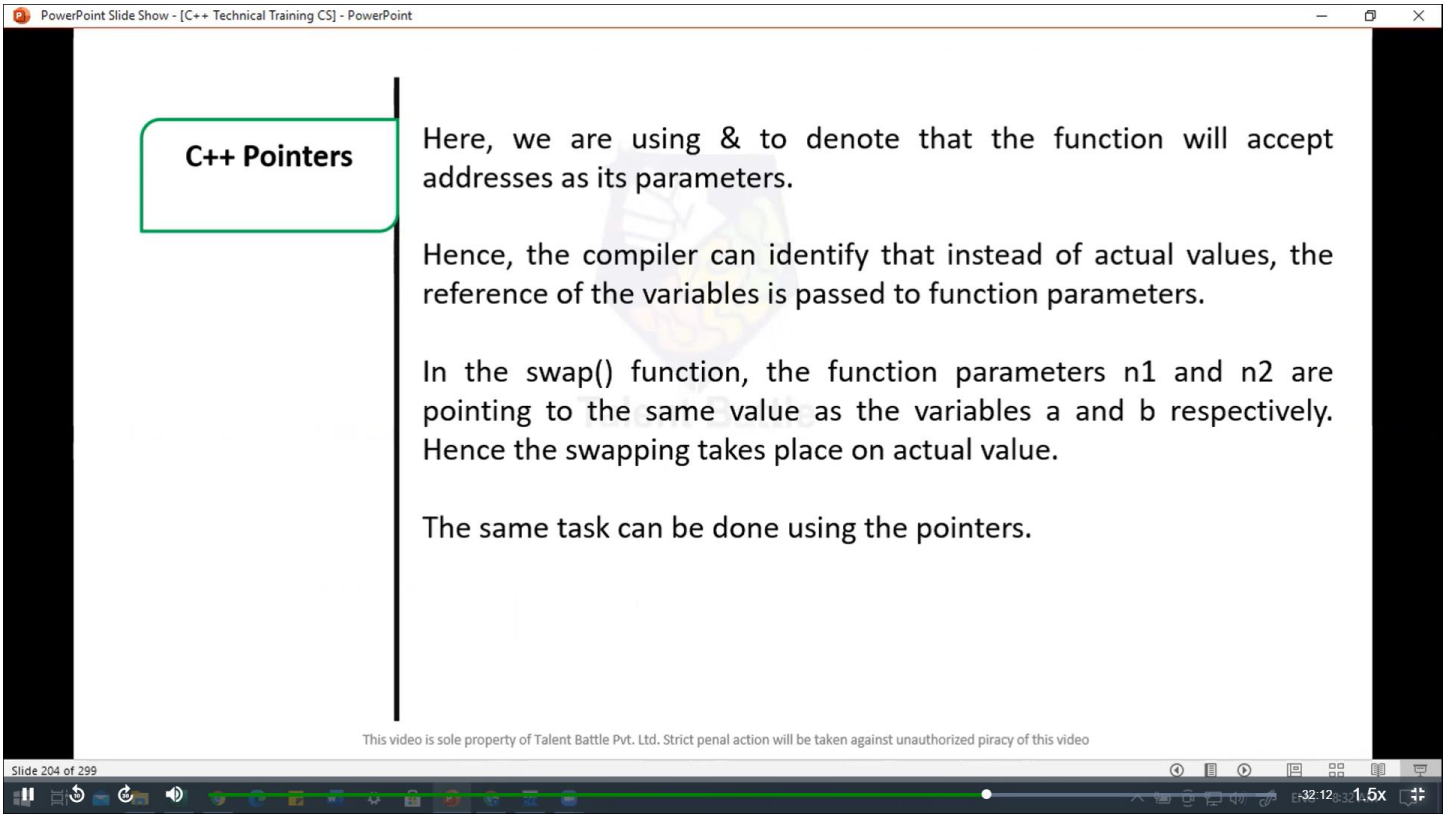
C++ Pointers

Passing by reference without pointers

```
#include <iostream>
using namespace std;

// function definition to swap values
void swap(int &n1, int &n2) {
   int temp;
   temp = n1;
   n1 = n2;
   n2 = temp;
}

int main()
{

   // initialize variables
   int a = 1, b = 2;

   cout << "Before swapping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;

   // call function to swap numbers
   swap(a, b);

   cout << "\nAfter swapping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;

   return 0;
}
```

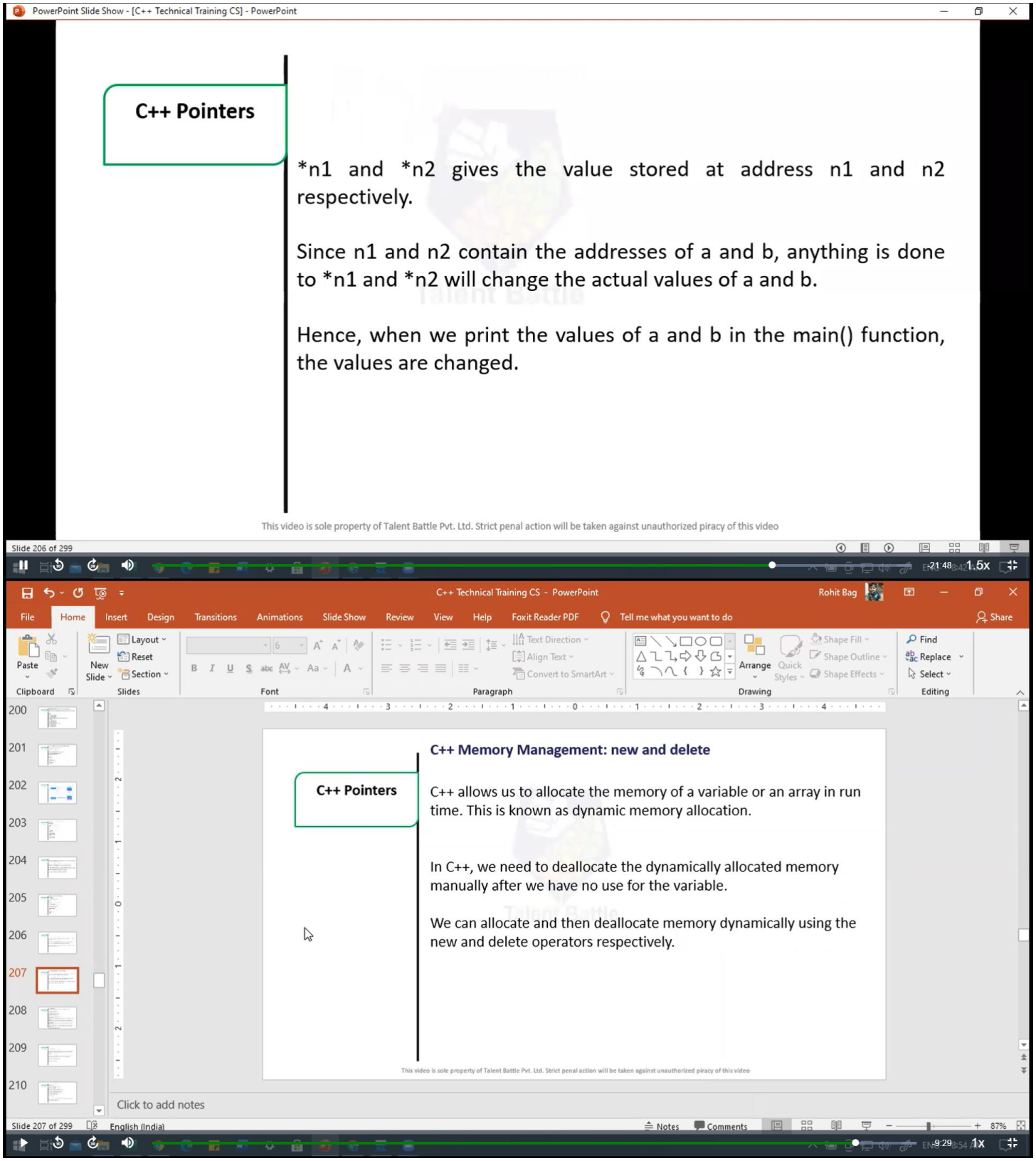

37:37 1x

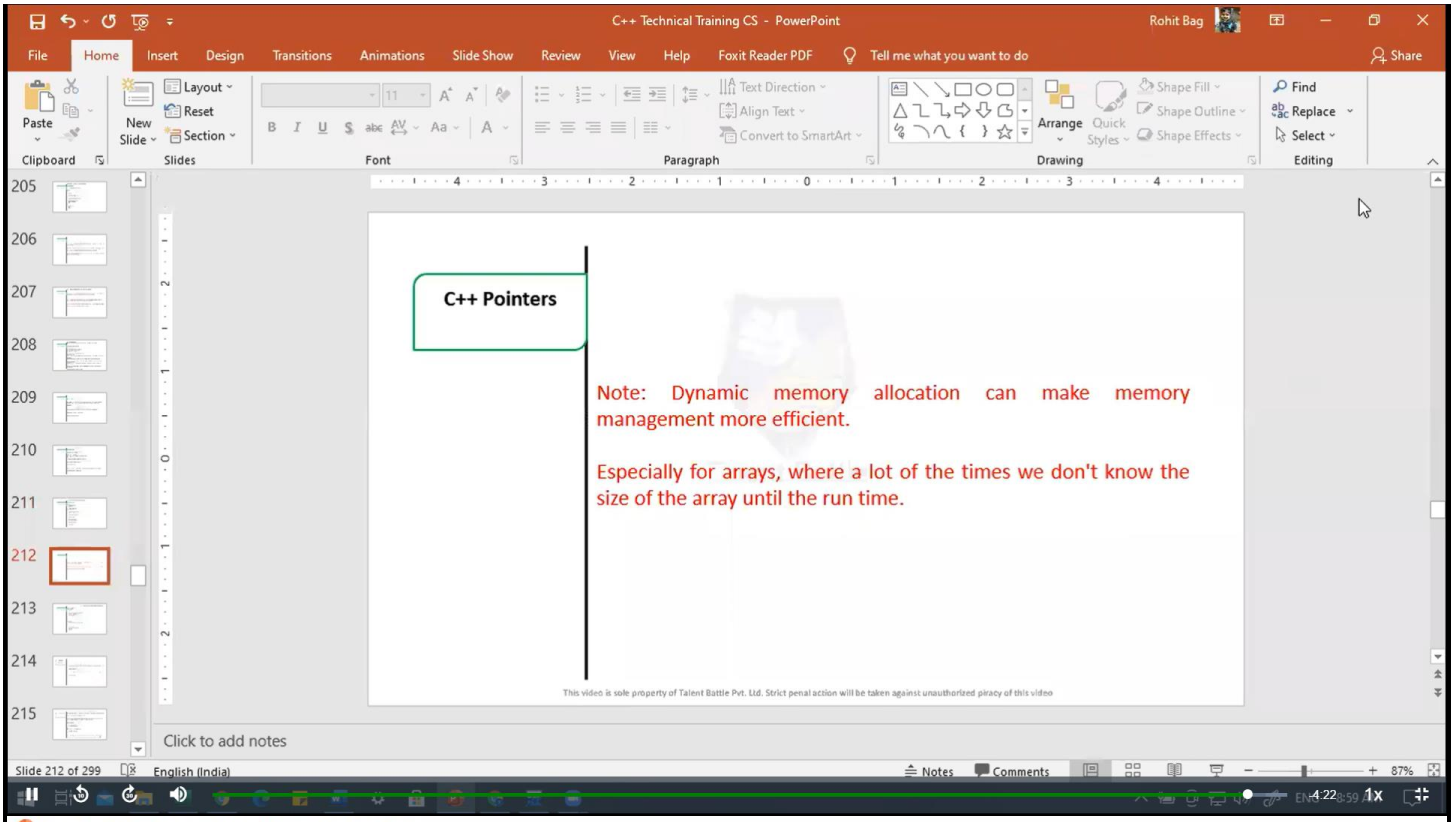
```cpp
//================================================
// Passing by reference without pointers

#include<iostream>
using  namespace std;

// function definition to swap values
void swap(int &n1, int &n2){
  int temp;
  temp = n1;
  n1 = n2;
  n2 = temp;
}

int main(){
  // initialize variables
  int a = 1, b = 2;

  cout << "Before swapping" << endl;
  cout << "a = " << a << endl;
  cout << "b = " << b << endl;

  // call function to swap numbers
  swap(a, b);

  cout << "\nAfter swapping" << endl;
  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
  return 0;
}



/*
Before swapping
a = 1
b = 2

After swapping
a = 2
b = 1

--------------------------------
Process exited after 0.117 seconds with return value 0
*/
```

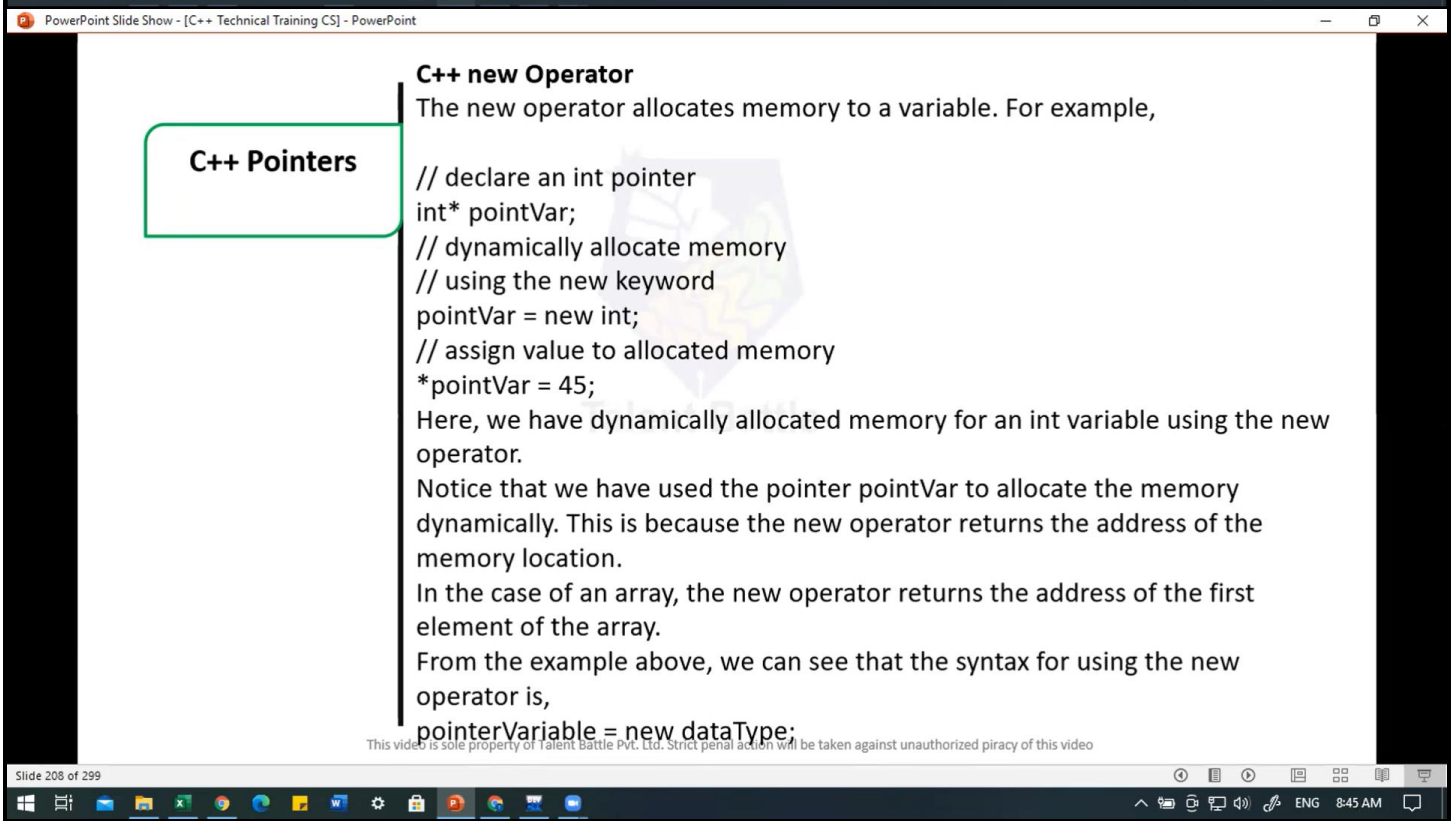



```cpp
//=================================================
// Passing by reference using pointers

#include <iostream>
using namespace std;

// function prototype with pointer as parameters
void swap(int*, int*);

int main(){
  // initialize variables
  int a = 1, b = 2;

  cout << "Befor swapping" << endl;
  cout << "a = " << a << endl;
  cout << "b = " << b << endl;

  // call function by passing variable addresses
  swap(&a, &b);

  cout << "\nAfter swapping" << endl;
  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
  return 0;
}

// function definition to swap numbers
```

```c
void swap(int* n1, int* n2){
  int temp;
  temp = *n1;
  *n1 = *n2;
  *n2 = temp;
}



/*
Befor swapping
a = 1
b = 2

After swapping
a = 2
b = 1

--------------------------------
Process exited after 0.1636 seconds with return value 0
*/
```


PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint
C++ Pointers
*n1 and *n2 gives the value stored at address n1 and n2 respectively.
Since n1 and n2 contain the addresses of a and b, anything is done to *n1 and *n2 will change the actual values of a and b.
Hence, when we print the values of a and b in the main() function, the values are changed.
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Slide 206 of 299
1.5x
C++ Technical Training CS - PowerPoint
Rohit Bag
File Home Insert Design Transitions Animations Slide Show Review View Help Foxit Reader PDF Tell me what you want to do
Share
Paste
New Slide
Layout
Reset
Section
Clipboard
Slides
Font
Text Direction
Align Text
Convert to SmartArt
Paragraph
Arrange
Quick Styles
Shape Fill
Shape Outline
Shape Effects
Drawing
Find
Replace
Select
Editing
C++ Memory Management: new and delete
C++ Pointers
C++ allows us to allocate the memory of a variable or an array in run time. This is known as dynamic memory allocation.
In C++, we need to deallocate the dynamically allocated memory manually after we have no use for the variable.
We can allocate and then deallocate memory dynamically using the new and delete operators respectively.
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Click to add notes
Slide 207 of 299
English (India)
Notes
Comments
87%
1x


C++ Technical Training CS - PowerPoint
Rohit Bag
C++ Pointers
Note: Dynamic memory allocation can make memory management more efficient.
Especially for arrays, where a lot of the times we don't know the size of the array until the run time.
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Click to add notes
Slide 212 of 299
English (India)


PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint
C++ Pointers
C++ new Operator
The new operator allocates memory to a variable. For example,
// declare an int pointer
int* pointVar;
// dynamically allocate memory
// using the new keyword
pointVar = new int;
// assign value to allocated memory
*pointVar = 45;
Here, we have dynamically allocated memory for an int variable using the new operator.
Notice that we have used the pointer pointVar to allocate the memory dynamically. This is because the new operator returns the address of the memory location.
In the case of an array, the new operator returns the address of the first element of the array.
From the example above, we can see that the syntax for using the new operator is,
pointerVariable = new dataType;
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Slide 208 of 299

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers

delete Operator

Once we no longer need to use a variable that we have declared dynamically, we can deallocate the memory occupied by the variable.

For this, the delete operator is used. It returns the memory to the operating system. This is known as memory deallocation.

The syntax for this operator is

```
delete pointerVariable;
```


ENG 8:48 AM

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Pointers

```
// declare an int pointer
int* pointVar;
// dynamically allocate memory
// for an int variable
pointVar = new int;
// assign value to the variable memory
*pointVar = 45;
// print the value stored in memory
cout << *pointVar; // Output: 45
// deallocate the memory
delete pointVar;
```

Here, we have dynamically allocated memory for an int variable using the pointer pointVar.

13:39 1x

```
//=================================================
// C++ Dynamic memory allocation


#include<iostream>
using namespace std;
```

```cpp
int main(){
  // declare an int pointer
  int* pointInt;

  // declare a float pointer
  float* pointFloat;

  // dynamically allocate memory
  pointInt = new int;
  pointFloat = new float;

  // assigning value to the  memory
  *pointInt = 45;
  *pointFloat = 45.45f;

  cout << *pointInt << endl;
  cout << *pointFloat << endl;

  // deallocate the memory
  delete pointInt;
  delete pointFloat;

  return 0;
}



/*
45
45.45

--------------------------------
Process exited after 0.1411 seconds with return value 0

*/
```