OOP
Lecture 3

12/10/2021

Command Line Arguments in C++ :

The most important and compulsory function of any C/C++ program is the main() function. It is mostly defined with a return type of int and without parameters :

```
int main( ){
   - - - -
}
```

However, Arguments/parameters can be passed to the main() function. If any input value is passed through command prompt at the time of running of program is known as **command line argument**. It is a concept of passing the arguments to the main() function by using command prompt.

In Command line arguments application main() function will take two arguments. There are standard parameters of main function,

- int argc
- char *argc[ ]

argc:

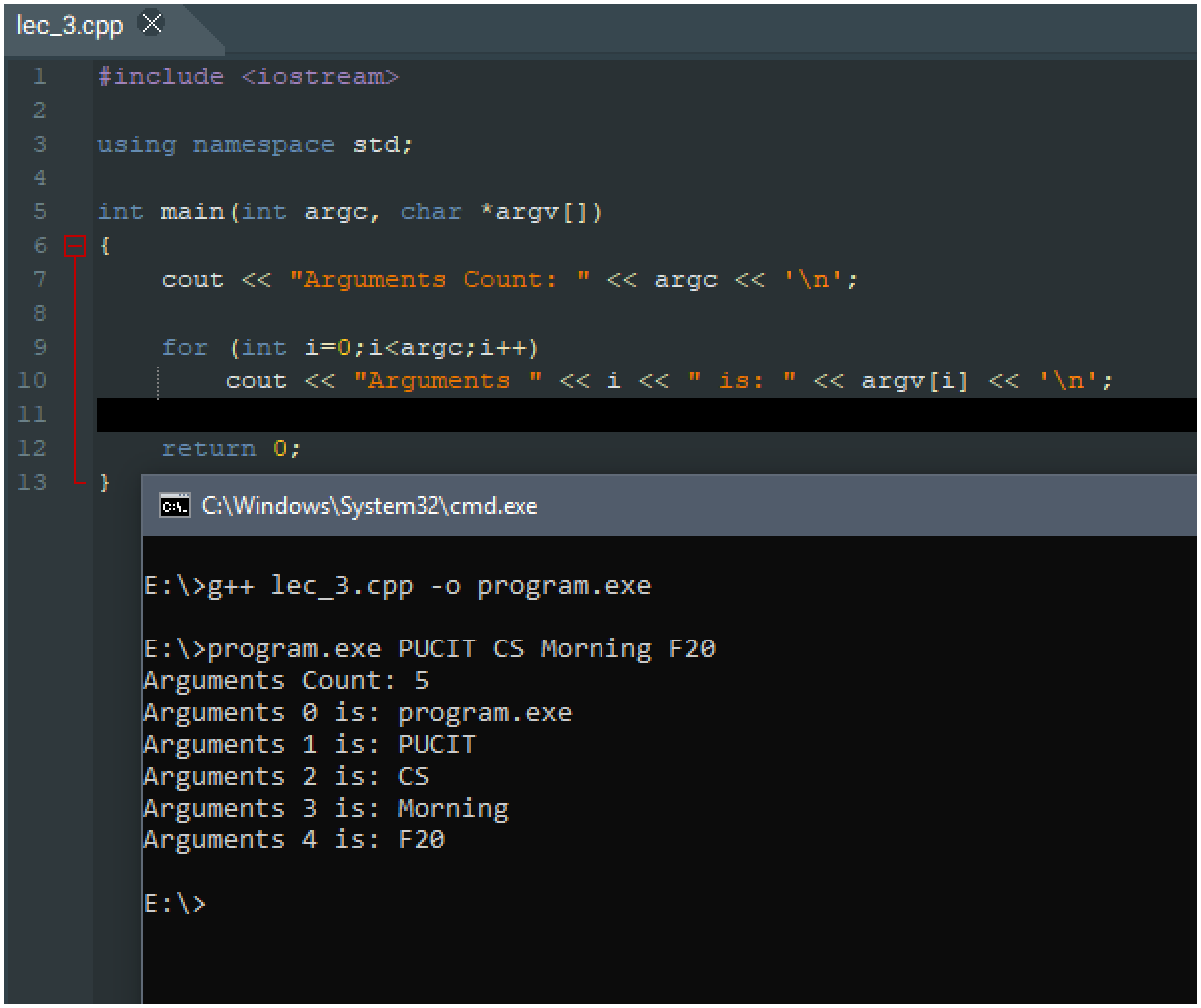
argc is an integer type variable and it holds the total number of arguments which is passed into the main() function. It take Number of arguments in the command line including the program name.

argv[ ]:

argv[ ] is a char* type variable, which holds actual arguments which are passed to the main() function. argv is a 2D character array or 1D string array. argv[0] is the name of the program.

Command line arguments are not compiled and run like normal C++ programs, these programs are compiled and run on command prompt. Arguments in these types of programs can only be passed from the command prompt. Command line arguments related programs are not executed directly from IDE (like Dev c++ or Visual Studio) because arguments can not be passed.

Sample Programs :



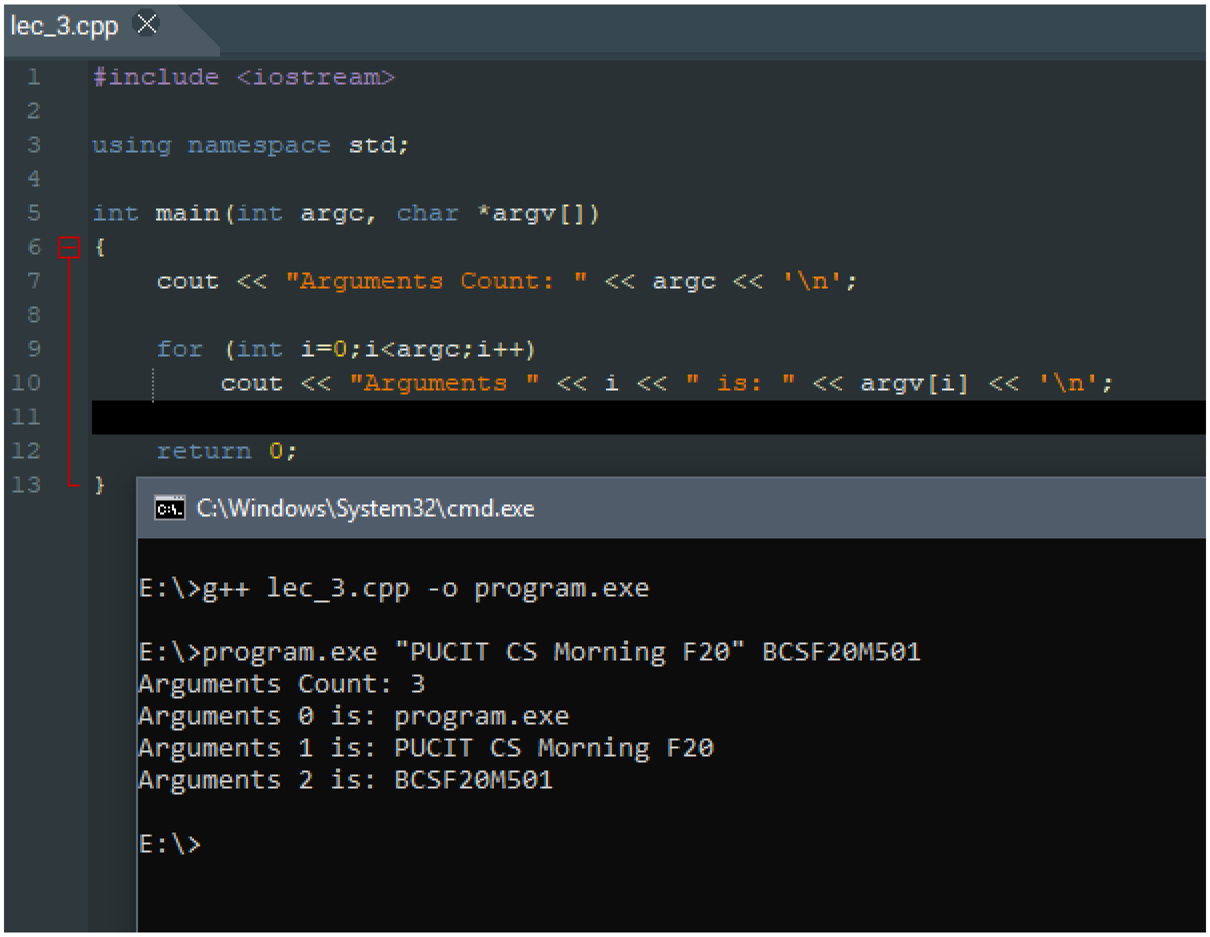
```cpp
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    cout << "Arguments Count: " << argc << '\n';

    for (int i=0;i<argc;i++)
        cout << "Arguments " << i << " is: " << argv[i] << '\n';

    return 0;
}
```

```
E:\>g++ lec_3.cpp -o program.exe

E:\>program.exe PUCIT CS Morning F20
Arguments Count: 5
Arguments 0 is: program.exe
Arguments 1 is: PUCIT
Arguments 2 is: CS
Arguments 3 is: Morning
Arguments 4 is: F20

E:\>
```

Here, We just wrote the program in any editor. Then opened the command prompt with Desktop directory. The First command i.e **“g++ lec_3.cpp -o program.exe”** is used to compile the program. Any names can be given here. The second command i.e **“program.exe PUCIT CS Morning F20”** is used to run the program with arguments. The first argument is the name of the program and then any number of arguments can be passed to the program separated by spaces.

Now, the question arises what if we have to pass multiple words separated by space as a single argument. For this Purpose we will enclose our argument in double quotes.

Example:


lec_3.cpp

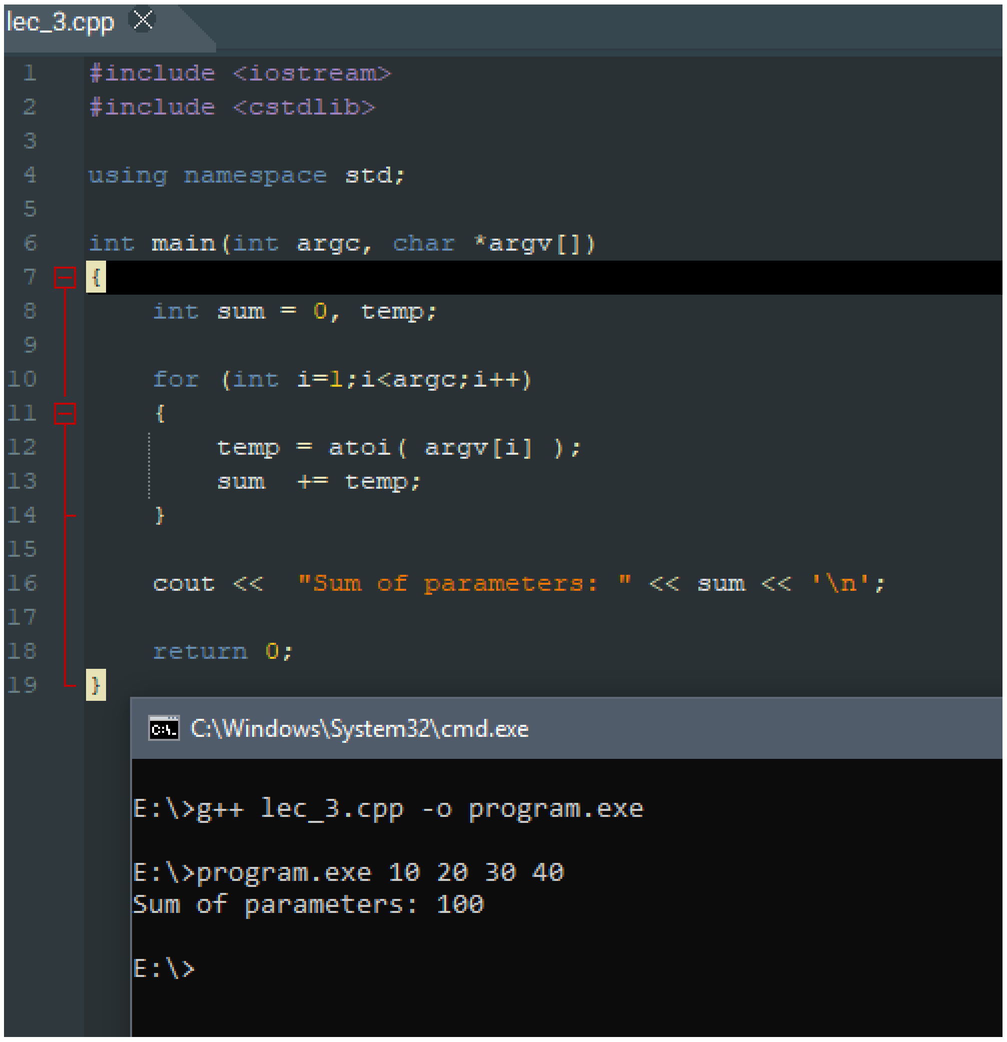
```cpp
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    cout << "Arguments Count: " << argc << '\n';

    for (int i=0;i<argc;i++)
        cout << "Arguments " << i << " is: " << argv[i] << '\n';

    return 0;
}
```

C:\Windows\System32\cmd.exe

```
E:\>g++ lec_3.cpp -o program.exe

E:\>program.exe "PUCIT CS Morning F20" BCSF20M501
Arguments Count: 3
Arguments 0 is: program.exe
Arguments 1 is: PUCIT CS Morning F20
Arguments 2 is: BCSF20M501

E:\>
```

Sample Program 2:

The built-in function used in this program is atoi(); It is defined in the <cstdlib> library. atoi stands for **“alphanumeric to integer”**. As it is clear from its name that it converts the digits in a string to an integer. More explicitly, It takes a string like **“1234”** and returns an integer number with value **1234**.


lec_3.cpp

```cpp
#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char *argv[])
{
    int sum = 0, temp;

    for (int i=1;i<argc;i++)
    {
        temp = atoi( argv[i] );
        sum  += temp;
    }

    cout <<  "Sum of parameters: " << sum << '\n';

    return 0;
}
```

C:\Windows\System32\cmd.exe

```
E:\>g++ lec_3.cpp -o program.exe

E:\>program.exe 10 20 30 40
Sum of parameters: 100

E:\>
```

Difference Between char type array and string:

Char type array is just a combination of sequential characters of data type char. Like:
char ch[]={'A','B','V','Z'};

Whereas, string is a character array, where collection of characters is terminated by null character. It refers to a sequence of characters, which is represented as a single data type. A string can be defined in multiple ways, these are:

char ch[ ] = {'A', 'S', 'I', 'M', '\0'};
char ch[ ] = {'A', 'S', 'I', 'M', 0};
char ch[ ]= "ASIM";

In C, the strings are basically an array of characters. In C++ the std :: string is an advancement of that array. There are some additional features with

the traditional character array(We will discuss more details about it in classes). The null terminated strings are basically a sequence of characters, and the last element is one null character (denoted by ‘\0’). When we write some string using double quotes (“…”), then it is converted into null terminated strings by the compiler.

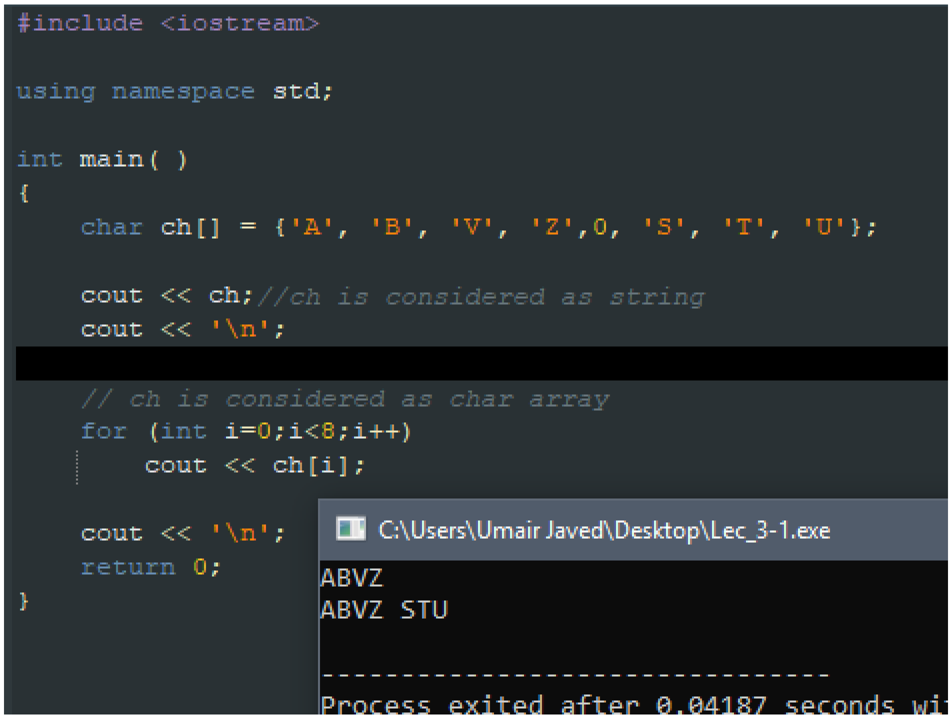
The size of the string may be smaller than the array size, but if there are some null characters inside that array, that will be treated as the end of that string.

NULL Character and Zero Character :

Null Character :	'\0' has ASCII value 0
Zero Character :	'0' has ascii value 48
Characters :	‘0’ to ‘9’ have ASCII values 48 to 57.

Sample Programs :

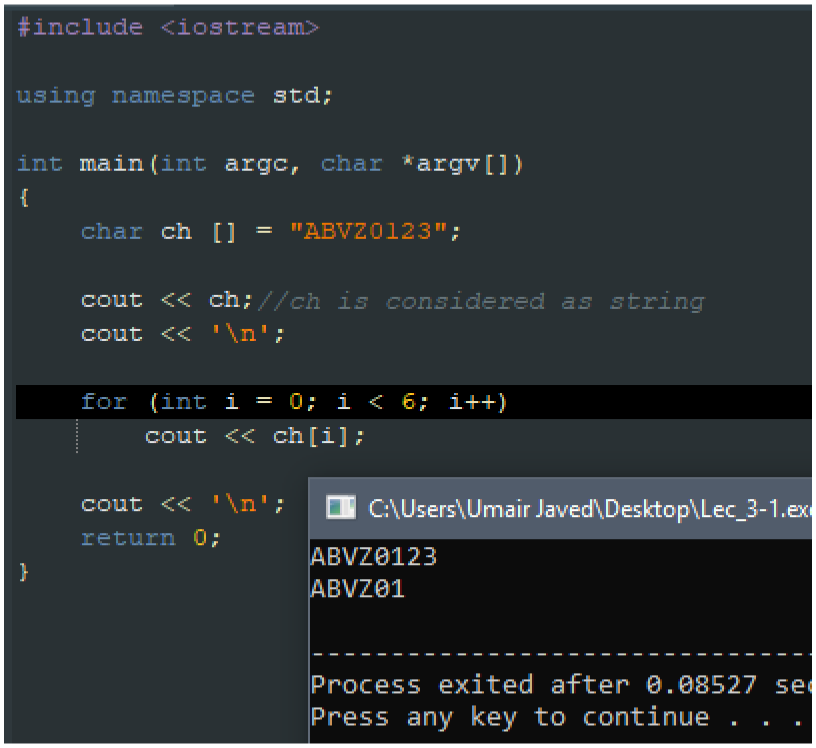
These are some sample programs of character type arrays and strings. Get familiar with the working of both of them before moving further.



```cpp
#include <iostream>

using namespace std;

int main( )
{
    char ch[] = {'A', 'B', 'V', 'Z',0, 'S', 'T', 'U'};

    cout << ch;//ch is considered as string
    cout << '\n';

    // ch is considered as char array
    for (int i=0;i<8;i++)
        cout << ch[i];

    cout << '\n';
    return 0;
}
```

C:\Users\Umair Javed\Desktop\Lec_3-1.exe

```
ABVZ
ABVZ STU

--------------------------------
Process exited after 0.04187 seconds wi
```

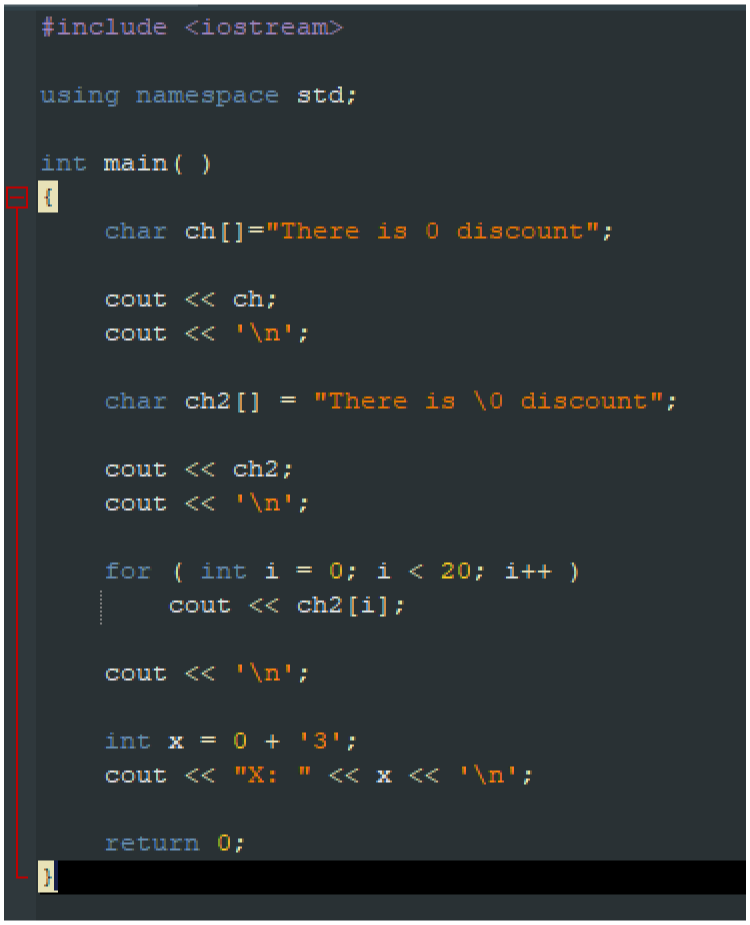


```cpp
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    char ch [] = "ABVZ0123";

    cout << ch;//ch is considered as string
    cout << '\n';

    for (int i = 0; i < 6; i++)
        cout << ch[i];

    cout << '\n';
    return 0;
}
```

C:\Users\Umair Javed\Desktop\Lec_3-1.ex

```
ABVZ0123
ABVZ01

--------------------------------
Process exited after 0.08527 se
Press any key to continue . . .
```

Quick Quiz : Guess the Output



```
#include <iostream>

using namespace std;

int main( )
{
    char ch[]="There is 0 discount";

    cout << ch;
    cout << '\n';

    char ch2[] = "There is \0 discount";

    cout << ch2;
    cout << '\n';

    for ( int i = 0; i < 20; i++ )
        cout << ch2[i];

    cout << '\n';

    int x = 0 + '3';
    cout << "X: " << x << '\n';

    return 0;
}
```

Where and When we Use char Data Type :

- We use char data type to store characters.
- we use char type array to store multiple characters.
- we use char type array to store string i.e. collection of characters terminated by null character.
- We have 8 bits in char data type, which can store a number from 0 to 255
  or in case of signed numbers, we can store -128 to +127 in 1 byte

Unsigned Characters :

In unsigned char all 8 bits are used to represent number so
00000000 is 0
11111111 is 255
Syntax: unsigned char char_name;

Signed Characters :

In signed char 7 bits are used to represent number and 1 bit is used for sign

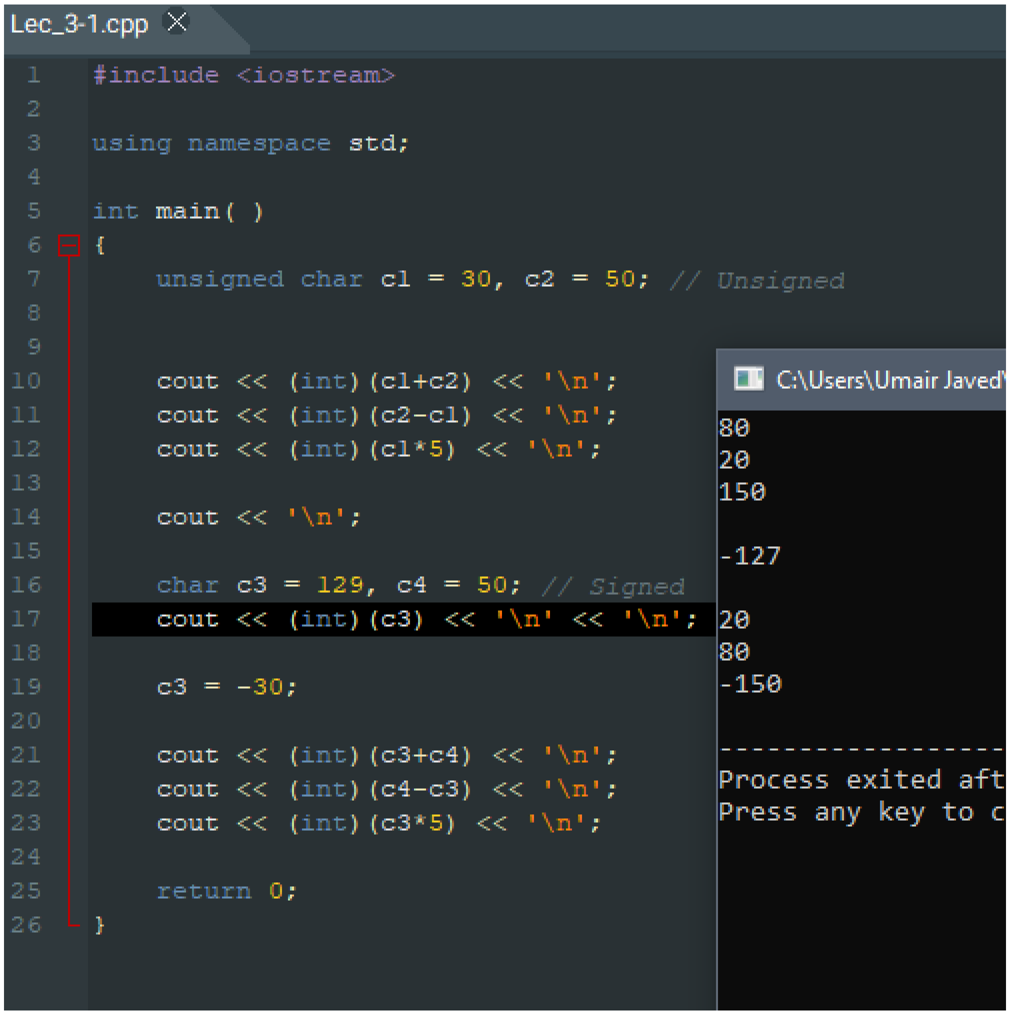
Sign	Number		
0	0000000 = 0	positive	
0	1111111 = 127	positive	
1	0000000 = -128	negative	(In 2's complement)
1	1111111 = -1	negative	(In 2's complement)

By default, all characters declared with char data type are signed.

Usage :

If we have numbers in the range of char,(i.e from -128 to 127 or 0 to 255) we may store them in char data type to save space.

Sample Program :



Lec_3-1.cpp

```cpp
#include <iostream>

using namespace std;

int main( )
{
    unsigned char c1 = 30, c2 = 50; // Unsigned


    cout << (int)(c1+c2) << '\n';
    cout << (int)(c2-c1) << '\n';
    cout << (int)(c1*5) << '\n';

    cout << '\n';

    char c3 = 129, c4 = 50; // Signed
    cout << (int)(c3) << '\n' << '\n';

    c3 = -30;

    cout << (int)(c3+c4) << '\n';
    cout << (int)(c4-c3) << '\n';
    cout << (int)(c3*5) << '\n';

    return 0;
}
```

C:\Users\Umair Javed\

```
80
20
150

-127

20
80
-150

------------------
Process exited aft
Press any key to c
```

Usage of Character type to store numbers :

If we have numbers in the range of char, we may store them in char data type to save space. As the char type takes 1 byte and int type takes 4 bytes in the memory. The most common example is how an image is saved?

An image is a set of pixels, a pixel is the smallest unit composing our image. An image is represented with a 2-dimensional array. Each cell of the array is composed of the intensity of the pixel at that particular location. The size of the image matrix totally depends on the size of the image.
For example, if the size of the image is (400 x 200) pixels (400 pixels wide and 200 pixels high) it means that the matrix of this image has 400 rows and 200 columns. Each cell of the matrix contains a value from 0 to 255 corresponding to the intensity of the color at that point (pixel).

Why not more than 255?

Because 0 to 255 values can be stored in one byte (8 bits) and because human eyes cannot really see more details with bigger values (255 is enough for most people). Therefore 256 values for each color are working in addition to being convenient.

Image has specific type, width, height, no of colors used, compression type etc, which is called metadata of image, first we store metadata of image then we store data of images.

There are 3 types of images:

- We may have a pure black & white image, where 1 bit is used for each pixel; each pixel is either black (0 bit) or white (1 bit).

- We may have a grayscale image that has shades of gray from 0 to 255, where 0 is the darkest shade (black color) and 255 is the lightest shade (white color).

- In general, we have a colored image normally called a 24-bit image, where we have 8 bits for red color, 8 bits for green color and 8 bits for blue color. All the colors are formed by combination of RGB. Where red has 256 shades, blue has 256 shades and green has 256 shades.

Why we use char type instead of int to store images :

Consider an Image (200x 300) with width 200 pixels and height 300 pixels. It specifies that we have 200 x 300 = 60,000 pixel / entries / values in its matrix. And if it is a colored image than we have three matrices of size 200 x 300 which corresponds to 60,000 * 3 = 180,000 values. Now, if we use integer datatype to store the values of image and we know that int takes 4 bytes in memory so the the size of image will be 180,000 x 4 = 720,000 bytes. But, as we know, the range of color entries in the image matrix will always be from 0 to 255. So, isn't it better to use the unsigned char to store these values? As the char type takes only 1 byte so the size of the image will be 180,000 x 1 = 180,000 bytes. We can see a huge difference between the size of int and size of char so that is the reason why we use char data type to store the values in range -127 to 128 or 0 to 255.

Lecture 3

THE END

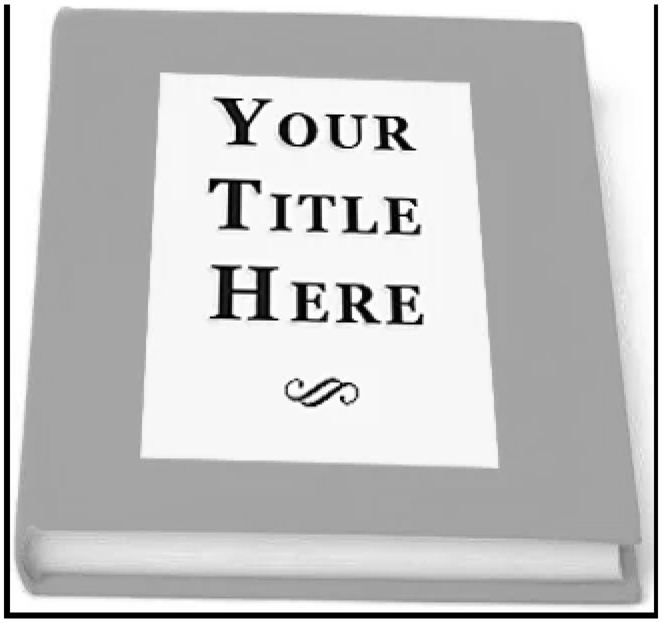
OOP

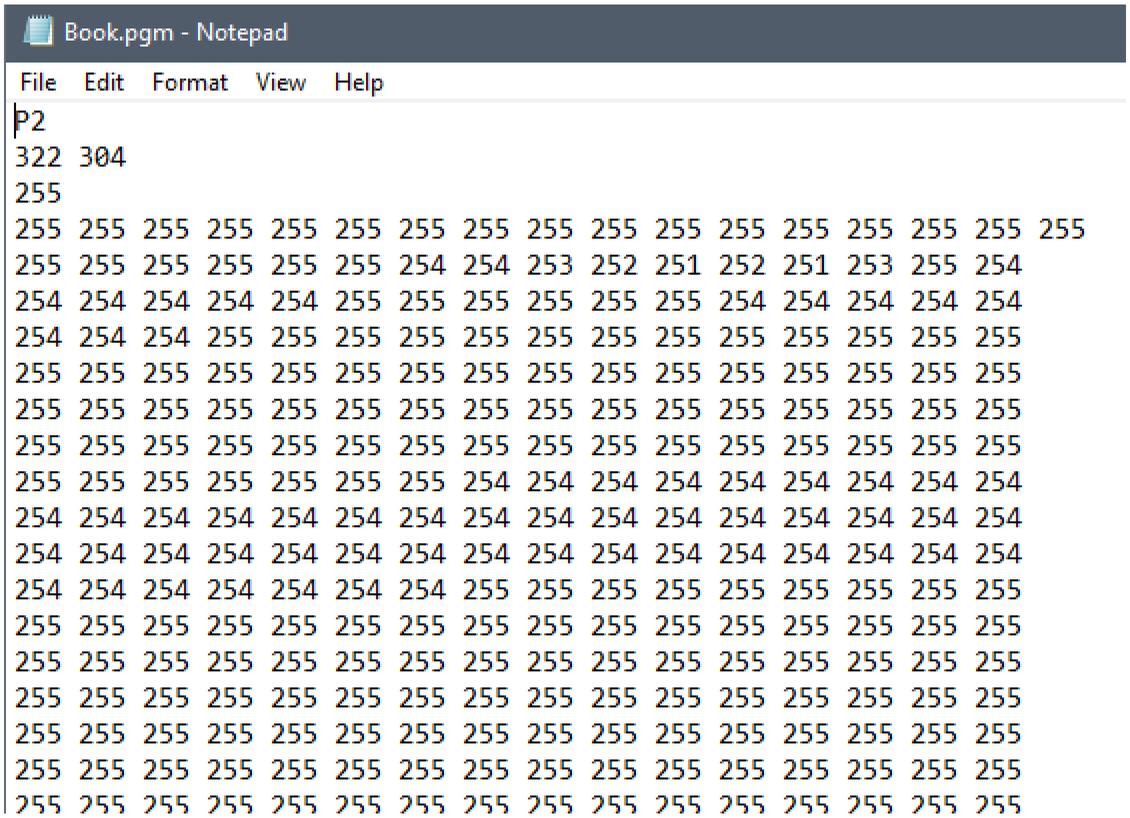
Lecture 4

14/10/2021

PGM :

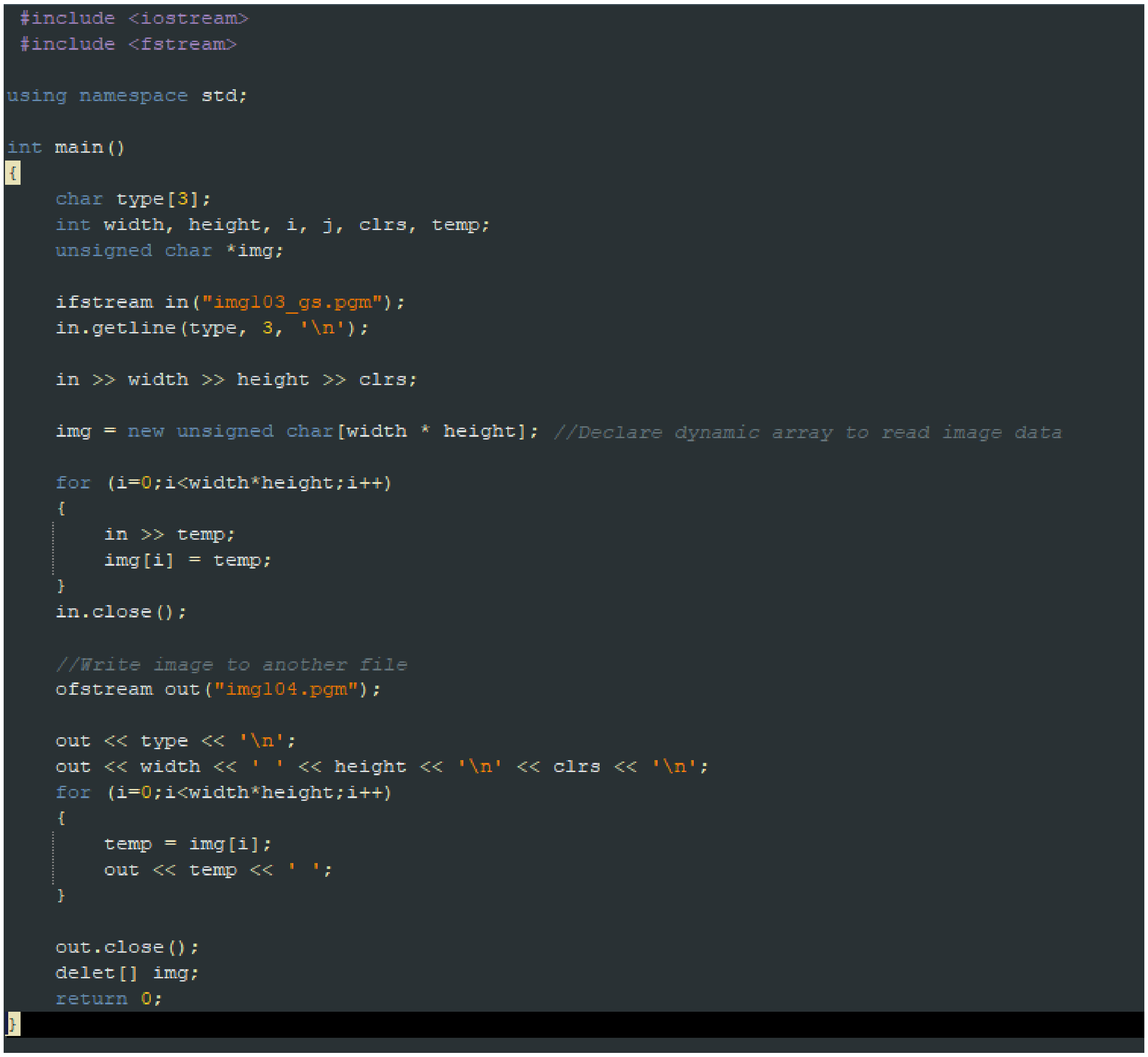
The PGM format is one of several image formats defined by the Netpbm project, which is an open source package of graphics programs. A PGM file is a grayscale image file saved in the portable gray map (PGM) format and encoded with one byte (8 bits) per pixel. It contains header information and a grid of numbers that represent different shades of gray from black (0) to white (255). PGM files are typically stored inASCII text format. PGM files include a header that defines the PGM format type ("P2" for text or "P5" for binary), image width and height, and the maximum number of shades. Here , we are using “IrfanView” software to open .pgm files.





```
P2
322 304
255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 254 254 253 252 251 252 251 253 255 254
254 254 254 254 254 255 255 255 255 255 255 254 254 254 254 254
254 254 254 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 254 254 254 254 254 254 254 254 254
254 254 254 254 254 254 254 254 254 254 254 254 254 254 254 254
254 254 254 254 254 254 254 254 254 254 254 254 254 254 254 254
254 254 254 254 254 254 254 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

Reading the Contents/Values of Image into a dynamic array and writing it another file as it is (without any change).

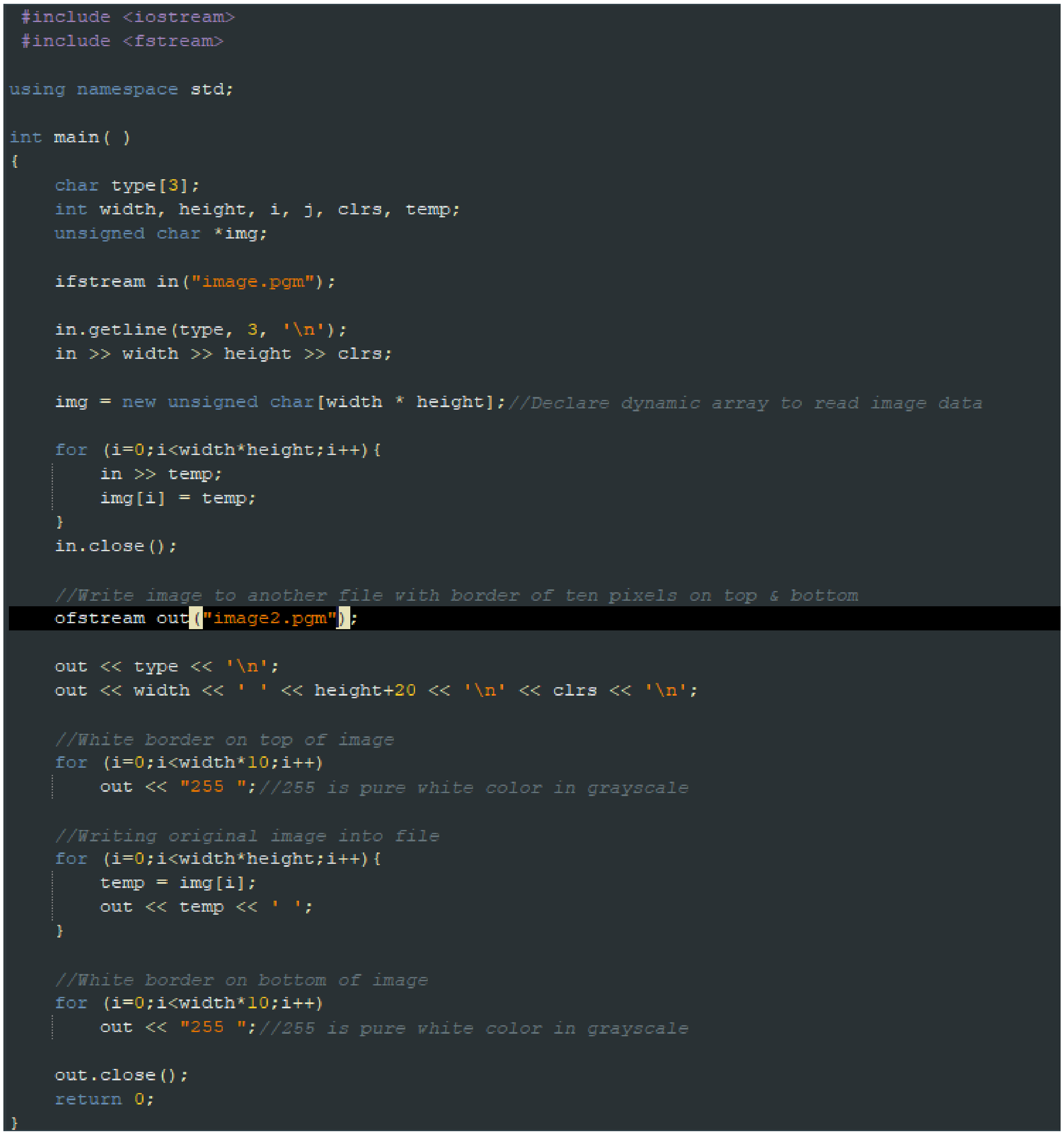


```
 #include <iostream>
 #include <fstream>

using namespace std;

int main()
{
    char type[3];
    int width, height, i, j, clrs, temp;
    unsigned char *img;

    ifstream in("img103_gs.pgm");
    in.getline(type, 3, '\n');

    in >> width >> height >> clrs;

    img = new unsigned char[width * height]; //Declare dynamic array to read image data

    for (i=0;i<width*height;i++)
    {
        in >> temp;
        img[i] = temp;
    }
    in.close();

    //Write image to another file
    ofstream out("img104.pgm");

    out << type << '\n';
    out << width << ' ' << height << '\n' << clrs << '\n';
    for (i=0;i<width*height;i++)
    {
        temp = img[i];
        out << temp << ' ';
    }

    out.close();
    delet[] img;
    return 0;
}
```

Adding Border of 10 pixel :

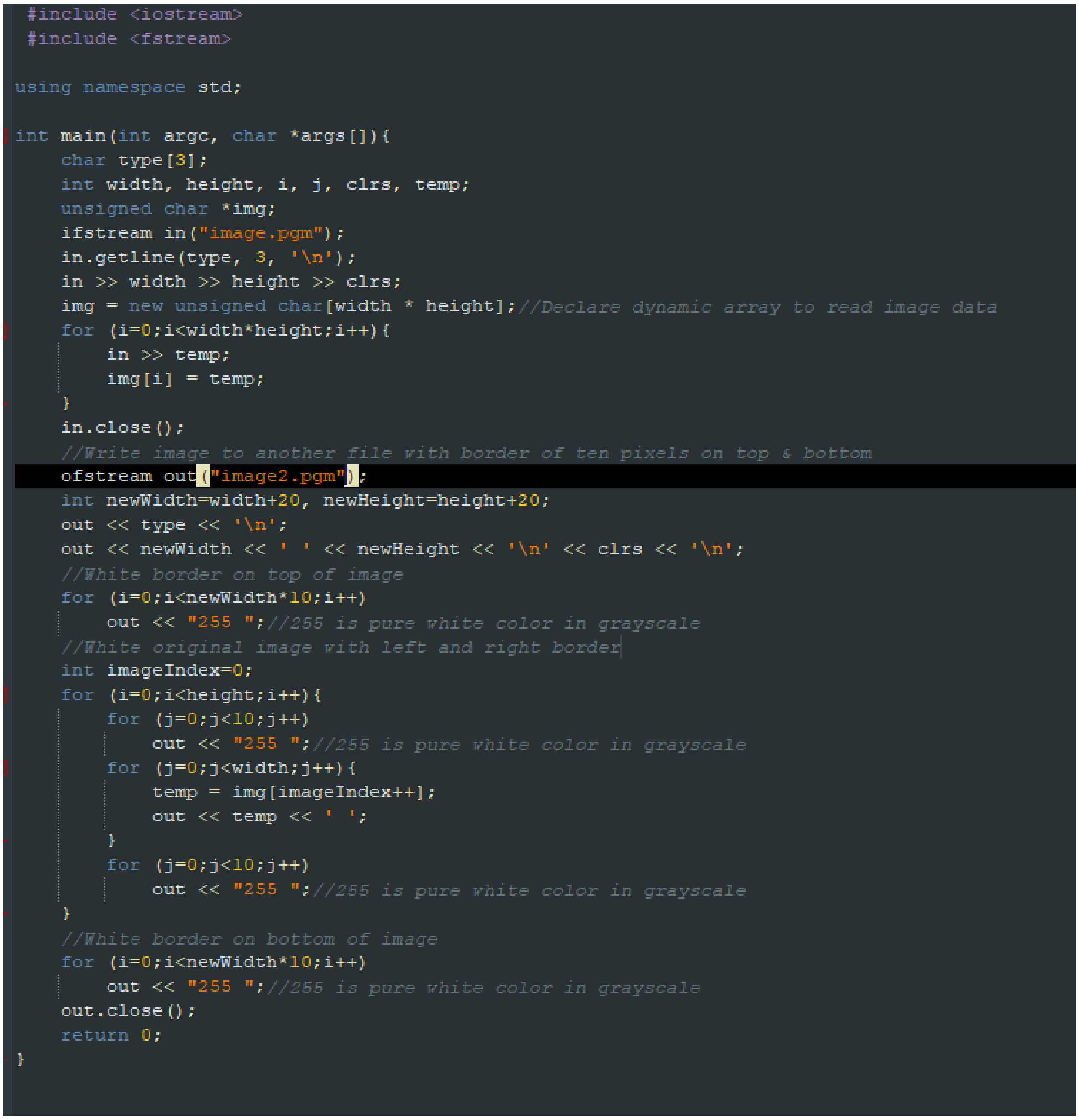
To add the border of 10 pixels on top and bottom of the image we need to add a loop to store values 255 according to the width of the image.

C++ Code to do this is:



```
 #include <iostream>
 #include <fstream>

using namespace std;

int main( )
{
    char type[3];
    int width, height, i, j, clrs, temp;
    unsigned char *img;

    ifstream in("image.pgm");

    in.getline(type, 3, '\n');
    in >> width >> height >> clrs;

    img = new unsigned char[width * height];//Declare dynamic array to read image data

    for (i=0;i<width*height;i++){
        in >> temp;
        img[i] = temp;
    }
    in.close();

    //Write image to another file with border of ten pixels on top & bottom
    ofstream out("image2.pgm");

    out << type << '\n';
    out << width << ' ' << height+20 << '\n' << clrs << '\n';

    //White border on top of image
    for (i=0;i<width*10;i++)
        out << "255 ";//255 is pure white color in grayscale

    //Writing original image into file
    for (i=0;i<width*height;i++){
        temp = img[i];
        out << temp << ' ';
    }

    //White border on bottom of image
    for (i=0;i<width*10;i++)
        out << "255 ";//255 is pure white color in grayscale

    out.close();
    return 0;
}
```

Now adding Borders to the left and right of the image:



```
 #include <iostream>
 #include <fstream>

using namespace std;

int main(int argc, char *args[]){
    char type[3];
    int width, height, i, j, clrs, temp;
    unsigned char *img;
    ifstream in("image.pgm");
    in.getline(type, 3, '\n');
    in >> width >> height >> clrs;
    img = new unsigned char[width * height];//Declare dynamic array to read image data
    for (i=0;i<width*height;i++){
        in >> temp;
        img[i] = temp;
    }
    in.close();
    //Write image to another file with border of ten pixels on top & bottom
    ofstream out("image2.pgm");
    int newWidth=width+20, newHeight=height+20;
    out << type << '\n';
    out << newWidth << ' ' << newHeight << '\n' << clrs << '\n';
    //White border on top of image
    for (i=0;i<newWidth*10;i++)
        out << "255 ";//255 is pure white color in grayscale
    //White original image with left and right border
    int imageIndex=0;
    for (i=0;i<height;i++){
        for (j=0;j<10;j++)
            out << "255 ";//255 is pure white color in grayscale
        for (j=0;j<width;j++){
            temp = img[imageIndex++];
            out << temp << ' ';
        }
        for (j=0;j<10;j++)
            out << "255 ";//255 is pure white color in grayscale
    }
    //White border on bottom of image
    for (i=0;i<newWidth*10;i++)
        out << "255 ";//255 is pure white color in grayscale
    out.close();
    return 0;
}
```

Explanations :

The lecture 4 is all practical based. We just took an image in .pgm format and copied all of its contents to a new file. The answer to the question of why we used the .pgm format here is that other formats like .jpg .png store the image in binary format and we can not actually see the values and pixel densities of the image. In .pgm format the image is stored in ASCII values so that we can actually see the values and their effects. Till now, We all know how to open a file and take input from a file and store it into a dynamic array and then open another empty file and store the contents of array into the file. The problem arises when we need to apply certain modifications in the file like adding borders or dividing the image, cropping the image etc. We can only give examples and samples of the code to do a certain task but it requires only practice to master the skill. In the above examples we added borders of ten pixels to the image on left, right, top and bottom. Only we are adding some more values into the image but the logic is where to add the values. You can practice by performing certain actions on the image. For example, you can add a border of 20 pixels instead of 10 to observe how the total size of the image changes. You can add lines into the image at certain positions. You can try cropping the image to different sizes. You can try rotating / flipping the image. You can even change the shades of the colors at certain positions to add some effects to the image.