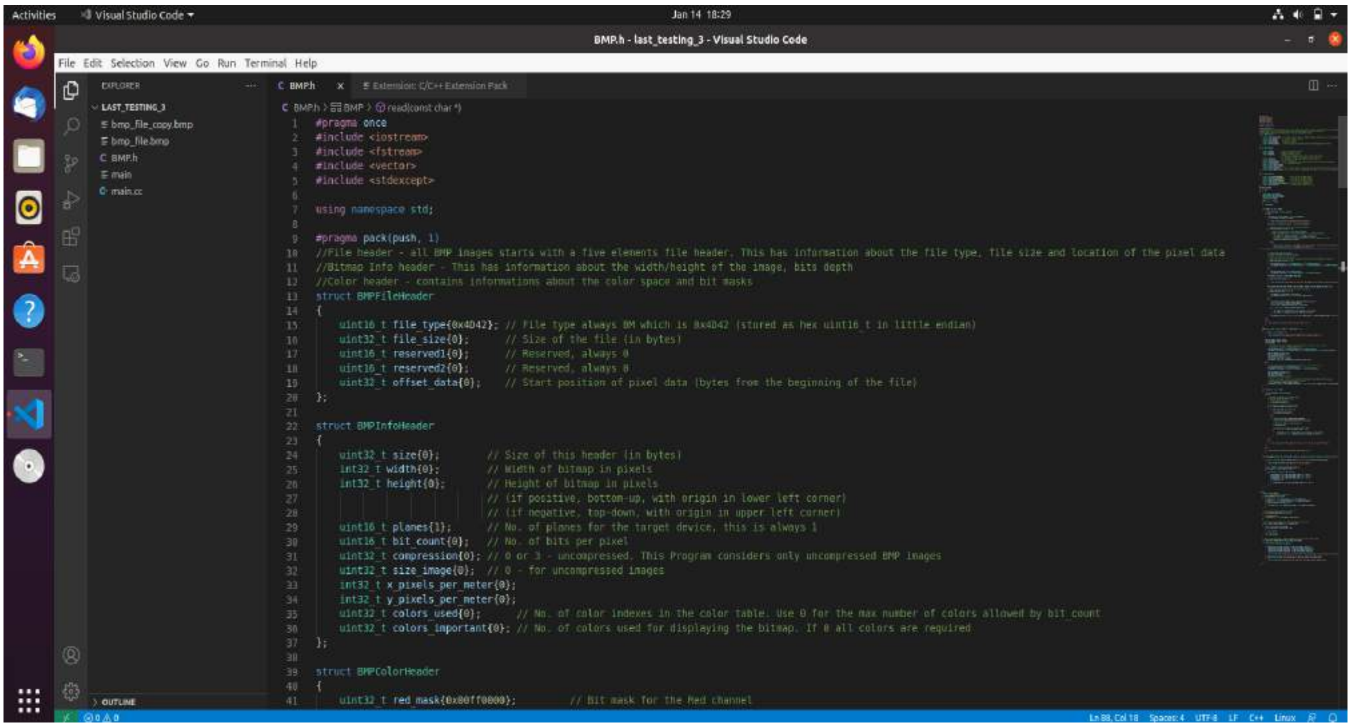
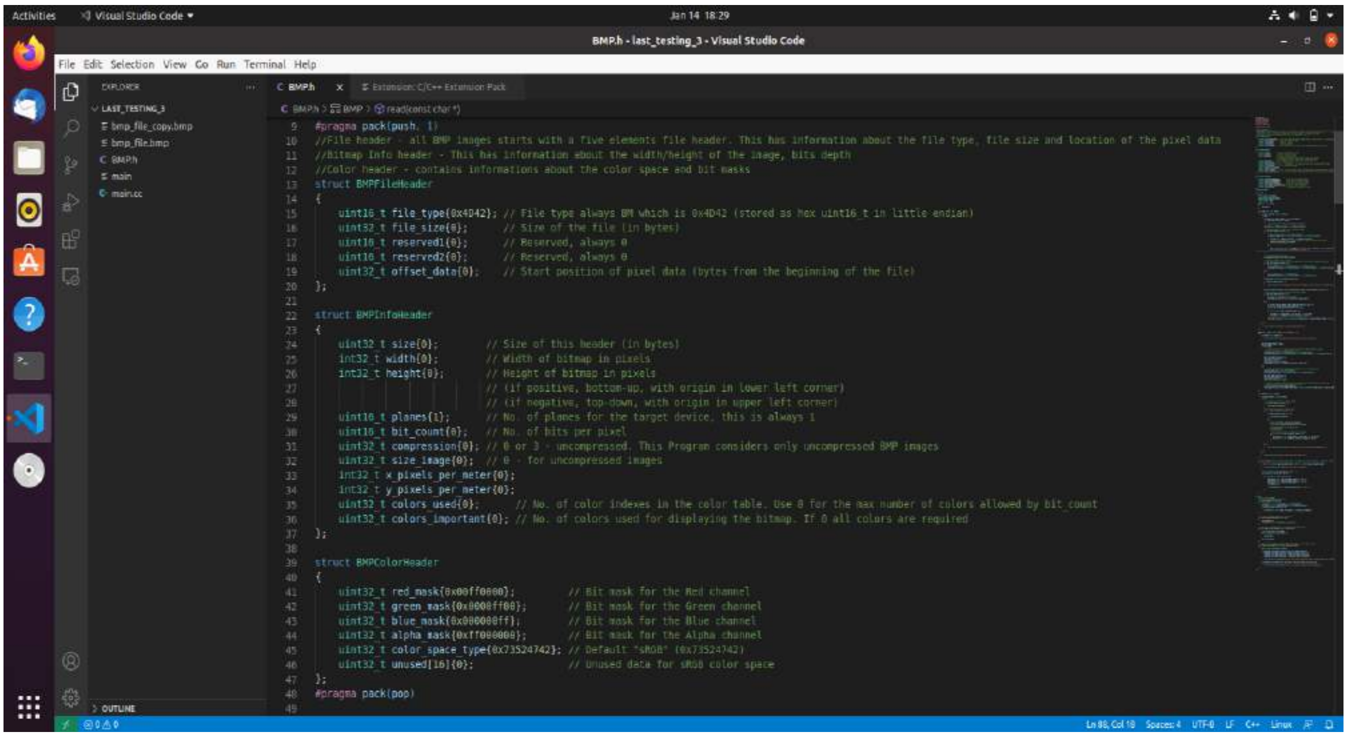
Task #3 Implement an Image File Reader and Writer

1. Structure of BMPFileHeader, BMPInfoHeader, BMPColorHeader.

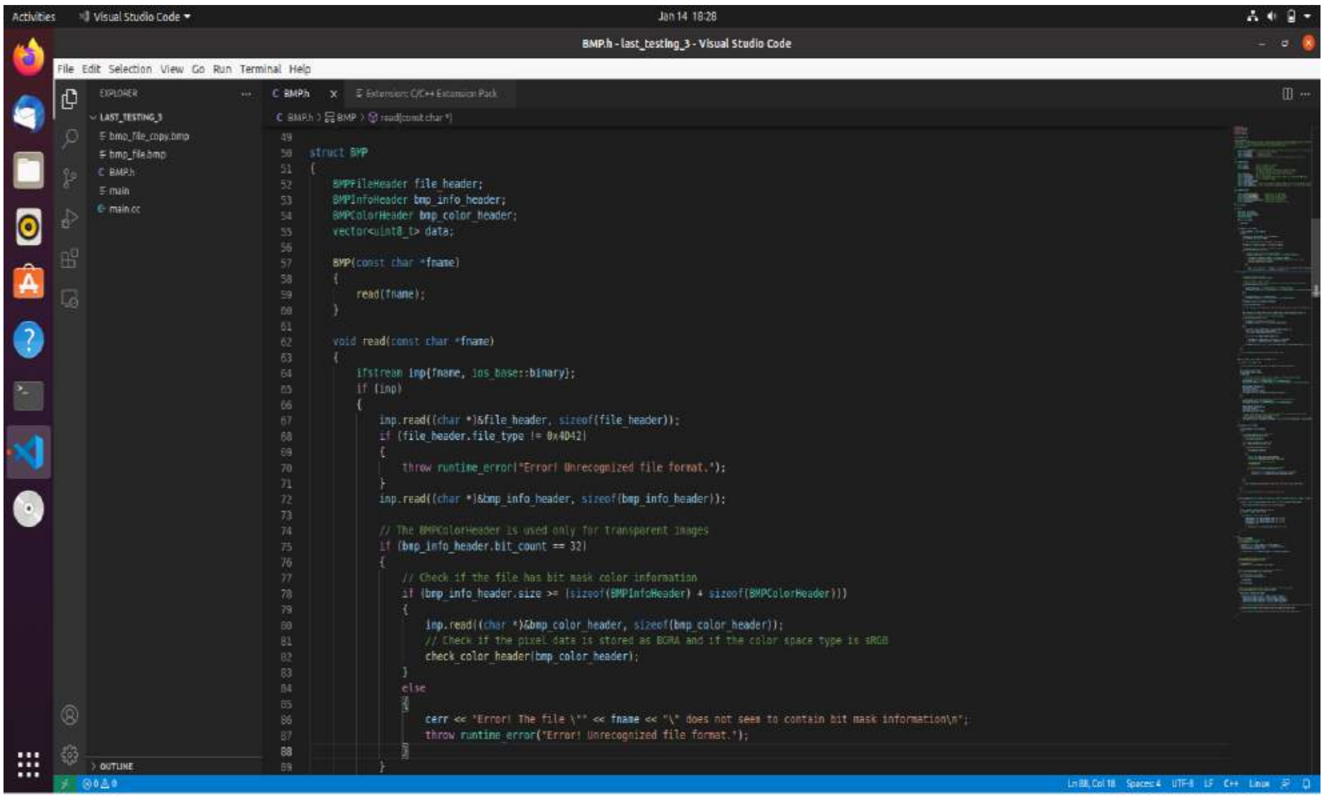

Activities
Visual Studio Code
Jan 14 18:29
BMP.h - last_testing_3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
LAST_TESTING_3
bmp_file_copy.bmp
bmp_file.bmp
BMP.h
main
main.cc
BMP.h
Extension: C/C++ Extension Pack
BMP.h > BMP > read(const char *)
OUTLINE
Ln 88, Col 18 Spaces: 4 UTF-8 LF C++ Linux

```cpp
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <stdexcept>

using namespace std;

#pragma pack(push, 1)
//File header - all BMP images starts with a five elements file header. This has information about the file type, file size and location of the pixel data
//Bitmap Info header - This has information about the width/height of the image, bits depth
//Color header - contains informations about the color space and bit masks
struct BMPFileHeader
{
    uint16_t file_type{0x4D42}; // File type always BM which is 0x4D42 (stored as hex uint16_t in little endian)
    uint32_t file_size{0};      // Size of the file (in bytes)
    uint16_t reserved1{0};      // Reserved, always 0
    uint16_t reserved2{0};      // Reserved, always 0
    uint32_t offset_data{0};    // Start position of pixel data (bytes from the beginning of the file)
};

struct BMPInfoHeader
{
    uint32_t size{0};        // Size of this header (in bytes)
    int32_t width{0};        // Width of bitmap in pixels
    int32_t height{0};       // Height of bitmap in pixels
                             // (if positive, bottom-up, with origin in lower left corner)
                             // (if negative, top-down, with origin in upper left corner)
    uint16_t planes{1};      // No. of planes for the target device, this is always 1
    uint16_t bit_count{0};   // No. of bits per pixel
    uint32_t compression{0}; // 0 or 3 - uncompressed. This Program considers only uncompressed BMP images
    uint32_t size_image{0};  // 0 - for uncompressed images
    int32_t x_pixels_per_meter{0};
    int32_t y_pixels_per_meter{0};
    uint32_t colors_used{0};      // No. of color indexes in the color table. Use 0 for the max number of colors allowed by bit_count
    uint32_t colors_important{0}; // No. of colors used for displaying the bitmap. If 0 all colors are required
};

struct BMPColorHeader
{
    uint32_t red_mask{0x00ff0000};         // Bit mask for the Red channel
```

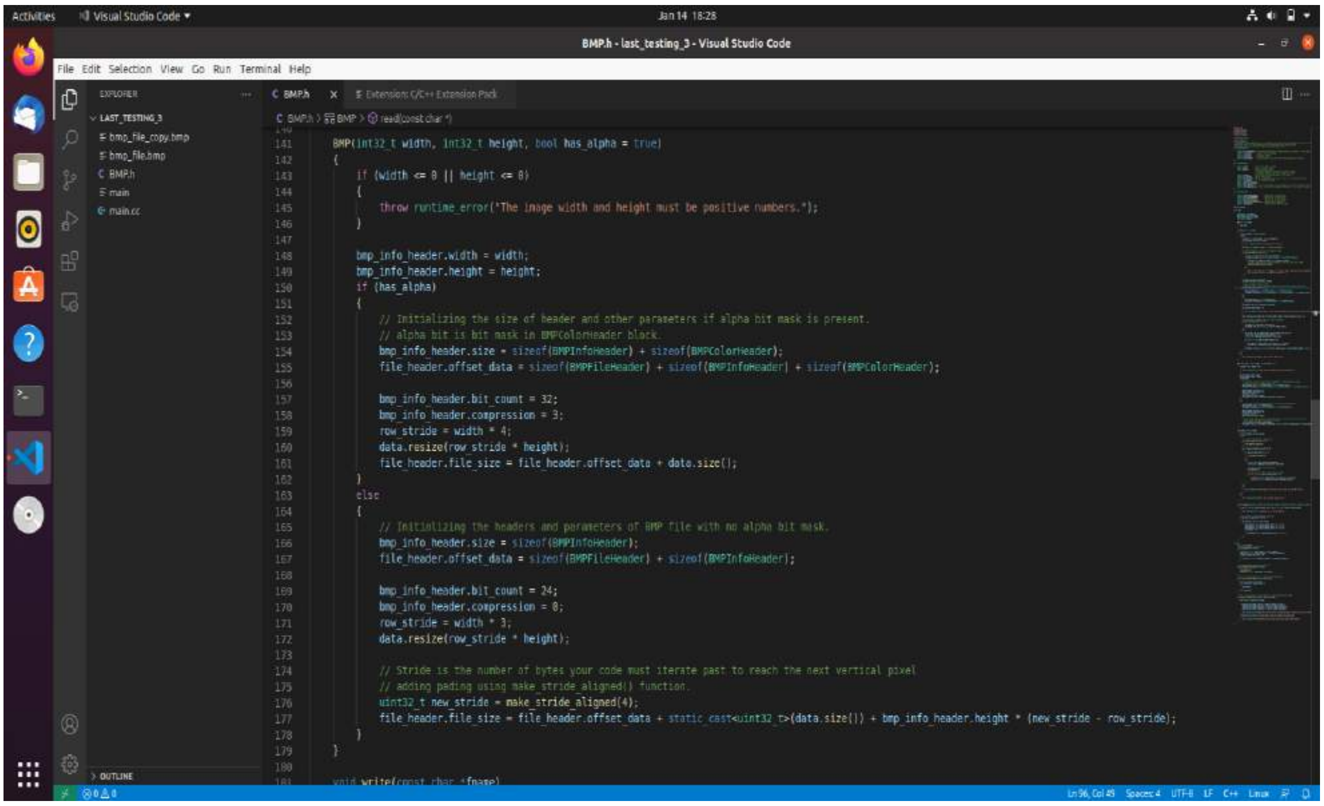

Activities
Visual Studio Code
Jan 14 18:29
BMP.h - last_testing_3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
LAST_TESTING_3
bmp_file_copy.bmp
bmp_file.bmp
BMP.h
main
main.cc
BMP.h
Extension: C/C++ Extension Pack
BMP.h > BMP > read(const char *)
OUTLINE
Ln 88, Col 18 Spaces: 4 UTF-8 LF C++ Linux

```cpp
#pragma pack(push, 1)
//File header - all BMP images starts with a five elements file header. This has information about the file type, file size and location of the pixel data
//Bitmap Info header - This has information about the width/height of the image, bits depth
//Color header - contains informations about the color space and bit masks
struct BMPFileHeader
{
    uint16_t file_type{0x4D42}; // File type always BM which is 0x4D42 (stored as hex uint16_t in little endian)
    uint32_t file_size{0};      // Size of the file (in bytes)
    uint16_t reserved1{0};      // Reserved, always 0
    uint16_t reserved2{0};      // Reserved, always 0
    uint32_t offset_data{0};    // Start position of pixel data (bytes from the beginning of the file)
};

struct BMPInfoHeader
{
    uint32_t size{0};        // Size of this header (in bytes)
    int32_t width{0};        // Width of bitmap in pixels
    int32_t height{0};       // Height of bitmap in pixels
                             // (if positive, bottom-up, with origin in lower left corner)
                             // (if negative, top-down, with origin in upper left corner)
    uint16_t planes{1};      // No. of planes for the target device, this is always 1
    uint16_t bit_count{0};   // No. of bits per pixel
    uint32_t compression{0}; // 0 or 3 - uncompressed. This Program considers only uncompressed BMP images
    uint32_t size_image{0};  // 0 - for uncompressed images
    int32_t x_pixels_per_meter{0};
    int32_t y_pixels_per_meter{0};
    uint32_t colors_used{0};      // No. of color indexes in the color table. Use 0 for the max number of colors allowed by bit_count
    uint32_t colors_important{0}; // No. of colors used for displaying the bitmap. If 0 all colors are required
};

struct BMPColorHeader
{
    uint32_t red_mask{0x00ff0000};         // Bit mask for the Red channel
    uint32_t green_mask{0x0000ff00};       // Bit mask for the Green channel
    uint32_t blue_mask{0x000000ff};        // Bit mask for the Blue channel
    uint32_t alpha_mask{0xff000000};       // Bit mask for the Alpha channel
    uint32_t color_space_type{0x73524742}; // Default "sRGB" (0x73524742)
    uint32_t unused[16]{0};                // Unused data for sRGB color space
};
#pragma pack(pop)
```

2. BMP constructor

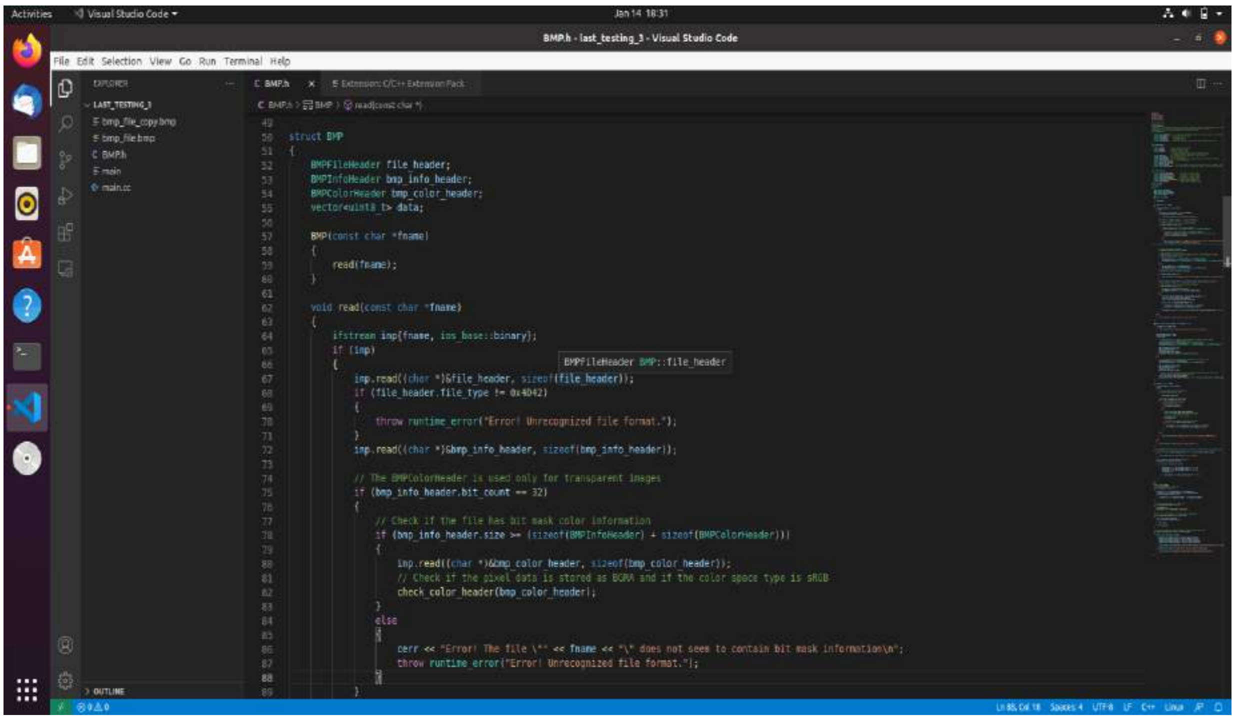


```cpp
struct BMP
{
    BMPFileHeader file_header;
    BMPInfoHeader bmp_info_header;
    BMPColorHeader bmp_color_header;
    vector<uint8_t> data;

    BMP(const char *fname)
    {
        read(fname);
    }

    void read(const char *fname)
    {
        ifstream inp{fname, ios_base::binary};
        if (inp)
        {
            inp.read((char *)&file_header, sizeof(file_header));
            if (file_header.file_type != 0x4D42)
            {
                throw runtime_error("Error! Unrecognized file format.");
            }
            inp.read((char *)&bmp_info_header, sizeof(bmp_info_header));

            // The BMPColorHeader is used only for transparent images
            if (bmp_info_header.bit_count == 32)
            {
                // Check if the file has bit mask color information
                if (bmp_info_header.size >= (sizeof(BMPInfoHeader) + sizeof(BMPColorHeader)))
                {
                    inp.read((char *)&bmp_color_header, sizeof(bmp_color_header));
                    // Check if the pixel data is stored as BGRA and if the color space type is sRGB
                    check_color_header(bmp_color_header);
                }
                else
                {
                    cerr << "Error! The file \"" << fname << "\" does not seem to contain bit mask information\n";
                    throw runtime_error("Error! Unrecognized file format.");
                }

        }
```

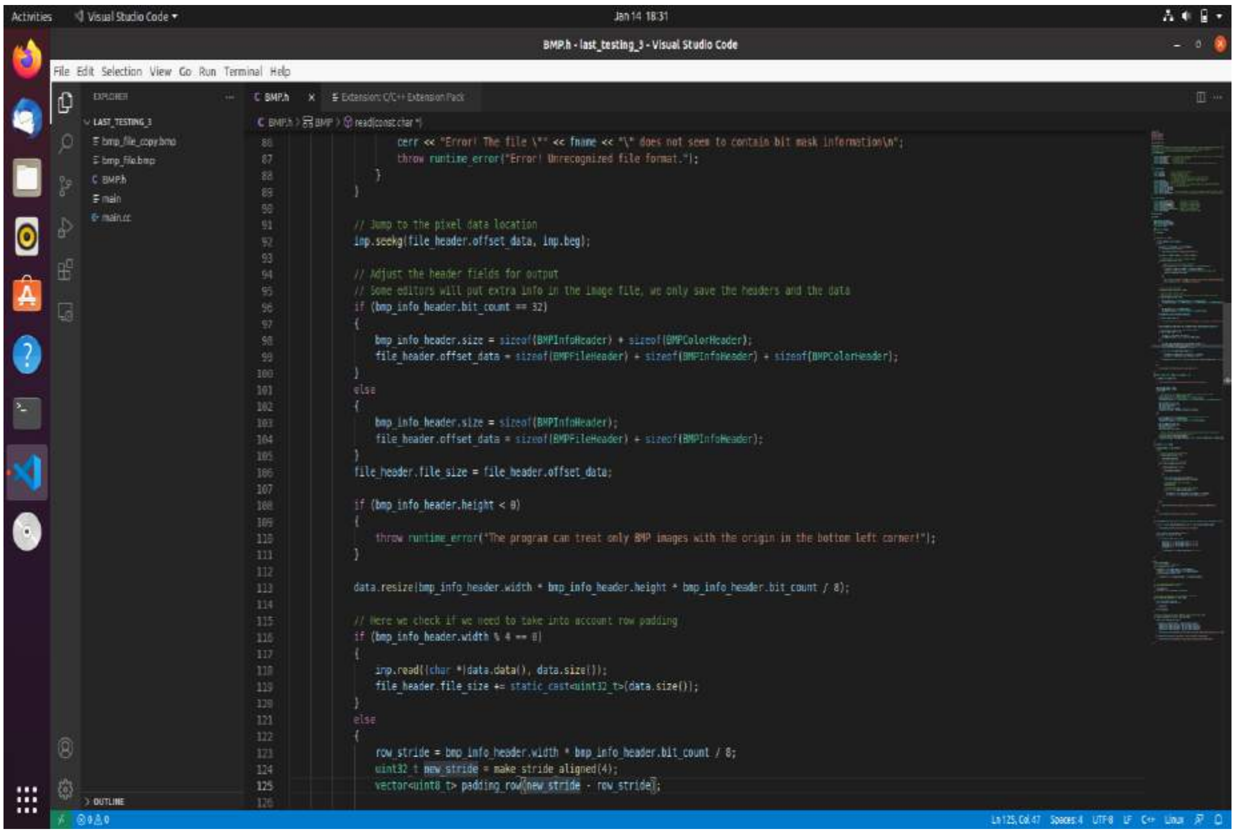


```cpp
    BMP(int32_t width, int32_t height, bool has_alpha = true)
    {
        if (width <= 0 || height <= 0)
        {
            throw runtime_error("The image width and height must be positive numbers.");
        }

        bmp_info_header.width = width;
        bmp_info_header.height = height;
        if (has_alpha)
        {
            // Initializing the size of header and other parameters if alpha bit mask is present.
            // alpha bit is bit mask in BMPColorHeader block.
            bmp_info_header.size = sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
            file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);

            bmp_info_header.bit_count = 32;
            bmp_info_header.compression = 3;
            row_stride = width * 4;
            data.resize(row_stride * height);
            file_header.file_size = file_header.offset_data + data.size();
        }
        else
        {
            // Initializing the headers and parameters of BMP file with no alpha bit mask.
            bmp_info_header.size = sizeof(BMPInfoHeader);
            file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader);

            bmp_info_header.bit_count = 24;
            bmp_info_header.compression = 0;
            row_stride = width * 3;
            data.resize(row_stride * height);

            // Stride is the number of bytes your code must iterate past to reach the next vertical pixel
            // adding pading using make_stride_aligned() function.
            uint32_t new_stride = make_stride_aligned(4);
            file_header.file_size = file_header.offset_data + static_cast<uint32_t>(data.size()) + bmp_info_header.height * (new_stride - row_stride);
        }
    }

    void write(const char *fname)
```

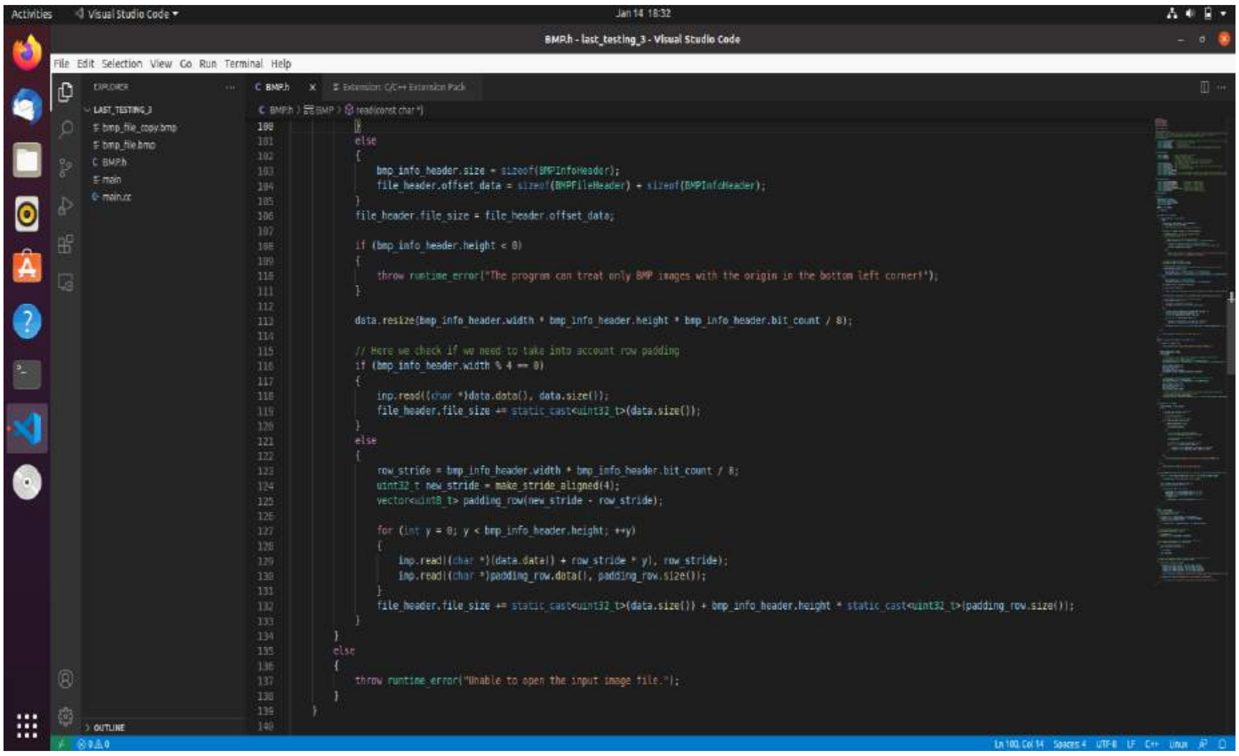
3. Function to Read Bitmap Image



```cpp
struct BMP
{
    BMPFileHeader file_header;
    BMPInfoHeader bmp_info_header;
    BMPColorHeader bmp_color_header;
    vector<uint8_t> data;

    BMP(const char *fname)
    {
        read(fname);
    }

    void read(const char *fname)
    {
        ifstream inp{fname, ios_base::binary};
        if (inp)
        {
            inp.read((char *)&file_header, sizeof(file_header));
            if (file_header.file_type != 0x4D42)
            {
                throw runtime_error("Error! Unrecognized file format.");
            }
            inp.read((char *)&bmp_info_header, sizeof(bmp_info_header));

            // The BMPColorHeader is used only for transparent images
            if (bmp_info_header.bit_count == 32)
            {
                // Check if the file has bit mask color information
                if (bmp_info_header.size >= (sizeof(BMPInfoHeader) + sizeof(BMPColorHeader)))
                {
                    inp.read((char *)&bmp_color_header, sizeof(bmp_color_header));
                    // Check if the pixel data is stored as BGRA and if the color space type is sRGB
                    check_color_header(bmp_color_header);
                }
                else
                {
                    cerr << "Error! The file \"" << fname << "\" does not seem to contain bit mask information\n";
                    throw runtime_error("Error! Unrecognized file format.");
                }
            }
```

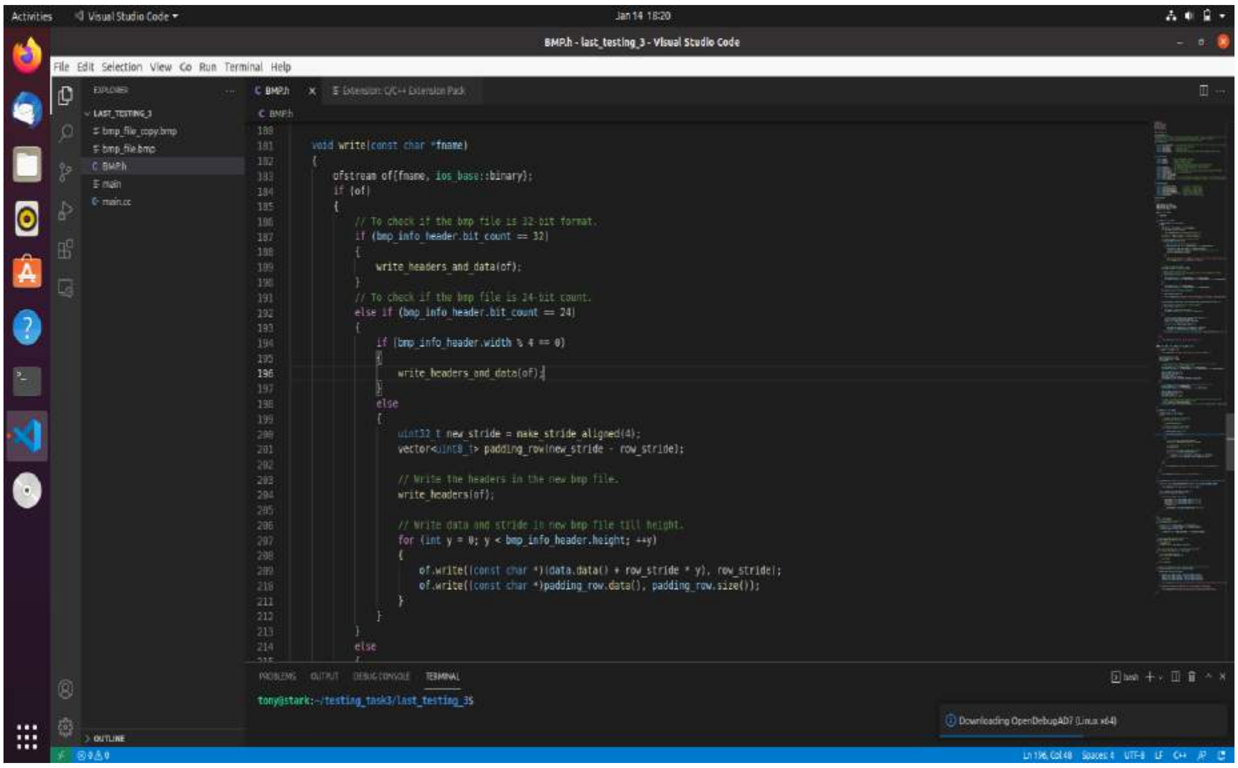


```cpp
                    cerr << "Error! The file \"" << fname << "\" does not seem to contain bit mask information\n";
                    throw runtime_error("Error! Unrecognized file format.");
                }
            }

            // Jump to the pixel data location
            inp.seekg(file_header.offset_data, inp.beg);

            // Adjust the header fields for output
            // Some editors will put extra info in the image file, we only save the headers and the data
            if (bmp_info_header.bit_count == 32)
            {
                bmp_info_header.size = sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
                file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
            }
            else
            {
                bmp_info_header.size = sizeof(BMPInfoHeader);
                file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader);
            }
            file_header.file_size = file_header.offset_data;

            if (bmp_info_header.height < 0)
            {
                throw runtime_error("The program can treat only BMP images with the origin in the bottom left corner!");
            }

            data.resize(bmp_info_header.width * bmp_info_header.height * bmp_info_header.bit_count / 8);

            // Here we check if we need to take into account row padding
            if (bmp_info_header.width % 4 == 0)
            {
                inp.read((char *)data.data(), data.size());
                file_header.file_size += static_cast<uint32_t>(data.size());
            }
            else
            {
                row_stride = bmp_info_header.width * bmp_info_header.bit_count / 8;
                uint32_t new_stride = make_stride_aligned(4);
                vector<uint8_t> padding_row(new_stride - row_stride);
```



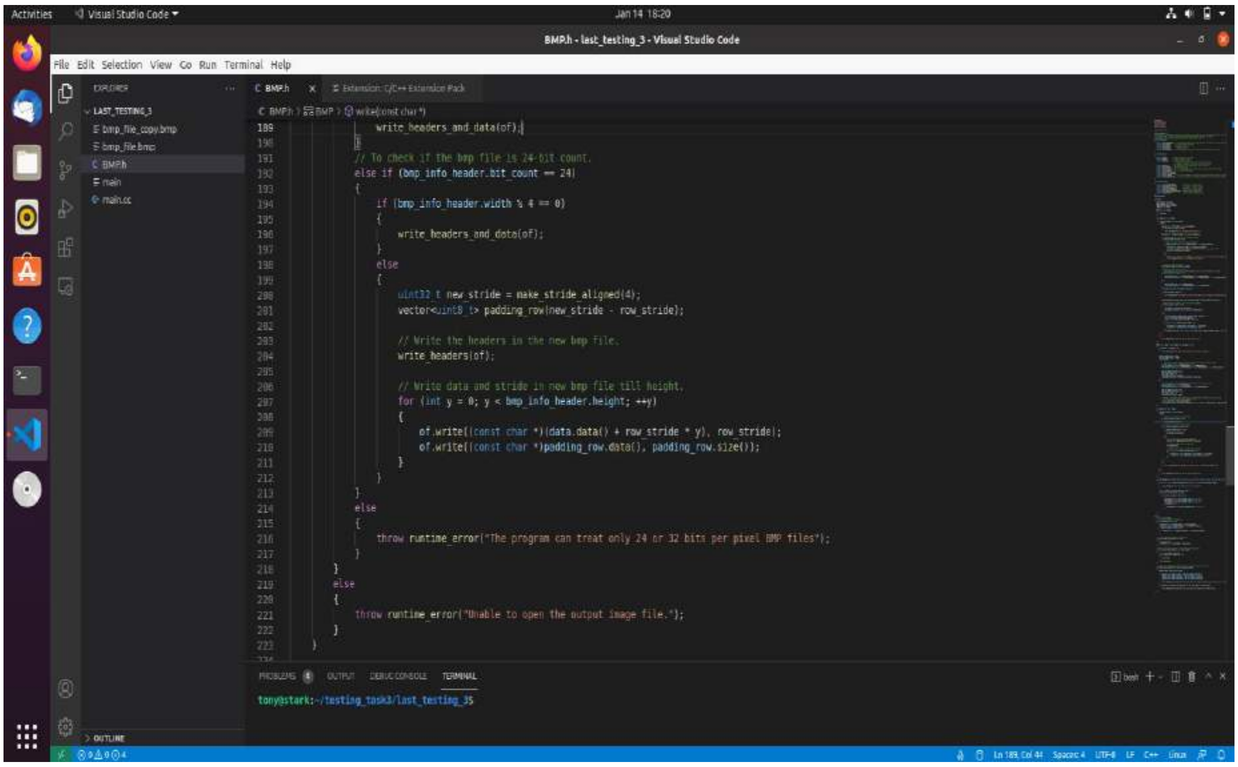
```
            }
            else
            {
                bmp_info_header.size = sizeof(BMPInfoHeader);
                file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader);
            }
            file_header.file_size = file_header.offset_data;

            if (bmp_info_header.height < 0)
            {
                throw runtime_error("The program can treat only BMP images with the origin in the bottom left corner!");
            }

            data.resize(bmp_info_header.width * bmp_info_header.height * bmp_info_header.bit_count / 8);

            // Here we check if we need to take into account row padding
            if (bmp_info_header.width % 4 == 0)
            {
                inp.read((char *)data.data(), data.size());
                file_header.file_size += static_cast<uint32_t>(data.size());
            }
            else
            {
                row_stride = bmp_info_header.width * bmp_info_header.bit_count / 8;
                uint32_t new_stride = make_stride_aligned(4);
                vector<uint8_t> padding_row(new_stride - row_stride);

                for (int y = 0; y < bmp_info_header.height; ++y)
                {
                    inp.read((char *)(data.data() + row_stride * y), row_stride);
                    inp.read((char *)padding_row.data(), padding_row.size());
                }
                file_header.file_size += static_cast<uint32_t>(data.size()) + bmp_info_header.height * static_cast<uint32_t>(padding_row.size());
            }
        }
        else
        {
            throw runtime_error("Unable to open the input image file.");
        }
    }
```

4. Function to Write Bitmap Image



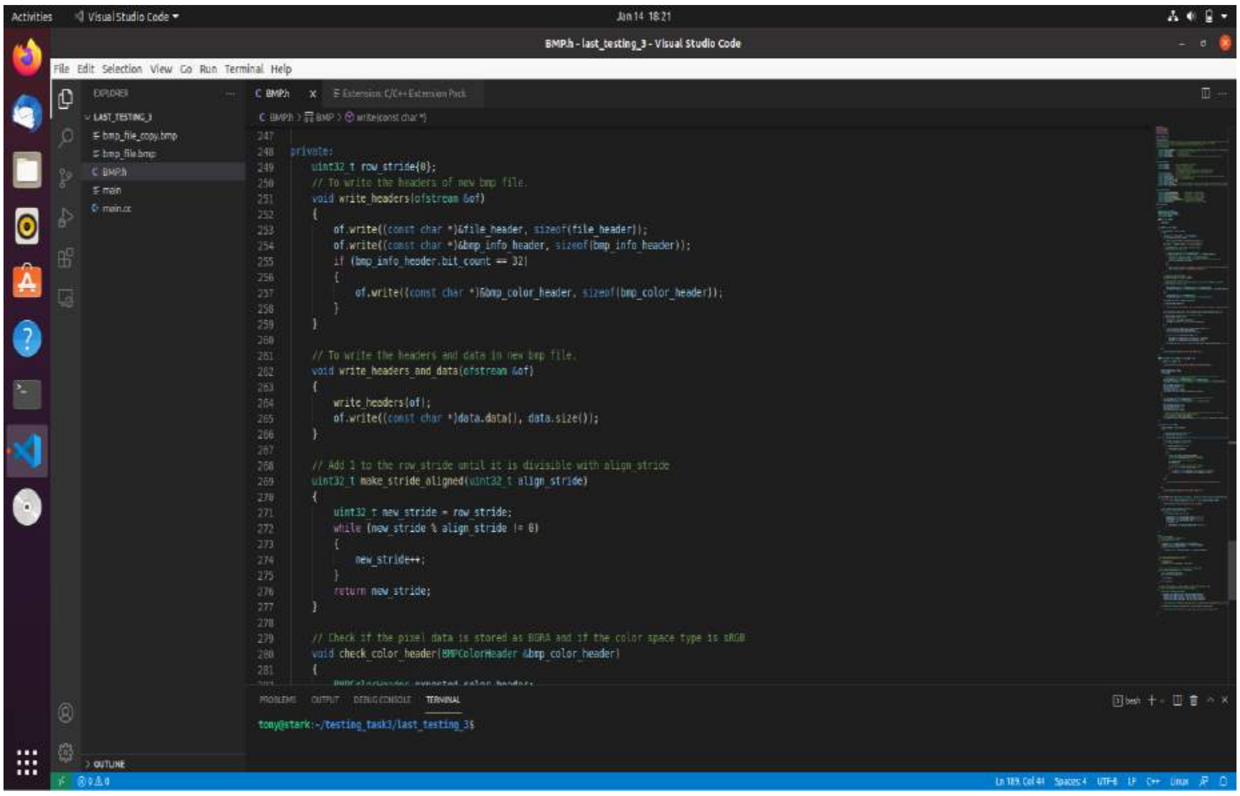
```
    void write(const char *fname)
    {
        ofstream of{fname, ios_base::binary};
        if (of)
        {
            // To check if the bmp file is 32-bit format.
            if (bmp_info_header.bit_count == 32)
            {
                write_headers_and_data(of);
            }
            // To check if the bmp file is 24-bit count.
            else if (bmp_info_header.bit_count == 24)
            {
                if (bmp_info_header.width % 4 == 0)
                {
                    write_headers_and_data(of);
                }
                else
                {
                    uint32_t new_stride = make_stride_aligned(4);
                    vector<uint8_t> padding_row(new_stride - row_stride);

                    // Write the headers in the new bmp file.
                    write_headers(of);

                    // Write data and stride in new bmp file till height.
                    for (int y = 0; y < bmp_info_header.height; ++y)
                    {
                        of.write((const char *)(data.data() + row_stride * y), row_stride);
                        of.write((const char *)padding_row.data(), padding_row.size());
                    }
                }
            }
            else
```

```
tony@stark:~/testing_task3/last_testing_3$
```



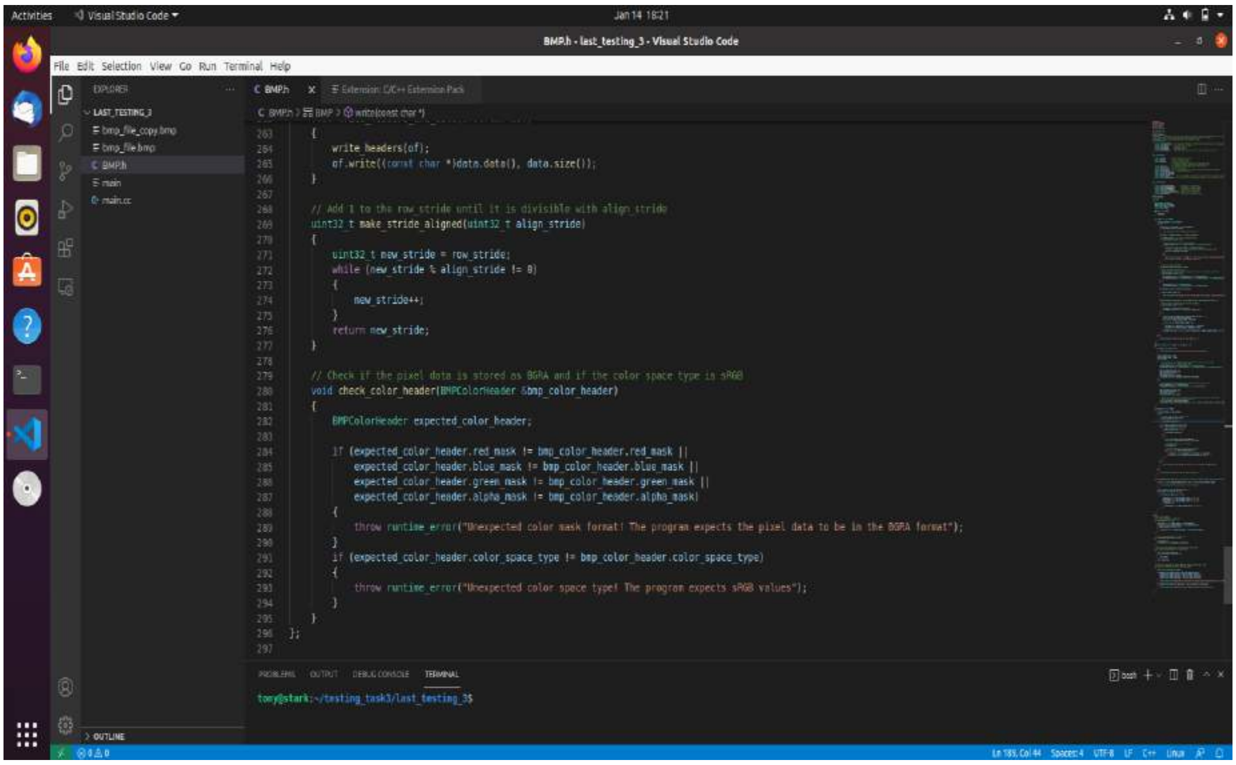
```
                write_headers_and_data(of);
            }
            // To check if the bmp file is 24-bit count.
            else if (bmp_info_header.bit_count == 24)
            {
                if (bmp_info_header.width % 4 == 0)
                {
                    write_headers_and_data(of);
                }
                else
                {
                    uint32_t new_stride = make_stride_aligned(4);
                    vector<uint8_t> padding_row(new_stride - row_stride);

                    // Write the headers in the new bmp file.
                    write_headers(of);

                    // Write data and stride in new bmp file till height.
                    for (int y = 0; y < bmp_info_header.height; ++y)
                    {
                        of.write((const char *)(data.data() + row_stride * y), row_stride);
                        of.write((const char *)padding_row.data(), padding_row.size());
                    }
                }
            }
            else
            {
                throw runtime_error("The program can treat only 24 or 32 bits per pixel BMP files");
            }
        }
        else
        {
            throw runtime_error("Unable to open the output image file.");
        }
    }
```

```
tony@stark:~/testing_task3/last_testing_3$
```



```

private:
    uint32_t row_stride{0};
    // To write the headers of new bmp file.
    void write_headers(ofstream &of)
    {
        of.write((const char *)&file_header, sizeof(file_header));
        of.write((const char *)&bmp_info_header, sizeof(bmp_info_header));
        if (bmp_info_header.bit_count == 32)
        {
            of.write((const char *)&bmp_color_header, sizeof(bmp_color_header));
        }
    }

    // To write the headers and data in new bmp file.
    void write_headers_and_data(ofstream &of)
    {
        write_headers(of);
        of.write((const char *)data.data(), data.size());
    }

    // Add 1 to the row_stride until it is divisible with align_stride
    uint32_t make_stride_aligned(uint32_t align_stride)
    {
        uint32_t new_stride = row_stride;
        while (new_stride % align_stride != 0)
        {
            new_stride++;
        }
        return new_stride;
    }

    // Check if the pixel data is stored as BGRA and if the color space type is sRGB
    void check_color_header(BMPColorHeader &bmp_color_header)
    {
```

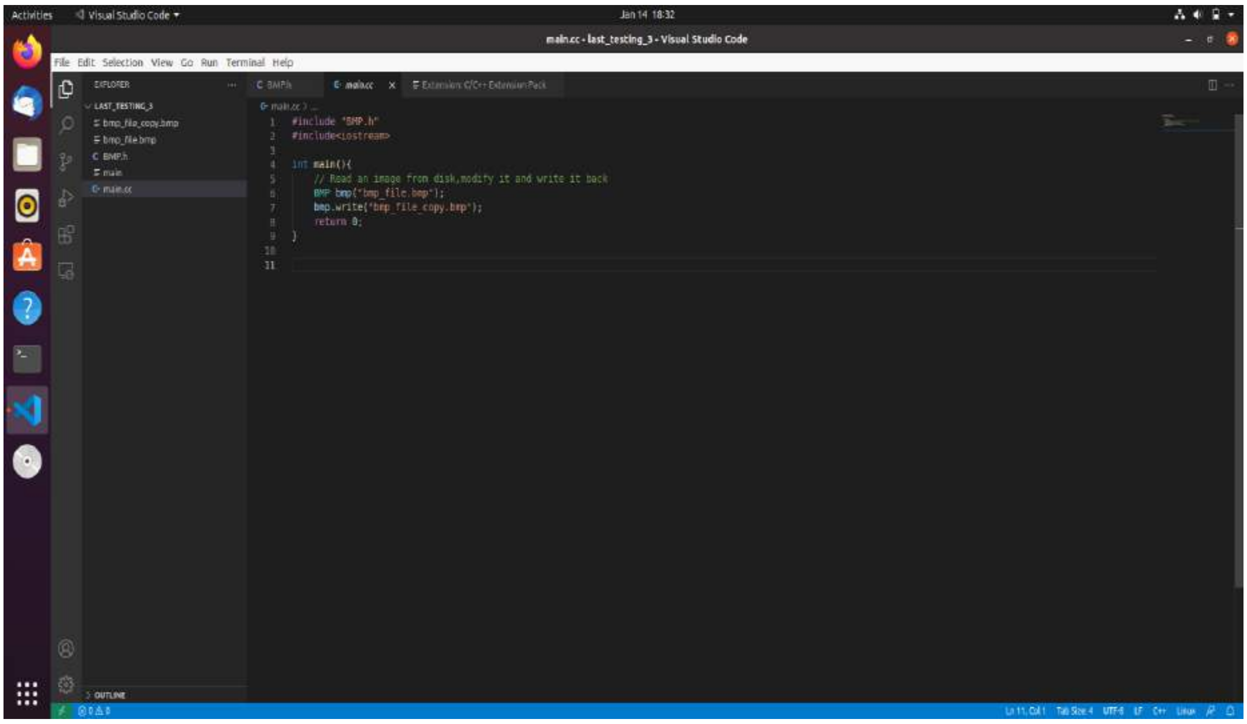
```
tony@stark:~/testing_task3/last_testing_3$
```



```
    {
        write_headers(of);
        of.write((const char *)data.data(), data.size());
    }

    // Add 1 to the row_stride until it is divisible with align_stride
    uint32_t make_stride_aligned(uint32_t align_stride)
    {
        uint32_t new_stride = row_stride;
        while (new_stride % align_stride != 0)
        {
            new_stride++;
        }
        return new_stride;
    }

    // Check if the pixel data is stored as BGRA and if the color space type is sRGB
    void check_color_header(BMPColorHeader &bmp_color_header)
    {
        BMPColorHeader expected_color_header;

        if (expected_color_header.red_mask != bmp_color_header.red_mask ||
            expected_color_header.blue_mask != bmp_color_header.blue_mask ||
            expected_color_header.green_mask != bmp_color_header.green_mask ||
            expected_color_header.alpha_mask != bmp_color_header.alpha_mask)
        {
            throw runtime_error("Unexpected color mask format! The program expects the pixel data to be in the BGRA format");
        }
        if (expected_color_header.color_space_type != bmp_color_header.color_space_type)
        {
            throw runtime_error("Unexpected color space type! The program expects sRGB values");
        }
    }
};
```

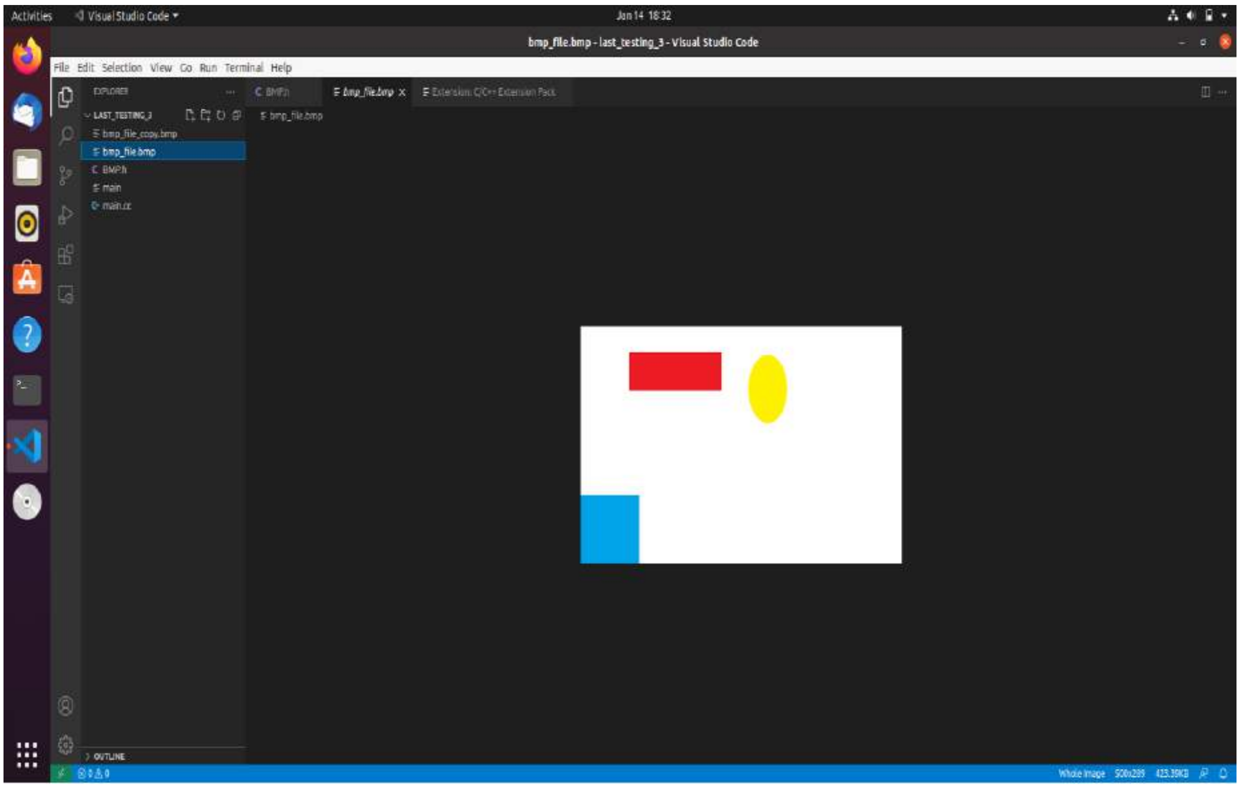
```
tony@stark:~/testing_task3/last_testing_3$
```

5. Main Function

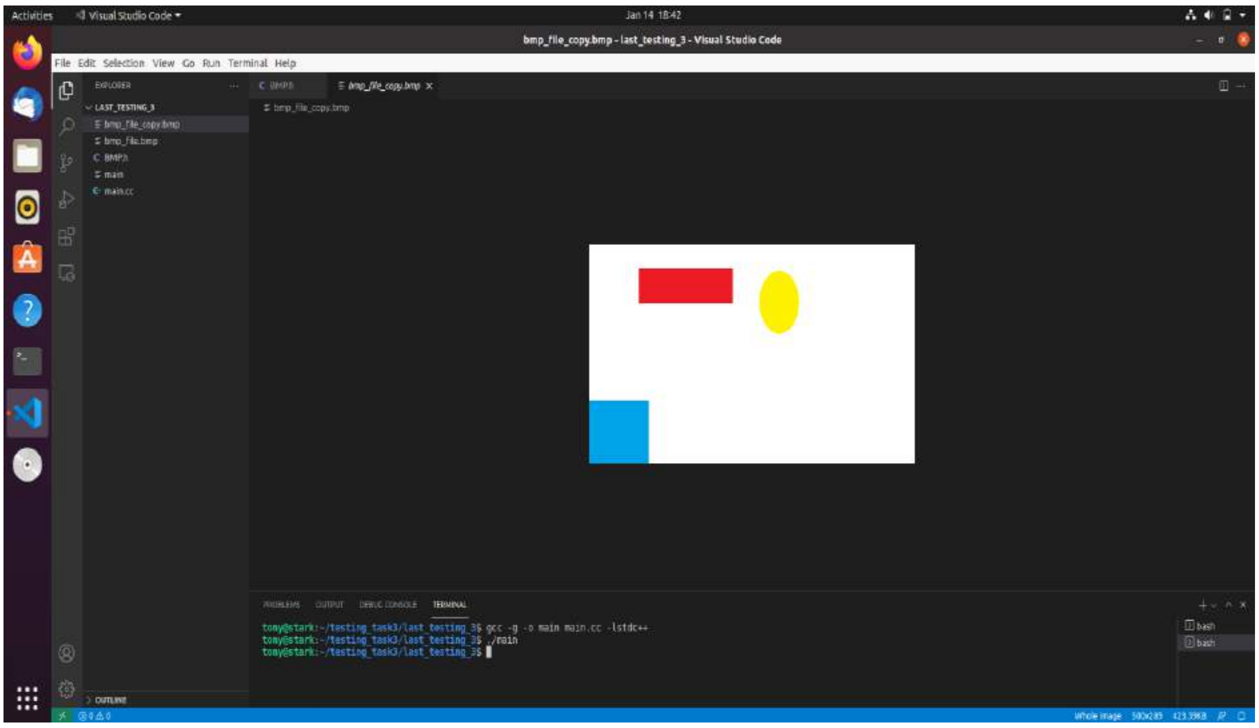


```
#include "BMP.h"
#include<iostream>

int main(){
    // Read an image from disk,modify it and write it back
    BMP bmp("bmp_file.bmp");
    bmp.write("bmp_file_copy.bmp");
    return 0;
}
```

6. Input (Read) Bitmap Image



7. Output (Write) Bitmap Image


Activities
Visual Studio Code
Jan 14 18:42
bmp_file_copy.bmp - last_testing_3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
LAST_TESTING_3
bmp_file_copy.bmp
bmp_file.bmp
BMP.h
main
main.cc
OUTLINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash
bash
Whole Image 500x289 423.39KB

```
tony@stark:~/testing_task3/last_testing_3$ gcc -g -o main main.cc -lstdc++
tony@stark:~/testing_task3/last_testing_3$ ./main
tony@stark:~/testing_task3/last_testing_3$ 
```