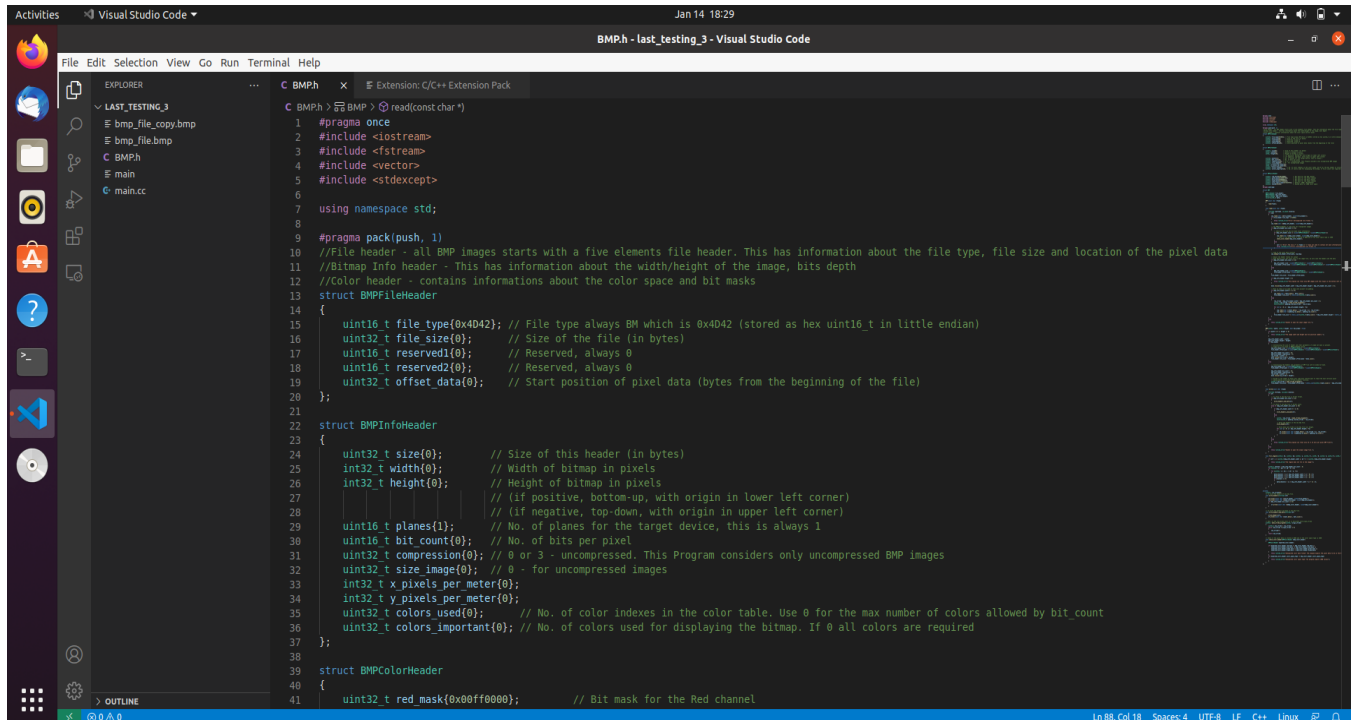
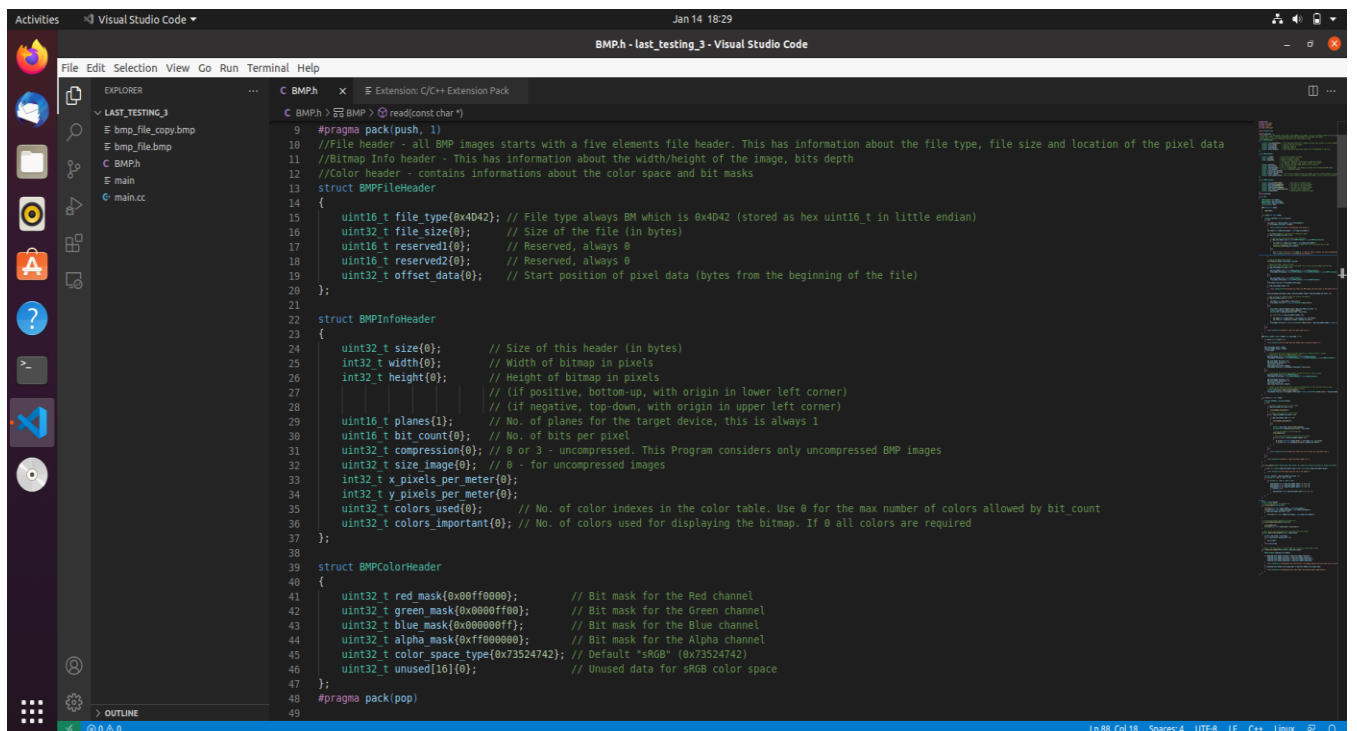


# Task #3 Implement an Image File Reader and Writer

## 1. Structure of BMPFileHeader, BMPInfoHeader, BMPColorHeader.

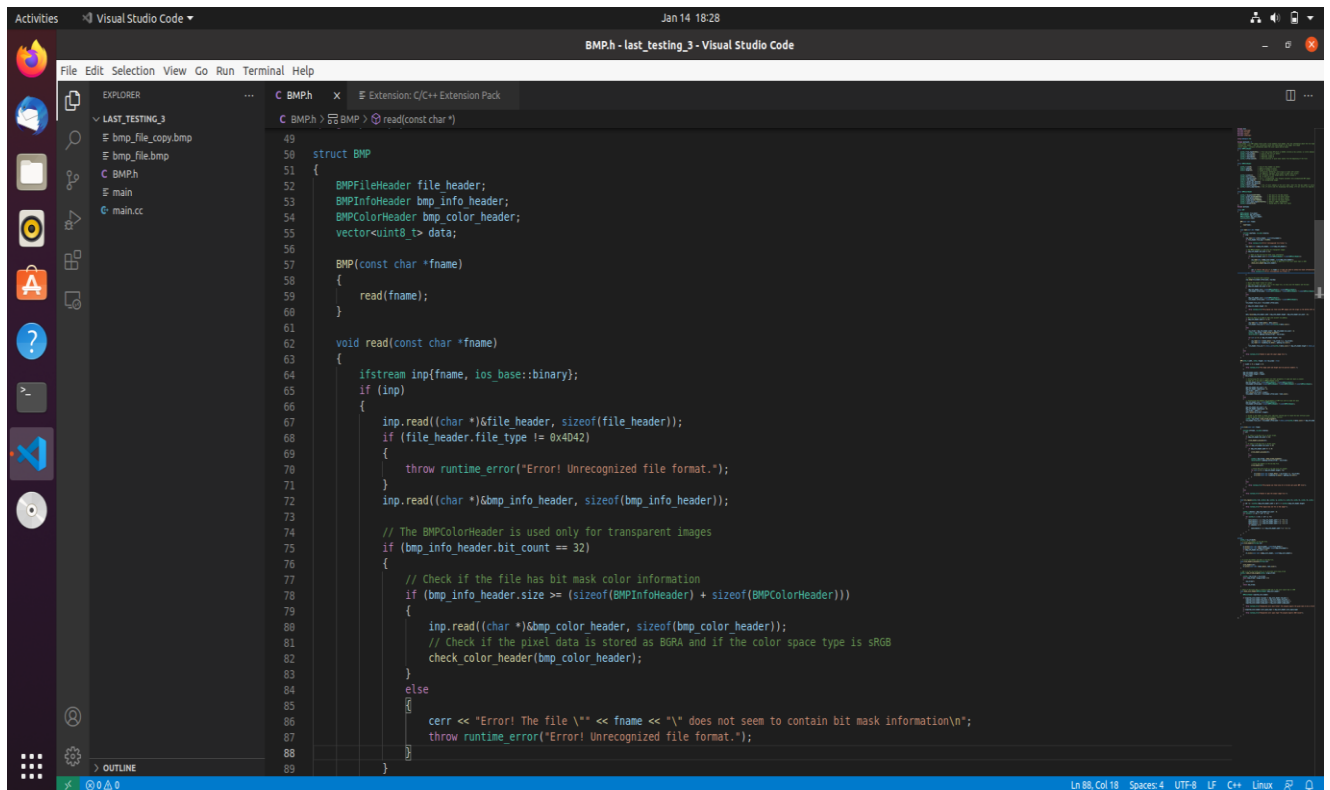


```
1 #pragma once
2 #include <iostream>
3 #include <fstream>
4 #include <vector>
5 #include <string>
6 using namespace std;
7
8 #pragma pack(push, 1)
9 //File header - all BMP images starts with a five elements file header. This has information about the file type, file size and location of the pixel data
10 //Bitmap Info header - This has information about the width/height of the image, bits depth
11 //Color header - contains informations about the color space and bit masks
12 struct BMPFileHeader
13 {
14     uint16_t file_type{0x4D52}; // File type always BM which is 0x4D52 (stored as hex uint16_t in little endian)
15     uint32_t file_size{0}; // Size of the file (in bytes)
16     uint16_t reserved1{0}; // Reserved, always 0
17     uint16_t reserved2{0}; // Reserved, always 0
18     uint32_t offset_data{0}; // Start position of pixel data (bytes from the beginning of the file)
19 };
20
21 struct BMPInfoHeader
22 {
23     uint32_t size{0}; // Size of this header (in bytes)
24     int32_t width{0}; // Width of bitmap in pixels
25     int32_t height{0}; // Height of bitmap in pixels
26     // (if positive, bottom-up, with origin in lower left corner)
27     // (if negative, top-down, with origin in upper left corner)
28     uint16_t planes{1}; // No. of planes for the target device, this is always 1
29     uint16_t bit_count{0}; // No. of bits per pixel
30     uint32_t compression{0}; // 0 or 3 - uncompressed. This Program considers only uncompressed BMP images
31     uint32_t size_image{0}; // 0 - for uncompressed images
32     int32_t x_pixels_per_meter{0};
33     int32_t y_pixels_per_meter{0};
34     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
35     uint32_t colors_important{0}; // No. of colors used for displaying the bitmap. If 0 all colors are required
36 };
37
38 struct BMPColorHeader
39 {
40     uint32_t red_mask{0x000000FF}; // Bit mask for the Red channel
```

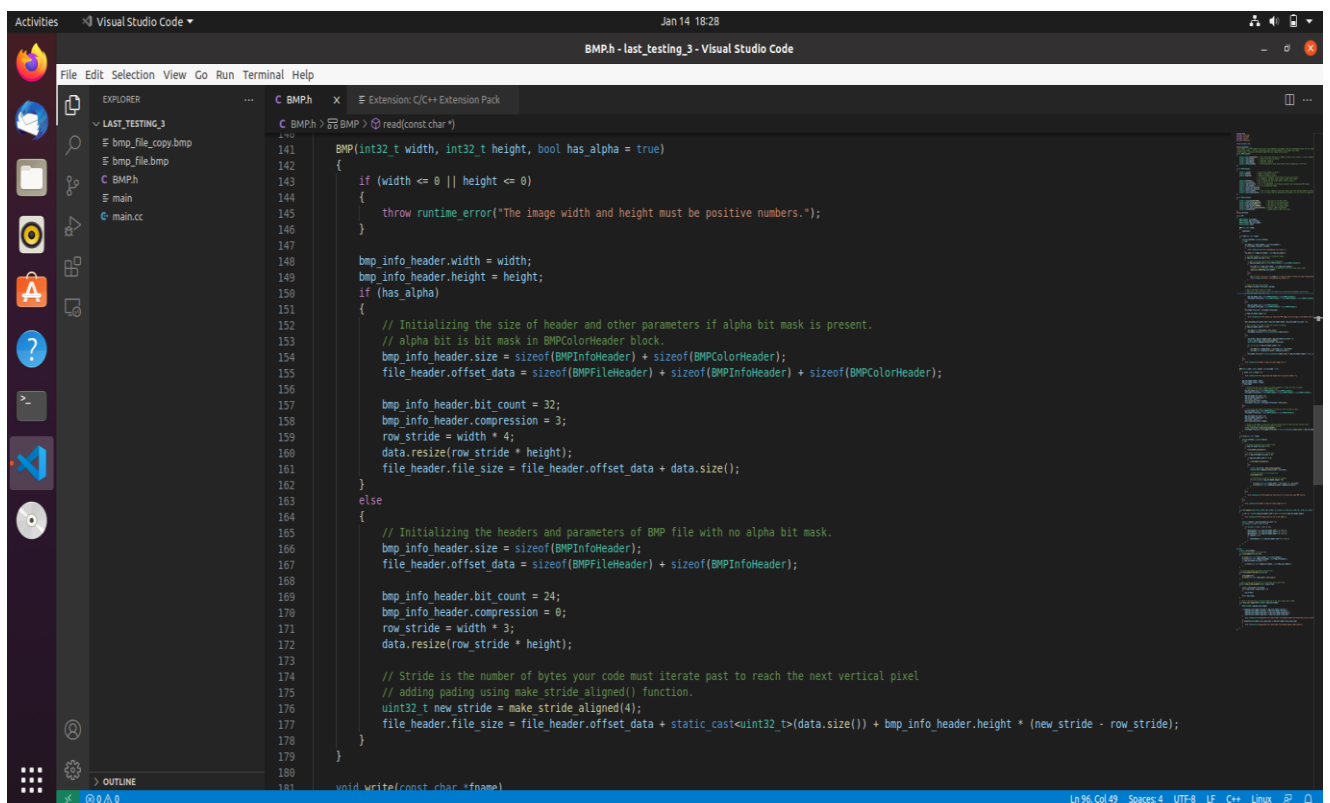


```
41     uint32_t green_mask{0x0000FF00}; // Bit mask for the Green channel
42     uint32_t blue_mask{0x00FF0000}; // Bit mask for the Blue channel
43     uint32_t alpha_mask{0xFF000000}; // Bit mask for the Alpha channel
44     uint32_t color_space_type{0x73524742}; // Default "sRGB" (0x73524742)
45     uint32_t unused16{0}; // Unused data for sRGB color space
46 };
47 #pragma pack(pop)
48
49 #endif
```

## 2. BMP constructor

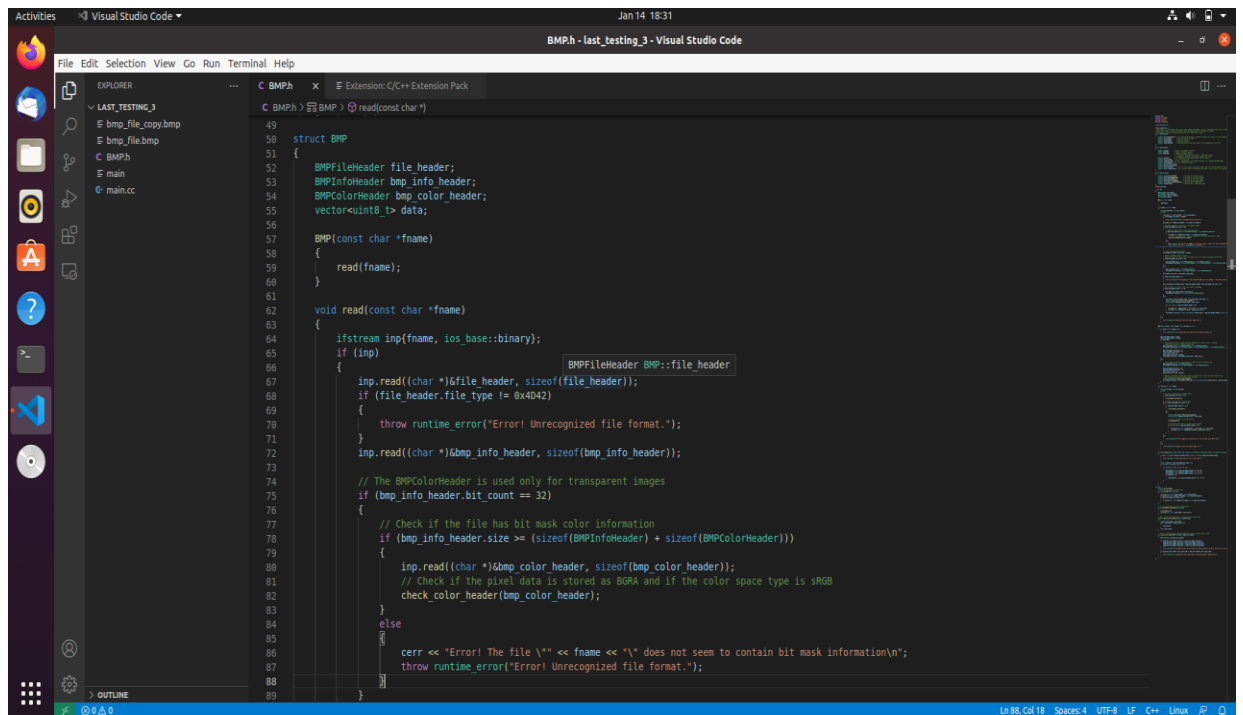


```
49 struct BMP
50 {
51     BMPFileHeader file_header;
52     BMPInfoHeader bmp_info_header;
53     BMPColorHeader bmp_color_header;
54     vector<uint8_t> data;
55
56     BMP(const char *fname)
57     {
58         read(fname);
59     }
60
61     void read(const char *fname)
62     {
63         ifstream inp(fname, ios_base::binary);
64         if (!inp)
65         {
66             throw runtime_error("Error! Unrecognized file format.");
67         }
68         inp.read((char *) &file_header, sizeof(file_header));
69         if (file_header.file_type != 0x4D42)
70         {
71             throw runtime_error("Error! Unrecognized file format.");
72         }
73         inp.read((char *) &bmp_info_header, sizeof(bmp_info_header));
74
75         // The BMPColorHeader is used only for transparent images
76         if (bmp_info_header.bit_count == 32)
77         {
78             // Check if the file has bit mask color information
79             if (bmp_info_header.size >= (sizeof(BMPInfoHeader) + sizeof(BMPColorHeader)))
80             {
81                 inp.read((char *) &bmp_color_header, sizeof(bmp_color_header));
82                 // Check if the pixel data is stored as BGRA and if the color space type is sRGB
83                 check_color_header(bmp_color_header);
84             }
85             else
86             {
87                 cerr << "Error! The file \"" << fname << "\" does not seem to contain bit mask information\n";
88                 throw runtime_error("Error! Unrecognized file format.");
89             }
90         }
91     }
```

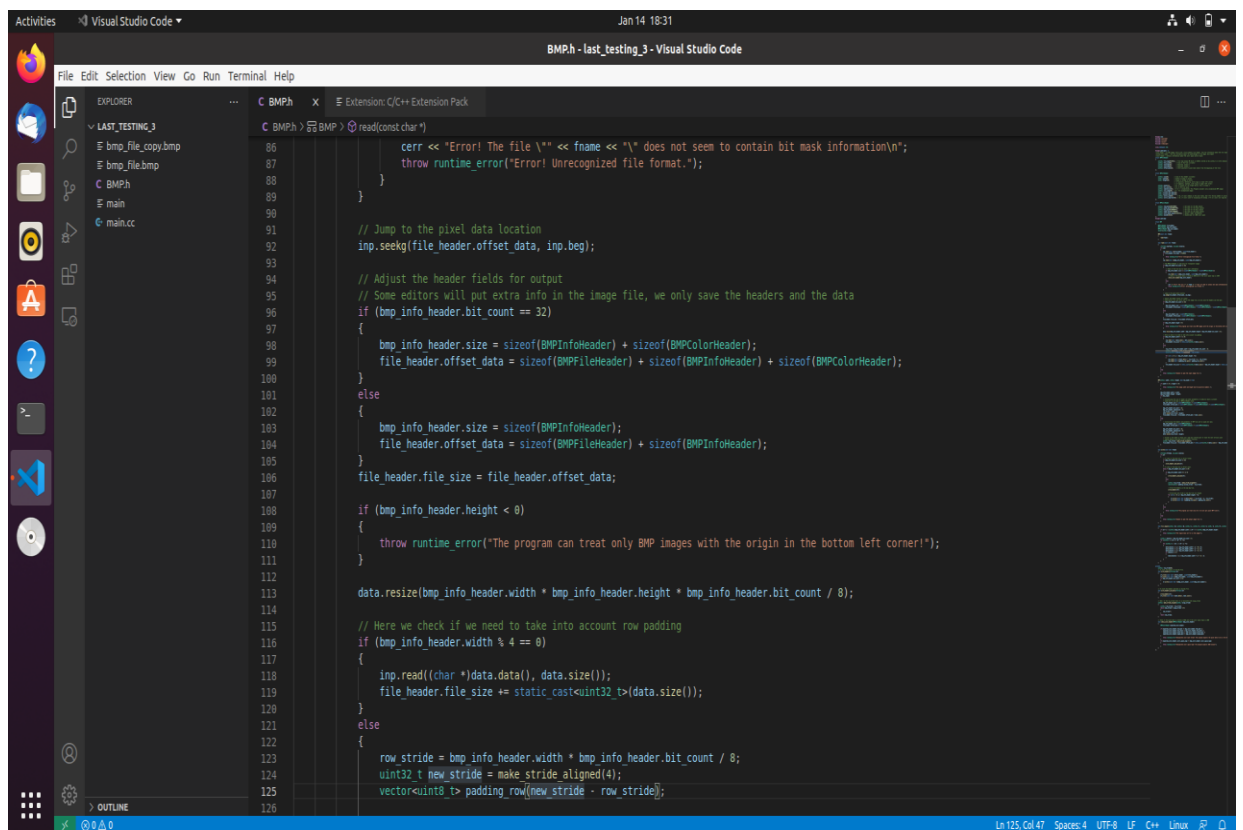


```
141 BMP(int32_t width, int32_t height, bool has_alpha = true)
142 {
143     if (width <= 0 || height <= 0)
144     {
145         throw runtime_error("The image width and height must be positive numbers.");
146     }
147
148     bmp_info_header.width = width;
149     bmp_info_header.height = height;
150     if (has_alpha)
151     {
152         // Initializing the size of header and other parameters if alpha bit mask is present.
153         // alpha bit is bit mask in BMPColorHeader block.
154         bmp_info_header.size = sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
155         file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
156
157         bmp_info_header.bit_count = 32;
158         bmp_info_header.compression = 3;
159         row_stride = width * 4;
160         data.resize(row_stride * height);
161         file_header.file_size = file_header.offset_data + data.size();
162     }
163     else
164     {
165         // Initializing the headers and parameters of BMP file with no alpha bit mask.
166         bmp_info_header.size = sizeof(BMPInfoHeader);
167         file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader);
168
169         bmp_info_header.bit_count = 24;
170         bmp_info_header.compression = 0;
171         row_stride = width * 3;
172         data.resize(row_stride * height);
173
174         // Stride is the number of bytes your code must iterate past to reach the next vertical pixel
175         // adding padding using make_stride_aligned() function.
176         uint32_t new_stride = make_stride_aligned(4);
177         file_header.file_size = file_header.offset_data + static_cast<uint32_t>(data.size()) + bmp_info_header.height * (new_stride - row_stride);
178     }
179 }
180
181 void write(const char *fname)
```

### 3. Function to Read Bitmap Image



```
49 struct BMP
50 {
51     BMPFileHeader file_header;
52     BMPInfoHeader bmp_info_header;
53     BMPColorHeader bmp_color_header;
54     vector<uint8_t> data;
55
56     BMP(const char *fname)
57     {
58         read(fname);
59     }
60
61     void read(const char *fname)
62     {
63         ifstream inp(fname, ios_base::binary);
64         if (!inp)
65         {
66             BMPFileHeader BMP::file_header
67             inp.read((char *) &file_header, sizeof(file_header));
68             if (file_header.file_type != 0x4D42)
69             {
70                 throw runtime_error("Error! Unrecognized file format.");
71             }
72             inp.read((char *) &bmp_info_header, sizeof(bmp_info_header));
73             // The BMPColorHeader is used only for transparent images
74             if (bmp_info_header.bit_count == 32)
75             {
76                 // Check if the file has bit mask color information
77                 if (bmp_info_header.size >= (sizeof(BMPInfoHeader) + sizeof(BMPColorHeader)))
78                 {
79                     inp.read((char *) &bmp_color_header, sizeof(bmp_color_header));
80                     // Check if the pixel data is stored as BGRA and if the color space type is sRGB
81                     check_color_header(bmp_color_header);
82                 }
83             }
84             else
85             {
86                 cerr << "Error! The file \"" << fname << "\" does not seem to contain bit mask information\n";
87                 throw runtime_error("Error! Unrecognized file format.");
88             }
89         }
90     }
```



```
86         cerr << "Error! The file \"" << fname << "\" does not seem to contain bit mask information\n";
87         throw runtime_error("Error! Unrecognized file format.");
88     }
89
90     // Jump to the pixel data location
91     inp.seekg(file_header.offset_data, inp.beg);
92
93     // Adjust the header fields for output
94     // Some editors will put extra info in the image file, we only save the headers and the data
95     if (bmp_info_header.bit_count == 32)
96     {
97         bmp_info_header.size = sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
98         file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader) + sizeof(BMPColorHeader);
99     }
100     else
101     {
102         bmp_info_header.size = sizeof(BMPInfoHeader);
103         file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader);
104     }
105     file_header.file_size = file_header.offset_data;
106
107     if (bmp_info_header.height < 0)
108     {
109         throw runtime_error("The program can treat only BMP images with the origin in the bottom left corner!");
110     }
111
112     data.resize(bmp_info_header.width * bmp_info_header.height * bmp_info_header.bit_count / 8);
113
114     // Here we check if we need to take into account row padding
115     if (bmp_info_header.width % 4 == 0)
116     {
117         inp.read((char *) data.data(), data.size());
118         file_header.file_size += static_cast<uint32_t>(data.size());
119     }
120     else
121     {
122         row_stride = bmp_info_header.width * bmp_info_header.bit_count / 8;
123         uint32_t new_stride = make_stride_aligned(4);
124         vector<uint8_t> padding_row(new_stride - row_stride);
125     }
```

This screenshot shows the Visual Studio Code editor with a C++ file named `BMP.h` open. The file is part of a project named `LAST_TESTING_3`. The code defines a `read(const char*)` function that reads a BMP file into a `data` vector. The function checks for valid BMP headers, handles row padding, and reads the image data. The status bar at the bottom indicates the current position is Line 105, Column 14.

```
100 }
101 else
102 {
103     bmp_info_header.size = sizeof(BMPInfoHeader);
104     file_header.offset_data = sizeof(BMPFileHeader) + sizeof(BMPInfoHeader);
105 }
106 file_header.file_size = file_header.offset_data;
107
108 if (bmp_info_header.height < 0)
109 {
110     throw runtime_error("The program can treat only BMP images with the origin in the bottom left corner!");
111 }
112
113 data.resize(bmp_info_header.width * bmp_info_header.height * bmp_info_header.bit_count / 8);
114
115 // Here we check if we need to take into account row padding
116 if (bmp_info_header.width % 4 == 0)
117 {
118     inp.read((char *)data.data(), data.size());
119     file_header.file_size += static_cast<uint32_t>(data.size());
120 }
121 else
122 {
123     row_stride = bmp_info_header.width * bmp_info_header.bit_count / 8;
124     uint32_t new_stride = make_stride_aligned(4);
125     vector<uint8_t> padding_row(new_stride - row_stride);
126
127     for (int y = 0; y < bmp_info_header.height; ++y)
128     {
129         inp.read((char *)data.data() + row_stride * y, row_stride);
130         inp.read((char *)padding_row.data(), padding_row.size());
131     }
132     file_header.file_size += static_cast<uint32_t>(data.size()) + bmp_info_header.height * static_cast<uint32_t>(padding_row.size());
133 }
134 }
135 else
136 {
137     throw runtime_error("Unable to open the input image file.");
138 }
139 }
140 }
```

## 4. Function to Write Bitmap Image

This screenshot shows the Visual Studio Code editor with the `BMP.h` file open, displaying the `write(const char *fname)` function. The function writes the `data` vector to a BMP file. It checks for valid BMP headers, handles row padding, and writes the image data. The status bar at the bottom indicates the current position is Line 196, Column 48.

```
188 void write(const char *fname)
189 {
190     ofstream of(fname, ios_base::binary);
191     if (of)
192     {
193         // To check if the bmp file is 32-bit format.
194         if (bmp_info_header.bit_count == 32)
195         {
196             write_headers_and_data(of);
197         }
198         // To check if the bmp file is 24-bit count.
199         else if (bmp_info_header.bit_count == 24)
200         {
201             if (bmp_info_header.width % 4 == 0)
202             {
203                 write_headers_and_data(of);
204             }
205             else
206             {
207                 uint32_t new_stride = make_stride_aligned(4);
208                 vector<uint8_t> padding_row(new_stride - row_stride);
209
210                 // Write the headers in the new bmp file.
211                 write_headers(of);
212
213                 // Write data and stride in new bmp file till height.
214                 for (int y = 0; y < bmp_info_header.height; ++y)
215                 {
216                     of.write((const char *)data.data() + row_stride * y, row_stride);
217                     of.write((const char *)padding_row.data(), padding_row.size());
218                 }
219             }
220         }
221     }
222 }
```

```
File Edit Selection View Go Run Terminal Help
BMP.h - last_testing_3 - Visual Studio Code
Jan 14 18:20

EXPLORER
LAST_TESTING_3
  bmp_file_copy.bmp
  bmp_file.bmp
  BMP.h
  main
  main.cc

C BMP.h
189 write_headers_and_data(of);
190 }
191 // To check if the bmp file is 24-bit count.
192 else if (bmp_info_header.bit_count == 24)
193 {
194     if (bmp_info_header.width % 4 == 0)
195     {
196         write_headers_and_data(of);
197     }
198     else
199     {
200         uint32_t new_stride = make_stride_aligned(4);
201         vector<uint8_t> padding_row(new_stride - row_stride);
202
203         // Write the headers in the new bmp file.
204         write_headers(of);
205
206         // Write data and stride in new bmp file till height.
207         for (int y = 0; y < bmp_info_header.height; ++y)
208         {
209             of.write((const char *) (data.data() + row_stride * y), row_stride);
210             of.write((const char *) padding_row.data(), padding_row.size());
211         }
212     }
213 }
214 else
215 {
216     throw runtime_error("The program can treat only 24 or 32 bits per pixel BMP files");
217 }
218 }
219 else
220 {
221     throw runtime_error("Unable to open the output image file.");
222 }
223 }
224 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
tony@stark:~/testing_task3/last_testing_3$
```

```
File Edit Selection View Go Run Terminal Help
BMP.h - last_testing_3 - Visual Studio Code
Jan 14 18:21

EXPLORER
LAST_TESTING_3
  bmp_file_copy.bmp
  bmp_file.bmp
  BMP.h
  main
  main.cc

C BMP.h
247 private:
248     uint32_t row_stride{0};
249     // To write the headers of new bmp file.
250     void write_headers(ofstream &of)
251     {
252         of.write((const char *) &file_header, sizeof(file_header));
253         of.write((const char *) &bmp_info_header, sizeof(bmp_info_header));
254         if (bmp_info_header.bit_count == 32)
255         {
256             of.write((const char *) &bmp_color_header, sizeof(bmp_color_header));
257         }
258     }
259
260     // To write the headers and data in new bmp file.
261     void write_headers_and_data(ofstream &of)
262     {
263         write_headers(of);
264         of.write((const char *) data.data(), data.size());
265     }
266
267     // Add 1 to the row_stride until it is divisible with align_stride
268     uint32_t make_stride_aligned(uint32_t align_stride)
269     {
270         uint32_t new_stride = row_stride;
271         while (new_stride % align_stride != 0)
272         {
273             new_stride++;
274         }
275         return new_stride;
276     }
277
278     // Check if the pixel data is stored as BGRA and if the color space type is sRGB
279     void check_color_header(BMPColorHeader &bmp_color_header)
280     {
281         BMPColorHeader converted_color_header;
282     }
283 }
284 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
tony@stark:~/testing_task3/last_testing_3$
```

This screenshot shows the Visual Studio Code editor with the `BMP.h` header file open. The Explorer sidebar on the left shows the project structure for `LAST_TESTING_3`, including `bmp_file_copy.bmp`, `bmp_file.bmp`, `BMP.h`, `main`, and `main.cc`. The main editor area displays the `BMP.h` file with the following code:

```
263 {
264     write_headers(of);
265     of.write((const char *)data.data(), data.size());
266 }
267
268 // Add 1 to the row stride until it is divisible with align_stride
269 uint32_t make_stride_aligned(uint32_t align_stride)
270 {
271     uint32_t new_stride = row_stride;
272     while (new_stride % align_stride != 0)
273     {
274         new_stride++;
275     }
276     return new_stride;
277 }
278
279 // Check if the pixel data is stored as BGRA and if the color space type is sRGB
280 void check_color_header(BMPColorHeader &bmp_color_header)
281 {
282     BMPColorHeader expected_color_header;
283
284     if (expected_color_header.red_mask != bmp_color_header.red_mask ||
285         expected_color_header.blue_mask != bmp_color_header.blue_mask ||
286         expected_color_header.green_mask != bmp_color_header.green_mask ||
287         expected_color_header.alpha_mask != bmp_color_header.alpha_mask)
288     {
289         throw runtime_error("Unexpected color mask format! The program expects the pixel data to be in the BGRA format");
290     }
291     if (expected_color_header.color_space_type != bmp_color_header.color_space_type)
292     {
293         throw runtime_error("Unexpected color space type! The program expects sRGB values");
294     }
295 }
296 };
297
```

The bottom status bar shows the file is at line 199, column 44, with 4 spaces, UTF-8 encoding, LF line endings, and C++ language.

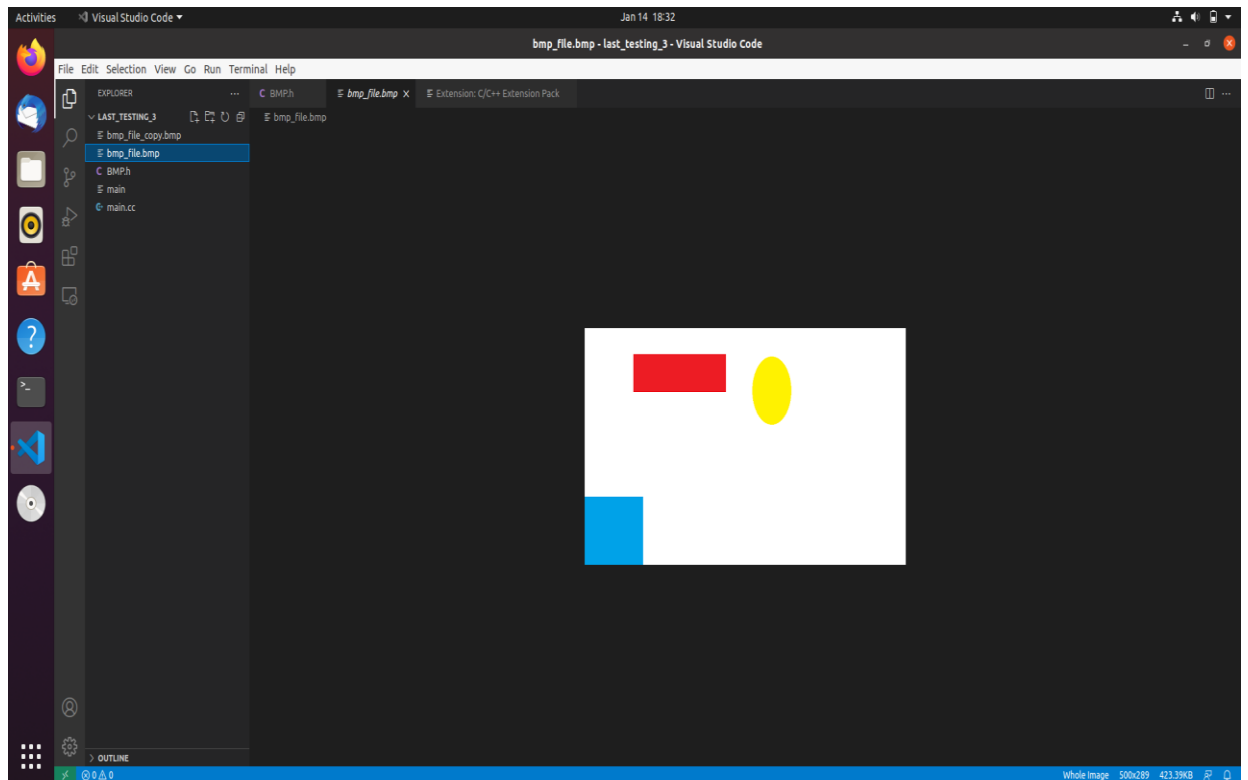
## 5. Main Function

This screenshot shows the Visual Studio Code editor with the `main.cc` source file open. The Explorer sidebar on the left shows the project structure for `LAST_TESTING_3`, including `bmp_file_copy.bmp`, `bmp_file.bmp`, `BMP.h`, `main`, and `main.cc`. The main editor area displays the `main.cc` file with the following code:

```
1 #include "BMP.h"
2 #include<iostream>
3
4 int main(){
5     // Read an image from disk, modify it and write it back
6     BMP bmp("bmp_file.bmp");
7     bmp.write("bmp_file_copy.bmp");
8     return 0;
9 }
10
11
```

The bottom status bar shows the file is at line 11, column 1, with 1 tab stop, UTF-8 encoding, LF line endings, and C++ language.

## 6. Input (Read) Bitmap Image



## 7. Output (Write) Bitmap Image

