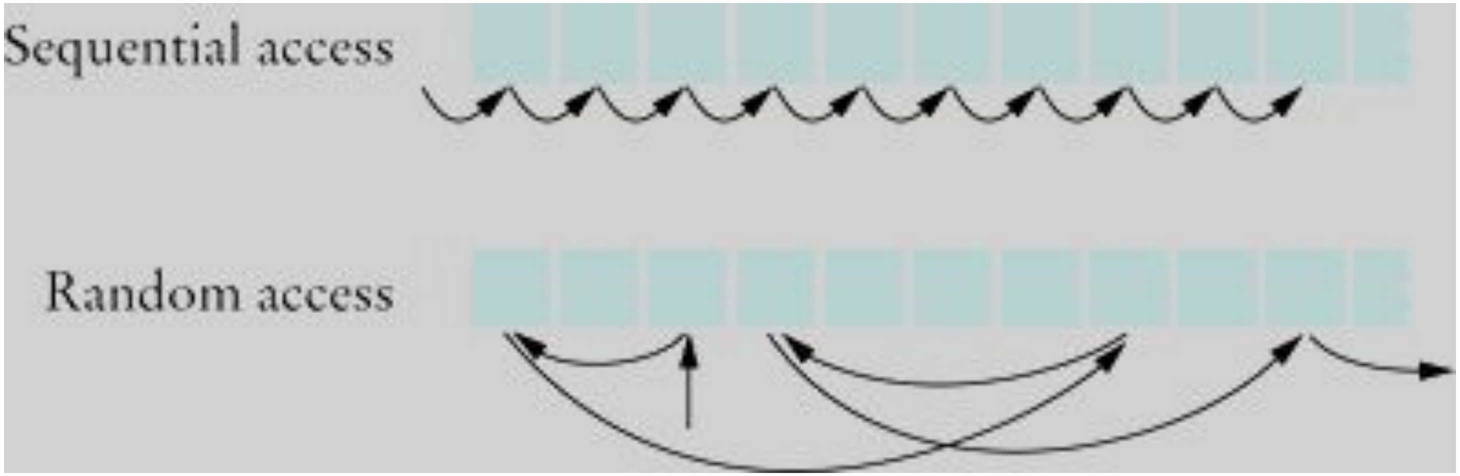
Topic 6

1. Reading and writing text files
2. Reading text input
3. Writing text output
4. Parsing and formatting strings
5. Command line arguments
6. Random access and binary files

Sequential Access and Random Access

Sequential Access: as we've been doing – one input at a time starting at the beginning

Random Access: you can go immediately to any point in the file, by specifying the location. File streams support this, but `cin/cout` do not.



Big C++ by Cay Horstmann

Random Access: put and get positions

The screen has a cursor to show the user where (s)he is typing.

Binary read/write files have two “cursor” positions:

1. the *put* position – where the next write will go.
2. the *get* position – where the next read will be.

```
//Functions: move the get and put positions to a
// given value, counted from the beginning of stream:
   strm.seekg(position);
   strm.seekp(position);

//Determine the current values of get and put positions
   position = strm.tellg();
   position = strm.tellp();
```

Whenever you write to the stream, the get position becomes undefined.

Call `seekg` when you switch back to reading.

Call `seekp` when you switch from reading to writing.

Binary Files

Many files, in particular those containing images and sounds, do not store information as text but as binary numbers.

The meanings and positions of these binary numbers must be known to process a binary file.

Binary Files vs. Text Files

Data is stored in files as sequences of bytes,
just as they are in the memory of the computer.

(Each byte has a value between 0 and 255.)

To store the word “CAB” as ASCII text takes four bytes:

`67 65 66 00`

The binary data in an image has a special
representation as a sequence of bytes
– but it’s still just a bunch of numbers.

Binary Files: Opening and Reading

To open a binary file for reading and writing, declare an fstream and use this version of the **`open`** method:

```
fstream strm;
strm.open("img.gif", ios::in | ios::out | ios::binary);
```

`ios::in` and **`ios::out`** allow us to read from and write into the same file.

You cannot use the >> operator to read a binary file. Instead, read a byte by calling `get()`

```
int input = strm.get();
    //returns a value between 0 and 255.
```

Big C++ by Cay Horstmann

Binary File Data

A “real” **`int`**, like **`1822327`**,
takes *four* bytes to store on most systems.

To read a “real” **`int`**, you will have to do *four* reads

– *and some arithmetic*.

(Or write a function to do this! – see the `get_int()` function in the following example…)

Processing Image Files

The BMP image file format is pretty simple. Other formats such as PNG, GIF, JPG are far more complex, as they compress the image to save space.

So we will use BMP file format in the following program. You can convert any of the above formats to BMP with most imaging programs.

In fact, we'll the use the most simple of the several versions of the BMP format:

the 24-bit true color format

Processing Image Files: The BMP File Format

The BMP file format for 24-bit true color format:

Each pixel's (picture element) color is represented in RGB form – Red, Green, and Blue amounts.

In the file, each pixel is represented as
a sequence of three bytes:

- a byte for the blue value (B)
- a byte for the green amount (G)
- a byte for the red amount (R)

Processing Image Files: Color Examples

Here are some RGB values stored in a BMP file
(you'll notice that it's really stored as BGR):

Cyan (a mixture of blue and green) is the bytes: **`255 255 0`**

Pure **red** is the values: **`0 0 255`** (no blue, no green, all red)

Medium gray is **`128 128 128`** (half of 255 for all three)

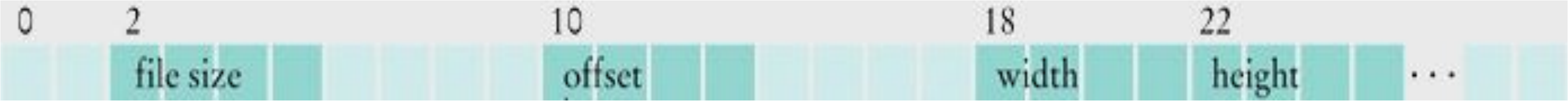
Big C++ by Cay Horstmann

Processing Image Files: The BMP Header

Most binary files start with some information about the contents called the *header*.

A BMP file header:

Position	Item
2	The size of this file in bytes
10	The start of the image data
18	The width of the image in pixels
22	The height of the image in pixels



Processing Image Files: Pixel Rows

The image itself is represented as a sequence of pixel rows (a scan line), starting with the bottom row in the image.

Each pixel row contains a sequence of BGR bytes.

The end of the row is padded with additional bytes so that the number of bytes in the row is divisible by 4.

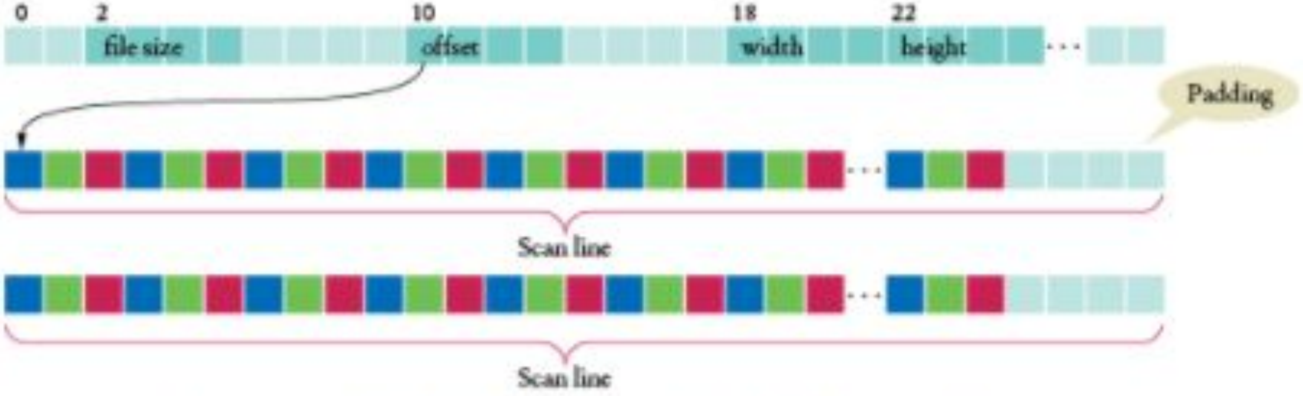


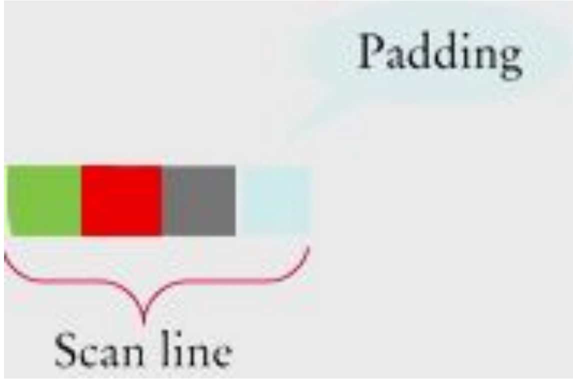
Figure 5 The BMP File Format for 24-bit True Color Images

Processing Image Files: Scan Line Example

For example,
if a row consisted of merely three pixels,
one cyan, one red, one medium gray,
there would three padding bytes.

The numbers would be:

```
255 255 0   0 0 255   128 128 128   x y z
```

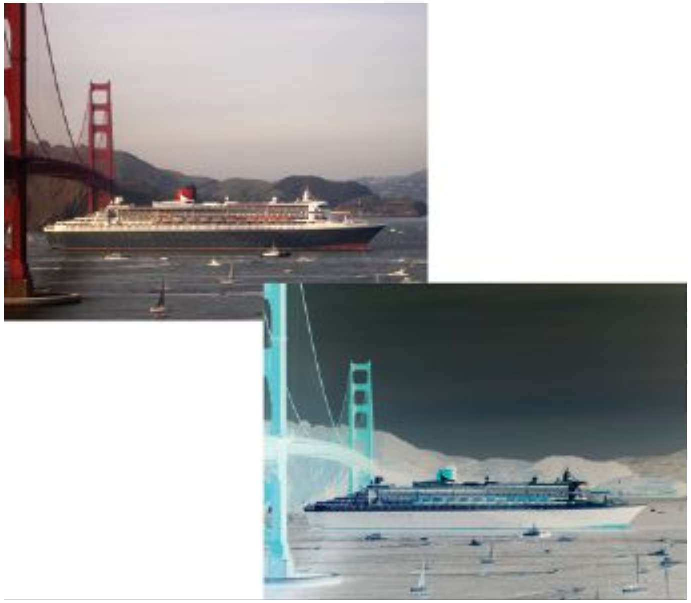


Big C++ by Cay Horstmann

Processing Image Files: the Negative Task

Now that you know all there is to know about BMP files for 24-bit true color images, we'll write code to create the negative of an input image file.

We create the negative of each pixel by subtracting the R, G, and B values from 255.



Program to Produce the Negative, Part 1

```
// sec06/imagemod.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
/**
 Processes a pixel by forming the negative.
 @param blue the blue value of the pixel
 @param green the green value of the pixel
 @param red the red value of the pixel
*/
void process(int& blue, int& green, int& red)
{
   blue = 255 - blue;
   green = 255 - green;
   red = 255 - red;
}
```

Program to Produce the Negative, Part 2

```
/**
 Gets an integer from a binary stream.
 @param stream the stream
 @param offset the offset at which to read the integer
 @return the integer starting at the given offset
*/
int get_int(fstream& stream, int offset)
{
   stream.seekg(offset);
   int result = 0;
   int base = 1;
   for (int i = 0; i < 4; i++)
   {
      result = result + stream.get() * base;
      base = base * 256;
   }
   return result;
}
```

Program to Produce the Negative, Part 3

```
int main()
{
   cout << "Please enter the file name: ";
   string filename;
   cin >> filename;
   fstream stream;

   // Open as a binary file
   stream.open(filename.c_str(),
      ios::in|ios::out|ios::binary);

   // Get the image dimensions
   int file_size = get_int(stream, 2);
   int start = get_int(stream, 10);
   int width = get_int(stream, 18);
   int height = get_int(stream, 22);
```

Program to Produce the Negative, Part 4

```
// Scan lines must occupy multiples of four bytes
int scanline_size = width * 3;
int padding = 0;
if (scanline_size % 4 != 0)
{
   padding = 4 - scanline_size % 4;
}
if (file_size != start +
   (scanline_size + padding) * height)
{
   cout << "Not a 24-bit true color image file.“
      << endl;
   return 1;
}
```

Program to Produce the Negative, Part 5

```
// Go to the start of the pixels
stream.seekg(start);

// For each scan line
for (int i = 0; i < height; i++)
{
   // For each pixel
   for (int j = 0; j < width; j++)
   {
      // Go to the start of the pixel
      int pos = stream.tellg();

      // Read the pixel
      int blue = stream.get();
      int green = stream.get();
      int red = stream.get();
```

Program to Produce the Negative, Part 6

```
         // Process the pixel
         process(blue, green, red);

         // Go back to the start of the pixel
         stream.seekp(pos);

         // Write the pixel
         stream.put(blue);
         stream.put(green);
         stream.put(red);
      }

      // Skip the padding
      stream.seekg(padding, ios::cur);
   }
   return 0;
}
```

Practice It: Negative Image Code

1. Modify the imagemod.cpp program to turn the green values of each pixel to red, the blue values to green, and the red values to blue for a psychedelic effect.

- Hint: you only need to change one function!

2. If a BMP file stores a 100 × 100 pixel image, with the image data starting at offset 64, what is the total file size?
   - 10,000 bytes
   - 30,000 bytes
   - 30,064 bytes
   - 40,000 bytes

Answer: 30,064 bytes. We need 3 × 100 bytes for each scan line. There is no padding because this number is divisible by 4. The total size = 3 × 100 × 100 + 64 = 30,064 bytes.

Chapter Summary, Part 1

Develop programs that read and write files.

- To read or write files, use variables of type `fstream, ifstream, or ofstream`.
- When opening a file stream, supply the name of the file stored on disk.

```
ifstream infile;
infile.open(“filename.txt”);
```

- Read from a file stream with the same operations that you use with `cin`.

```
int num;
infile >> num;
```

- Write to a file stream with the same operations that you use with `cout`.

```
ofstream out(“filename.txt”);
out << num;
```

- Always use a reference parameter for a stream function argument, because streams are modified as they are read or written

```
void process_name(ifstream& in_file, double& total)
```

Chapter Summary, Part 2

Be able to process text in files.

- When reading a `string` with the >> operator, the white space between words is consumed.
- You can `get` individual characters from a stream and `unget` the last one.

```
in_file.get(ch);
if (isdigit(ch))
{
   in_file.unget(); // Put the digit back
   data >> n; // Read integer starting with ch
}
```

- You can read a line of input with `getline()` and then process it further. There are 2 flavors of the function:

```
string line;
ifstream in_file("myfile.txt");
getline(in_file, line);

char cstring[100];
in_file.getline(cstring, 99);
```

Chapter Summary, Part 3

Write programs that neatly format their output.

- `#include <iomanip>`
- Use the `setw` manipulator to set the width of the next output.

```
out << setw(40) << left << country << setw(15) << right << density
<< endl;
```

- Use the `fixed` and `setprecision` manipulators to format floating-point numbers with a fixed number of digits after the decimal point.

```
out_file << fixed << x << endl << setprecision(2) << x;
```

Convert between strings and numbers.

- Use an `istringstream` to convert the numbers inside a `string` to integers or floating-point numbers.

```
istringstream strm;
strm.str("January 24, 1973");
string month, comma;
int day, year;
strm >> month >> day >> comma >> year;
```

- Use an `ostringstream` to convert numeric values to `string`s.

Big C++ by Cay Horstmann

Chapter Summary, Part 4

Process the command line arguments of a C++ program.

- Programs that start from the command line can receive the name of the program and the command line arguments in the `main` function.

```
int main(int argc, char* argv[])
```

Develop programs that read and write binary files.

- Open the file with:

```
fstream strm;
strm.open("img.gif", ios::in | ios::out | ios::binary);
```

- You can access any position in a random access file by moving the file pointer prior to a read or write operation.

```
strm.seekg(position); //read = get
char c = strm.get();
strm.seekp(position); // write = put
position = strm.tellg();
position = strm.tellp();
```