TEXT FILES and BINARY FILES
RANDOM/DIRECT ACCESS TO BINARY FILES
STRINGSTREAMS

Examples

JAS

```
// 01
// WHY YOU MUST TRY TO READ FROM FILE, BEFORE TESTING EOF (end of file)

#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
  ifstream f_in;
  // TEST THE PROGRAM USING THE FOLLOWING TEXT FILES (ONE FILE IN EACH RUN)
  // strings_01.txt (empty file)
  // strings_02.txt (no newline after last line)
  // strings_03.txt (newline after last line)
  f_in.open("strings_01.txt");
  if (!f_in.is_open())
  {
    cerr << "File not found!\n";
    exit(1);
  }

  string s = "-----"; // just to initialize string with a non-empty string
  while (!f_in.eof()) // SHOULDN'T DO THIS TEST BEFORE TRYING TO READ FROM THE FILE
  {
    f_in >> s;  // try   f_in>>s;    and   getline(f_in,s);
    cout << '|' << s << '|' << endl; // the vertical bars are just to show the string limits, and
                                     // enhance the presence of empty strings
  }

  f_in.close();
  return 0;
}


//=================================================================================
// 02
// THE CORRECT WAY: FIRST, TRY TO READ, THEN, TEST FOR EOF

#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
  ifstream f_in;
  // strings_01.txt (empty file)
  // strings_02.txt (no newline after last line)
  // strings_03.txt (newline after last line)
  f_in.open("strings_01.txt");
  if (!f_in.is_open())
  {
    cerr << "File not found!\n";
    exit(1);
  }

  string s = "-----"; // just to initialize string with a non-empty string
  // THE CORRECT WAY: FIRST, TRY TO READ, THEN TEST FOR EOF
  // BUT... LOOK AT THE RESULT WITH strings_02.txt, when you use f_in >> s; or getline(f_in,s); !!!
  getline(f_in, s);  // try   f_in>>s;    and   getline(f_in,s);
  while (!f_in.eof())
  {
    cout << '|' << s << '|' << endl;
    getline(f_in, s);  // try   f_in>>s;    and   getline(f_in,s);
  }

  f_in.close();
  return 0;
}

//=================================================================================
// 03
// ANOTHER CORRECT WAY: while (getline(f_in, s)) OR while (f_in >> s))

#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
  ifstream f_in;
  // strings_01.txt (empty file)
  // strings_02.txt (no newline after last line)
  // strings_03.txt (newline after last line)
  f_in.open("strings_01.txt");
  if (!f_in.is_open())
  {
    cerr << "File not found!\n";
    exit(1);
  }

  string s = "-----";
  // ALSO A CORRECT WAY:
  // COMPARE THE RESULTS WITH THOSE OF PROGRAM 02, IN THE CASE OF string_02.txt

  while (getline(f_in, s)) // TRY: while (f_in >> s)   AND  while (getline(f_in,s))
  {
    cout << "|" << s << "|" << endl;
  }
  f_in.close();
  return 0;
}


//==================================================================================
// 04
// READING FROM A TEXT FILE  WRITING TO ANOTHER
// (no output to the screen)

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

int main()
{
  ifstream f_in("numbers_01.txt"); // USING THE CONSTRUCTOR TO TRY TO OPEN A FILE
  ofstream f_out("numbers_01_sum.txt");

  if (!f_in.is_open())
  {
    cerr << "File not found!\n";
    exit(1);
  }

  double n, sum = 0;
  while (f_in >> n)
  {
    f_out << fixed << setprecision(3);
    f_out << setw(10) << n << endl; // 10 & 3 -> SHOULD BE NAMED CONSTANTS ...
    sum = sum + n;
  }
  f_out << "sum = " << setw(10) << sum << endl;

  f_in.close();
  f_out.close();
  return 0;
}

//================================================================================
```

```cpp
// 05
// USING STREAMS AS FUNCTION PARAMETERS
// (no output to the screen)

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

void processNumbers(ifstream &f_in, ofstream &f_out)
// NOTE: try istream AND ostream AND replace the call with processNumbers(cin, f_out);
{
  double n, sum = 0;
  f_in >> n;
  while (!f_in.eof())
  {
    f_out << fixed << setprecision(3);
    f_out << setw(10) << n << endl;
    if (f_out.fail()) cerr << "failed\n";
    sum = sum + n;
    f_in >> n;
  }
  f_out << "sum = " << setw(10) << sum << endl;
}
//--------------------------------------------------------------------------------
int main()
{
  ifstream f_in("numbers_01.txt"); // USING THE CONSTRUCTOR TO TRY TO OPEN A FILE
  ofstream f_out("numbers_01_sum.txt");

  if (!f_in.is_open())
  {
    cerr << "File not found!\n";
    exit(1);
  }

  processNumbers(f_in, f_out);

  f_in.close();
  f_out.close();
  return 0;
}


//==================================================================================
// 06
// READING A TEXT FILE IN BINARY FORMAT
// OUTPUT CHAR BY CHAR, INCLUDING CARRIAGE RETURN & LINE FEED CHARACTERS
// (no output to the screen)

#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  ifstream f_in("strings_03.txt", ios::binary);
  if (!f_in.is_open())
  {
    cerr << "File not found!\n";
    exit(1);
  }

  char c;
  c = f_in.get(); // overloaded function -
http://cplusplus.com/reference/istream/istream/get/
  while (!f_in.eof())
  {
    cout << '|' << c << '|' << endl; // TRY: cout << '|' << (int) c << '|' << endl;
    // NOTE THE OUTPUT: one of the '|' disapears !!! - effect of CARRIAGE RETURN
    c = f_in.get();
  }

  f_in.close();
  return 0;
}

//=================================================================================
// 07
// WRITING A BINARY FILE, CONTAINING INTEGER NUMBERS

#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  ofstream f("numbers.dat", ios::binary);

  for (int i = 1; i <= 3; i++)
    // for (int i = 65+32*256+32*256*256+32*256*256*256, count=0; count <= 3; i++,
count++)
    f.write((char*)&i, sizeof(int));

  f.close();  // TRY TO SEE THE FILE CONTENTS USING NOTEPAD
  return 0;
}

// ===========================================================================

// 08
// READING A BINARY FILE, CONTAINING INTEGER NUMBERS

#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  ifstream f;

  f.open("numbers.dat", ios::binary); // SHOULD TEST IF IT IS OPEN …

  int i;
  while(f.read((char*)&i, sizeof(int)))
  {
    cout << i << endl;
  }

  f.close();
  return 0;
}

// ===========================================================================
```

```cpp
// 09
// READING A BINARY FILE, BYTE BY BYTE

#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>

using namespace std;

int main()
{
  ifstream f("numbers.dat");

  char c;
  c = f.get();
  while (!f.eof())
  {
    cout << '|' << (int)c << '|' << endl; // cout << '|' << (int) c << '|' << endl;
    c = f.get();
  }

  //// OR

  //while ((c = f.get()) != EOF)
  //{
  //  cout << '|' << (int)c << '|' << endl; // cout << '|' << (int) c << '|' << endl;
  //}

  f.close();
  return 0;
}


// ==========================================================================
// 10
// WRITING A BINARY FILE, OF PERSON RECORDS (STRUCT'S)

#include <iostream>
#include <cstring>
#include <fstream>
#include <vector>

using namespace std;

struct Person
{
  char name[10]; // NOTE: C-STRING
  unsigned int age;
};

int main()
{
  ofstream f("persons.dat", ios::binary);

  vector<Person> vecp={{ "Maria", 16 }, { "Ze", 12 }, { "Rita", 31 }, { "Manel", 31 }};

  for (auto p:vecp)
    f.write((char*)&p, sizeof(Person));

  f.close();
  return 0;
}
//==============================================================================
```

```cpp
// 11
// READING A BINARY FILE, OF PERSON RECORDS (STRUCT'S)

#include <iostream>
#include <cstring>
#include <fstream>

using namespace std;

struct Person
{
  char name[10];
  unsigned int age;
};

int main()
{
  ifstream f("persons.dat", ios::binary);

  Person p;

  while(f.read((char*)&p, sizeof(Person)))
    cout << p.name << " - " << p.age << endl;

  f.close();
  return 0;
}

//=====================================================================================
// 12
// READING A BINARY FILE, OF PERSON RECORDS (STRUCT'S)
// DIRECT (RANDOM) ACCESS TO A RECORD

#include <iostream>
#include <cstring>
#include <fstream>

using namespace std;

struct Person
{
  char name[10];
  unsigned int age;
};

int main()
{
  ifstream f("persons.dat", ios::binary);

  Person p;

  int n;
  cout << "Record number to read (0..3) ? "; //TODO: modify program 10, and the following
programs, so that number of records is stored at the top of the file

  while (cin >> n)
  {
    f.seekg(n*sizeof(Person));
    f.read((char*)&p, sizeof(Person));
    cout << p.name << " - " << p.age << endl;
    cout << "Record number to read (0..3) ? ";
  }

  f.close();
  return 0;
}

//=========================================================================
```

```cpp
// 13
// READING A BINARY FILE, OF PERSON RECORDS (STRUCT'S)
// DIRECT (RANDOM) ACCESS TO A RECORD
// MODIFYING THE CONTENTS OF A RECORD

#include <iostream>
#include <cstring>
#include <fstream>
#include <iomanip>
using namespace std;
//-------------------------------------------------------------------------------------
struct Person
{
  char name[10];
  unsigned int age;
};
//-------------------------------------------------------------------------------------
void showAllFileContents(istream &f)
{
  Person p;
  cout << "All contents:\n";
  f.clear();
  f.seekg(0); // RESET FILE POINTER TO BEGIN OF FILE - ALTERNATIVES: f.seekg(0,
ios::beg); or f.seekg(f.beg);
  int count = 0;
  while (f.read((char*)&p, sizeof(Person)))  // DID YOU SEE THE RESULT OF THIS CYCLE ?!!!
    cout << count++ << " - " << setw(10) << p.name << " / " << setw(2) << p.age << endl;
}
//-------------------------------------------------------------------------------------
int main()
{
  fstream f("persons.dat", ios::binary | ios::in | ios::out);

  int n;
  showAllFileContents(f);
  cout << "Record number to modify (0..3 / CTRL-Z) ? "; // NOTE: program 10 created a
file with 4 records
  while (cin >> n)  // WHAT HAPPENS IF YOU INPUT A NUMBER GREATER THAN 3 ?
  {
    Person p;

    f.clear(); // COMMENT AND SEE WHAT HAPPENS
    f.seekg(n * sizeof(Person));
    f.read((char*)&p, sizeof(Person));

    cout << "new name ? "; cin >> p.name;
    cout << "new  age ? "; cin >> p.age;

    f.clear(); // COMMENT AND SEE WHAT HAPPENS
    f.seekp(n * sizeof(Person));
    f.write((char*)&p, sizeof(Person));

    showAllFileContents(f);

    cout << "Record number to modify (0..3 / CTRL-Z) ? ";
  }

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

```
// 14
// SHOW CONTENTS OF EUROMILLION BETS FILE, LINE BY LINE
// EXAMPLE OF FILE CONTENTS (see below)

#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
  ifstream f("eurom_bets.txt");

  string bet;

  while (getline(f, bet))
    cout << bet << endl;

  f.close();
  return 0;
}

//============================================================================
```

Contents of "eurom_bets.txt":

```
13 18 29 39 50 - 5 12
1 8 12 21 23 35 50 - 6 8
3 13 20 39 49 - 2 9
9 18 19 25 30 - 11 12
```

```cpp
// 15
// STRINGSTREAMS – DECOMPOSING EUROMILLION BETS FILE, LINE BY LINE

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

using namespace std;

int main()
{
  ifstream f("eurom_bets.txt");  // should test if it is open …

  string bet;

  f.seekg(0,ios::beg);
  while (getline(f, bet))
  {
    // Separate the "bet" string into 2 strings: "numbers" and "stars"
    size_t posHifen = bet.find('-');
    string numbers = bet.substr(0, posHifen);
    string stars = bet.substr(posHifen+1);
    cout << bet << endl;
    cout << "|" << numbers << "|" << stars << "|" << endl;

    int n;

    // Decompose "numbers" string into integer values
    istringstream iss(numbers);
    cout << "numbers:\n";
    while (iss >> n)
      cout << n << endl;
    cout << "---\n";

    // Decompose "stars" string into integer values
    cout << "stars:\n";
    //iss.clear();
    iss.str(stars); // NOTE THIS
    while (iss >> n)
      cout << n << endl;
    cout << "=====\n";
  }

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