**Input/output with files**

C++ provides the following classes to perform output and input of characters to/from files:

* [**ofstream**](http://www.cplusplus.com/ofstream)**:** Stream class to write on files
* [**ifstream**](http://www.cplusplus.com/ifstream)**:** Stream class to read from files
* [**fstream**](http://www.cplusplus.com/fstream)**:** Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes istream and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files. Let's see an example:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 | // basic file operations  #include <iostream>  #include <fstream>  using namespace std;  int main () {  ofstream myfile;  myfile.open ("example.txt");  myfile << "Writing this to a file.\n";  myfile.close();  return 0;  } | [file example.txt]  Writing this to a file. | [Edit & Run](http://www.cplusplus.com/doc/tutorial/files/) |

This code creates a file called example.txt and inserts a sentence into it in the same way we are used to do with cout, but using the file stream myfile instead.  
  
But let's go step by step:

**Open a file**

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a *stream* (i.e., an object of one of these classes; in the previous example, this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.  
  
In order to open a file with a stream object we use its member function open:  
  
open (filename, mode);  
  
Where filename is a string representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

|  |  |
| --- | --- |
| ios::in | Open for input operations. |
| ios::out | Open for output operations. |
| ios::binary | Open in binary mode. |
| ios::ate | Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file. |
| ios::app | All output operations are performed at the end of the file, appending the content to the current content of the file. |
| ios::trunc | If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one. |

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open:

|  |  |  |
| --- | --- | --- |
| 1 2 | ofstream myfile;  myfile.open ("example.bin", ios::out | ios::app | ios::binary); |  |

Each of the open member functions of classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

|  |  |
| --- | --- |
| **class** | **default mode parameter** |
| ofstream | ios::out |
| ifstream | ios::in |
| fstream | ios::in | ios::out |

For ifstream and ofstream classes, ios::in and ios::out are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the open member function (the flags are combined).  
  
For fstream, the default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.  
  
File streams opened in *binary mode* perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).  
  
Since the first task that is performed on a file stream is generally to open a file, these three classes include a constructor that automatically calls the open member function and has the exact same parameters as this member. Therefore, we could also have declared the previous myfile object and conduct the same opening operation in our previous example by writing:

|  |  |  |
| --- | --- | --- |
|  | ofstream myfile ("example.bin", ios::out | ios::app | ios::binary); |  |

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.  
  
To check if a file stream was successful opening a file, you can do it by calling to member is\_open. This member function returns a bool value of true in the case that indeed the stream object is associated with an open file, or false otherwise:

|  |  |  |
| --- | --- | --- |
|  | if (myfile.is\_open()) { /\* ok, proceed with output \*/ } |  |

**Closing a file**

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function close. This member function takes flushes the associated buffers and closes the file:

|  |  |  |
| --- | --- | --- |
|  | myfile.close(); |  |

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.  
  
In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function close.

**Text files**

Text file streams are those where the ios::binary flag is not included in their opening mode. These files are designed to store text and thus all values that are input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.  
  
Writing operations on text files are performed in the same way we operated with cout:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | // writing on a text file  #include <iostream>  #include <fstream>  using namespace std;  int main () {  ofstream myfile ("example.txt");  if (myfile.is\_open())  {  myfile << "This is a line.\n";  myfile << "This is another line.\n";  myfile.close();  }  else cout << "Unable to open file";  return 0;  } | [file example.txt]  This is a line.  This is another line. | [Edit & Run](http://www.cplusplus.com/doc/tutorial/files/) |

Reading from a file can also be performed in the same way that we did with cin:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | // reading a text file  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  int main () {  string line;  ifstream myfile ("example.txt");  if (myfile.is\_open())  {  while ( getline (myfile,line) )  {  cout << line << '\n';  }  myfile.close();  }  else cout << "Unable to open file";  return 0;  } | This is a line.  This is another line. | [Edit & Run](http://www.cplusplus.com/doc/tutorial/files/) |

This last example reads a text file and prints out its content on the screen. We have created a while loop that reads the file line by line, using [getline](http://www.cplusplus.com/getline). The value returned by [getline](http://www.cplusplus.com/getline) is a reference to the stream object itself, which when evaluated as a boolean expression (as in this while-loop) is true if the stream is ready for more operations, and false if either the end of the file has been reached or if some other error occurred.

**Checking state flags**

The following member functions exist to check for specific states of a stream (all of them return a bool value):

bad()

Returns true if a reading or writing operation fails. For example, in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

fail()

Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

eof()

Returns true if a file open for reading has reached the end.

good()

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true. Note that good and bad are not exact opposites (good checks more state flags at once).

The member function clear() can be used to reset the state flags.

**get and put stream positioning**

All i/o streams objects keep internally -at least- one internal position:  
  
ifstream, like istream, keeps an internal *get position* with the location of the element to be read in the next input operation.  
  
ofstream, like ostream, keeps an internal *put position* with the location where the next element has to be written.  
  
Finally, fstream, keeps both, the *get* and the *put position*, like iostream.  
  
These internal stream positions point to the locations within the stream where the next reading or writing operation is performed. These positions can be observed and modified using the following member functions:

**tellg() and tellp()**

These two member functions with no parameters return a value of the member type streampos, which is a type representing the current *get position* (in the case of tellg) or the *put position* (in the case of tellp).

**seekg() and seekp()**

These functions allow to change the location of the *get* and *put positions*. Both functions are overloaded with two different prototypes. The first form is:  
  
seekg ( position );  
seekp ( position );  
  
Using this prototype, the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is streampos, which is the same type as returned by functions tellg and tellp.  
  
The other form for these functions is:  
  
seekg ( offset, direction );  
seekp ( offset, direction );  
  
Using this prototype, the *get* or *put position* is set to an offset value relative to some specific point determined by the parameter direction. offset is of type streamoff. And direction is of type seekdir, which is an *enumerated type* that determines the point from where offset is counted from, and that can take any of the following values:

|  |  |
| --- | --- |
| ios::beg | offset counted from the beginning of the stream |
| ios::cur | offset counted from the current position |
| ios::end | offset counted from the end of the stream |

The following example uses the member functions we have just seen to obtain the size of a file:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | // obtaining file size  #include <iostream>  #include <fstream>  using namespace std;  int main () {  streampos begin,end;  ifstream myfile ("example.bin", ios::binary);  begin = myfile.tellg();  myfile.seekg (0, ios::end);  end = myfile.tellg();  myfile.close();  cout << "size is: " << (end-begin) << " bytes.\n";  return 0;  } | size is: 40 bytes. | [Edit & Run](http://www.cplusplus.com/doc/tutorial/files/) |

Notice the type we have used for variables begin and end:

|  |  |  |
| --- | --- | --- |
|  | streampos size; |  |

streampos is a specific type used for buffer and file positioning and is the type returned by file.tellg(). Values of this type can safely be subtracted from other values of the same type, and can also be converted to an integer type large enough to contain the size of the file.  
  
These stream positioning functions use two particular types: streampos and streamoff. These types are also defined as member types of the stream class:

|  |  |  |
| --- | --- | --- |
| **Type** | **Member type** | **Description** |
| [streampos](http://www.cplusplus.com/streampos) | [ios::pos\_type](http://www.cplusplus.com/ios#types) | Defined as [fpos<mbstate\_t>](http://www.cplusplus.com/fpos). It can be converted to/from [streamoff](http://www.cplusplus.com/streamoff) and can be added or subtracted values of these types. |
| [streamoff](http://www.cplusplus.com/streamoff) | [ios::off\_type](http://www.cplusplus.com/ios#types) | It is an alias of one of the fundamental integral types (such as int or long long). |

Each of the member types above is an alias of its non-member equivalent (they are the exact same type). It does not matter which one is used. The member types are more generic, because they are the same on all stream objects (even on streams using exotic types of characters), but the non-member types are widely used in existing code for historical reasons.

**Binary files**

For binary files, reading and writing data with the extraction and insertion operators (<< and >>) and functions likegetline is not efficient, since we do not need to format any data and data is likely not formatted in lines.  
  
File streams include two member functions specifically designed to read and write binary data sequentially: write andread. The first one (write) is a member function of ostream (inherited by ofstream). And read is a member function ofistream (inherited by ifstream). Objects of class fstream have both. Their prototypes are:  
  
write ( memory\_block, size );  
read ( memory\_block, size );  
  
Where memory\_block is of type char\* (pointer to char), and represents the address of an array of bytes where the read data elements are stored or from where the data elements to be written are taken. The size parameter is an integer value that specifies the number of characters to be read or written from/to the memory block.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 | // reading an entire binary file  #include <iostream>  #include <fstream>  using namespace std;  int main () {  streampos size;  char \* memblock;  ifstream file ("example.bin", ios::in|ios::binary|ios::ate);  if (file.is\_open())  {  size = file.tellg();  memblock = new char [size];  file.seekg (0, ios::beg);  file.read (memblock, size);  file.close();  cout << "the entire file content is in memory";  delete[] memblock;  }  else cout << "Unable to open file";  return 0;  } | the entire file content is in memory | [Edit & Run](http://www.cplusplus.com/doc/tutorial/files/) |

In this example, the entire file is read and stored in a memory block. Let's examine how this is done:  
  
First, the file is open with the ios::ate flag, which means that the get pointer will be positioned at the end of the file. This way, when we call to member tellg(), we will directly obtain the size of the file.  
  
Once we have obtained the size of the file, we request the allocation of a memory block large enough to hold the entire file:

|  |  |  |
| --- | --- | --- |
|  | memblock = new char[size]; |  |

Right after that, we proceed to set the *get position* at the beginning of the file (remember that we opened the file with this pointer at the end), then we read the entire file, and finally close it:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | file.seekg (0, ios::beg);  file.read (memblock, size);  file.close(); |  |

At this point we could operate with the data obtained from the file. But our program simply announces that the content of the file is in memory and then finishes.

**Buffers and Synchronization**

When we operate with file streams, these are associated to an internal buffer object of type streambuf. This buffer object may represent a memory block that acts as an intermediary between the stream and the physical file. For example, with an ofstream, each time the member function put (which writes a single character) is called, the character may be inserted in this intermediate buffer instead of being written directly to the physical file with which the stream is associated.  
  
The operating system may also define other layers of buffering for reading and writing to files.  
  
When the buffer is flushed, all the data contained in it is written to the physical medium (if it is an output stream). This process is called *synchronization* and takes place under any of the following circumstances:

* **When the file is closed:** before closing a file, all buffers that have not yet been flushed are synchronized and all pending data is written or read to the physical medium.
* **When the buffer is full:** Buffers have a certain size. When the buffer is full it is automatically synchronized.
* **Explicitly, with manipulators:** When certain manipulators are used on streams, an explicit synchronization takes place. These manipulators are: [flush](http://www.cplusplus.com/flush) and [endl](http://www.cplusplus.com/endl).
* **Explicitly, with member function sync():** Calling the stream's member function sync() causes an immediate synchronization. This function returns an int value equal to -1 if the stream has no associated buffer or in case of failure. Otherwise (if the stream buffer was successfully synchronized) it returns 0.

#include <iostream>

#include <math.h>

#include <time.h>

#include <string.h>

#include <windows.h>

#include <climits>

<climit> các giới hạn của c

#include <iostream>

The size of short is 2 bytes  
The size of int is 4 bytes  
The size of long is 4 bytes  
The size of long long is 8 bytes  
The size of char is 1 bytes  
The size of bool is 4 bytes  
The size of float is 4 bytes  
The size of double is 8 bytes  
The size of long double is 8 bytes  
  
  
The size of short is 2 bytes  
The size of int is 4 bytes  
The size of long is 4 bytes  
The size of long long is 8 bytes  
The size of char is 1 bytes  
The size of bool is 1 bytes  
The size of float is 4 bytes  
The size of double is 8 bytes  
The size of long double is 12 bytes

#include <iostream>  
2 #include <climits> // max & min size of integer types  
3 #include <cfloat> // max & min size of real types  
4 using namespace std;  
56  
/\*\*  
7 \* Print out the extreme values of various integer types.  
8 \*/  
9  
10 int main() {  
11 cout << "The maximum size of a short is " << SHRT\_MAX << endl;  
12 cout << "The minimum size of a short is " << SHRT\_MIN << endl;  
13  
14 cout << "The maximum size of an int is " << INT\_MAX << endl;  
15 cout << "The minimum size of an int is " << INT\_MIN << endl;  
16  
17 cout << "The maximum size of a long is " << LONG\_MAX << endl;  
18 cout << "The minimum size of a long is " << LONG\_MIN << endl;  
19  
20 // long long values might not exist on some computers  
21 cout << "The maximum size of a long long is " << LLONG\_MAX << endl;  
22 cout << "The minimum size of a long long is " << LLONG\_MIN << endl;  
23  
24  
25 cout << "The minimum positive value of a float is "  
26 << FLT\_MIN << endl;  
27 cout << "The minimum epsilon value of a float is "  
28 << FLT\_EPSILON << endl;  
29 cout << "The maximum value of a float is "  
30 << FLT\_MAX << endl;  
31  
32 cout << "The minimum positive value of a double is "  
33 << DBL\_MIN<< endl;  
34 cout << "The minimum epsilon value of a double is "  
35 << DBL\_EPSILON << endl;  
36 cout << "The maximum value of a double is "  
37 << DBL\_MAX << endl;  
38  
39 // long double might not be defined on some systems  
40 cout << "The minimum positive value of a long double is "  
41 << LDBL\_MIN<< endl;  
42 cout << "The minimum epsilon value of a long double is "  
43 << LDBL\_EPSILON << endl;  
44 cout << "The maximum value of a long double is "  
45 << LDBL\_MAX << endl;  
46  
47 return 0;  
48 }  
18 *C++ for Mathematici*

The maximum size of a short is 32767  
The minimum size of a short is -32768  
The maximum size of an int is 2147483647  
The minimum size of an int is -2147483648  
The maximum size of a long is 2147483647  
The minimum size of a long is -2147483648  
The maximum size of a long long is 9223372036854775807  
The minimum size of a long long is -9223372036854775808  
The minimum positive value of a float is 1.17549e-38  
The minimum epsilon value of a float is 1.19209e-07  
The maximum value of a float is 3.40282e+38  
The minimum positive value of a double is 2.22507e-308  
The minimum epsilon value of a double is 2.22045e-16  
The maximum value of a double is 1.79769e+308  
The minimum positive value of a long double is 2.22507e-308  
The minimum epsilon value of a long double is 2.22045e-16  
The maximum value of a long double is 1.79769e+308  
✝

Exit(i) trong <stdlib.h>

template <typename T>

ostream& operator<<(ostream& ostr, const Grid<T>& grid)

{

for (int i = 0; i < grid.mHeight; i++) {

for (int j = 0; j < grid.mWidth; j++) {

// Add a tab between each element of a row.

ostr << grid.mCells[j][i] << “\t”;

}

ostr << std::endl; // Add a newline between each row.

}

return (ostr);

}

//Grid.h

#include <iostream>

using std::ostream;

// Forward declare Grid template.

template <typename T> class Grid;

// Prototype for templatized operator<<.

template<typename T>

ostream& operator<<(ostream& ostr, const Grid<T>& grid);

template <typename T>

class Grid

{

public:

// Omitted for brevity

friend ostream& operator<< <T>(ostream& ostr, const Grid<T>& grid);

// Omitted for brevity

};

long long

INT\_MIN

Don’t use % for negative numbers

#include <iostream>

#include <cmath>

using namespace std;

main() {

double e = exp(1.);

double pi = M\_PI;

cout << "e to the pi is " << exp(pi) << endl;

cout << "pi to the e is " << pow(pi,e) << endl;

cout << endl << " Fini !"; cin.get();

}

...

#include <complex>

using namespace std;

typedef complex<long> icomp;

typedef complex<double> dcomp;

main() {

dcomp z = (4., -0.5);

icomp w = (6,2);

cout << endl << z << " " << w;

Program 2.4: A program to show the maximum and minimum values of various data

types.

1 #include <iostream>

2 #include <climits> // max & min size of integer types

3 #include <cfloat> // max & min size of real types

4 using namespace std;

5

6 /\*\*

7 \* Print out the extreme values of various integer types.

8 \*/

9

10 int main() {

11 cout << "The maximum size of a short is " << SHRT\_MAX << endl;

12 cout << "The minimum size of a short is " << SHRT\_MIN << endl;

13

14 cout << "The maximum size of an int is " << INT\_MAX << endl;

15 cout << "The minimum size of an int is " << INT\_MIN << endl;

16

17 cout << "The maximum size of a long is " << LONG\_MAX << endl;

18 cout << "The minimum size of a long is " << LONG\_MIN << endl;

19

20 // long long values might not exist on some computers

21 cout << "The maximum size of a long long is " << LLONG\_MAX << endl;

22 cout << "The minimum size of a long long is " << LLONG\_MIN << endl;

23

24

25 cout << "The minimum positive value of a float is "

26 << FLT\_MIN << endl;

27 cout << "The minimum epsilon value of a float is "

28 << FLT\_EPSILON << endl;

29 cout << "The maximum value of a float is "

30 << FLT\_MAX << endl;

31

32 cout << "The minimum positive value of a double is "

33 << DBL\_MIN<< endl;

34 cout << "The minimum epsilon value of a double is "

35 << DBL\_EPSILON << endl;

36 cout << "The maximum value of a double is "

37 << DBL\_MAX << endl;

38

39 // long double might not be defined on some systems

40 cout << "The minimum positive value of a long double is "

41 << LDBL\_MIN<< endl;

42 cout << "The minimum epsilon value of a long double is "

43 << LDBL\_EPSILON << endl;

44 cout << "The maximum value of a long double is "

45 << LDBL\_MAX << endl;

46

47 return 0;

48 }

**Compression**

/\*

Name:

Copyright:

Author:

Date: 15/03/11 20:37

Description:

\*/

#include <iostream>

#include <fstream>

using namespace std;

const int mn = 100000;

const int codeSize = 256;

char code[codeSize];

int Read(char \* fn, char s[], int n) {

int k;

FILE \* f;

f = fopen(fn, "rb");

if (!f) {

cout << " Can't open file " << fn;

cin.get(); return 0;

}

k = fread(s,sizeof(char),n,f); fclose(f);

s[n] = '\0';

return k;

}

void Print(char s[], int n, char \* msg = "") {

cout << msg;

for (int i = 0; i < n; ++i) cout << s[i];

}

void Print(char s[], int n, int d, char \* msg = "") {

cout << msg;

for (int i = 0; i < n; ++i) cout << s[(d+i)%n];

}

int Cmp(char s[], int n, int i, int j) {

for (int k = 0; k < n; ++k)

if (s[(i+k)%n] != s[(j+k)%n])

return (s[(i+k)%n] > s[(j+k)%n]) ? 1 : -1;

return 0;

}

void BubbleSort(char s[], int p[], int n) {

int i, j, n1 = n-1, t;

for (i = 0; i < n; ++i) p[i] = i;

for (i = 0; i < n; ++i)

for (j = n1; j > i; --j)

if (Cmp(s, n, p[j], p[j-1]) < 0) {

t = p[j]; p[j] = p[j-1]; p[j-1] = t;

}

}

void BubbleSortw(char w[], int p[], int n) {

int i, j, n1 = n-1, t;

for (i = 0; i < n; ++i) p[i] = i;

for (i = 0; i < n; ++i)

for (j = n1; j > i; --j)

if (w[p[j]] < w[p[j-1]]) {

t = p[j]; p[j] = p[j-1]; p[j-1] = t;

}

}

int BW(char s[], char w[], int n) {

int p[mn];

int r;

BubbleSort(s,p,n);

// for (int i = 0; i < n; ++i) Print(s, n, p[i], "\n");

for (int i = 0; i < n; ++i) {

w[i] = s[(p[i]+(n-1))%n];

if (p[i] == 0) r = i;

}

w[n] = '\0';

return r;

}

/\*

0 1 2 3 4 5

n p m a a a

3 4 5 2 0 1

\*/

void WB(char w[], char s[], int n, int d) {

int p[mn];

int r;

BubbleSortw(w,p,n);

d = p[d];

for (int i = 0; i < n; ++i) {

s[i] = w[d];

d = p[d];

}

s[n] = '\0';

}

void InitCode() {

int cs = codeSize-1;

for (int i = 0; i < cs; ++i) code[i] = i+1;

code[cs] = 0;

}

int Find(char c) {

return int(strchr(code,c)-code);

}

void ShiptCode(int p) {

char c = code[p];

memmove((code+1), code, p\*sizeof(char));

code[0] = c;

}

void MoveToFront(char \* fn, char w[], int d, int n) {

int i, p;

char c;

InitCode();

ofstream f(fn); f << n << ' ' << d << ' ';

for (i = 0; i < n; ++i) {

p = Find(w[i]);

f << p << ' ';

if (p) ShiptCode(p);

}

f.close();

}

void Test(char \* fn, char \* gn) {

char s[mn];

char w[mn];

int n = Read(fn,s,mn-1);

// Print(s,n,"\n Input: \n");

// int n = strlen(s);

// cout << s << " " << n << " " << strlen(s); cin.get();

int d = BW(s,w,n);

// WB(w, s, n, d);

// cout << s << " " << n << " " << strlen(s); cin.get();

MoveToFront(gn,w,d,n);

}

void Burn(char \* fn, char \* gn) {

char s[mn];

char w[mn];

char c;

ifstream f(fn);

InitCode();

int n, i, p, d;

f >> n >> d; // cout << n << ' ' << d << ' '; cin.get();

for (i = 0; i < n; ++i) {

f >> p;

// if (i < 50 || i > n-5) cout << p << ' ';

// if (n % 100 == 0) cin.get();

w[i] = code[p]; // cout << code[p];

if (p) ShiptCode(p);

}

f.close();

w[n] = '\0';

WB(w, s, n, d);

FILE \*g;

g = fopen(gn,"wb");

fwrite(s,sizeof(char),n,g); fclose(g);

}

void TestBW(char \* fn) {

char s[mn];

char w[mn];

int n = Read(fn,s,mn);

int d = BW(s,w,n);

WB(w,s,n,d);

cout << endl << s;

}

main() {

Test("Comp.cpp","comp.mtf");

Burn("Comp.mtf", "NewComp.cpp");

cout << endl << " Fini !"; cin.get();

return 0;

}

# Tham biến Hàm

# void BubbleSort(char \*s, int \* c, int (\*cmp)(char \* v ,int i, int j));

# int Cmp(char \*s,int i,int j);

**bool Comp(int a, int b, bool(\*f)(int x, int y)) {**

**return f(a,b);**

**}**

**bool f(int a, int b) { return (a <= b); }**

**// Tong So bit 1 trong a**

**int GetBit(int a) {**

**int d = 0;**

**while (a) {**

**d += (a&1);**

**a >>= 1;**

**}**

**return d;**

**}**

**bool g(int a, int b) { return (GetBit(a) <= GetBit(b)); }**

# Call: Comp(a,b,f); Comp(a,b,g);

**random**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <time.h>**

**int main(void)**

**{**

**int i;**

**time\_t t;**

**srand((unsigned) time(&t));**

**printf("Ten random numbers from 0 to 99\n\n");**

**for(i=0; i<10; i++)**

**printf("%d\n", rand() % 100);**

**return 0;**

**}**

Pointers to methods and membes

SpreadsheetCell myCell;

double (SpreadsheetCell::\*methodPtr) () const = &SpreadsheetCell::getValue;

cout << (myCell.\*methodPtr)() << endl;

Most of the time C++ programmers simplify the first line by using a typedef:

SpreadsheetCell myCell;

Typedef

typedef double (SpreadsheetCell::\*PtrToGet) () const;

PtrToGet methodPtr = &SpreadsheetCell::getValue;

cout << (myCell.\*methodPtr)() << endl;

The form of a DevCPP program

# //#include <cstdlib>

# #include <string.h>

# #include <fstream>

# #include <iostream>

# #include <stdio.h>

# //#include <Math.h>

# //#include <time.h>

# //#include <stdlib.h>

# //#include <conio.h>

# using namespace std;

# // D A T A A N D V A R I A B L E

# 

# // P R O T O T Y P E S

# // I M P L E M E N T A T I O N

# int main() // int argc, char \* argv[])

# {

# string x = "abc"; cout << endl << "x = " << x;

# string y = "aed"; cout << endl << "y = " << y;

# x += y; cout << endl << " x + y = " << x;

# if (x < y) cout << endl << " x < y";

# cout << endl;

# system("PAUSE");

# return EXIT\_SUCCESS;

# }

**$Thoigian**

**<time.h>**

**time\_t t1,t2;**

**t1 = time(NULL);**

**// devc ko co ham delay(2000); // 2 sec**

**…**

**t2 = time(NULL);**

**float d = difftime(t2,t1);// sec.**

# typedef

#ifdef \_\_cplusplus

typedef void (\*fptr)(int);

#else

typedef void (\*fptr)();

#endif

typedef enum { kPieceTypeKing, kPieceTypeQueen, kPieceTypeRook,

kPieceTypePawn

} PieceT;

typedef struct {

char firstInitial;

char middleInitial;

char lastInitial;

int employeeNumber;

int salary;

} EmployeeT;

setcolor

enum Color { DARKBLUE = 1, DARKGREEN, DARKTEAL, DARKRED, DARKPINK,

DARKYELLOW,GRAY, DARKGRAY, BLUE, GREEN, TEAL, RED,

PINK, YELLOW, WHITE };

void SetColor(Color c){

HANDLE hCon = NULL;

hCon = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hCon, c);

}

$frac

**Thiết kế lớp Frac: 3 files; Visual Studio Expression MS 2005**

**Project: FracTest**

**Chú ý trường Sum trong Frac dùng để đếm số đói tượng được sinh ra và hủy. Gán trước trong Frac.cpp**

**int Frac::Sum = 0;**

**// Frac.h**

#ifndef Frac\_h

#define Frac\_h

#include "stdafx.h"

// #include "Frac.h"

#include <iostream>

#include <math.h>

using namespace std;

class Frac {

protected:

static int Sum;

int Num;

int Den;

public:

Frac(int n = 0, int d = 1) {

if (d <= 0) {

cout << endl << " Denominator must be a positive integer ";

cin.get(); exit(1);

}

Num = n; Den = d;

++Sum; cout << endl << " created new one. Now Sum = " << Sum;

}

~Frac() {

--Sum; cout << endl << " destroyed one. Now Sum = " << Sum;

}

void Set(int n = 0, int d = 1) {

if (d <= 0) {

cout << endl << " Denominator must be a positive integer ";

cin.get(); exit(1);

}

Num = n; Den = d;

}

void Print(char \* s = "", char \* e = "") {

cout << s << Num << "/" << Den << e;

}

void RPrint(char \* s = "", char \* e = "") {

cout << s << Num;

if (Num == 0 || Den == 1) return;

cout << "/" << Den << e;

}

Frac Reduce();

friend Frac operator +(Frac x, Frac y);

friend Frac operator -(Frac x, Frac y);

friend Frac operator \*(Frac x, Frac y);

friend Frac operator /(Frac x, Frac y);

void operator +=(Frac y) { \*this = \*this + y; }

void operator -=(Frac y) { \*this = \*this - y; }

void operator \*=(Frac y) { \*this = \*this \* y; }

void operator /=(Frac y) { \*this = \*this / y; }

Frac operator ++() { \*this += 1; return \*this; }

Frac operator --() { \*this -= 1; return \*this; }

};

#endif

**// Frac.cpp**

#include "stdafx.h"

#include "Frac.h"

#include <iostream>

using namespace std;

int Frac::Sum = 0;

int Gcd(int a, int b) { // Greatest Common Divisor

int r;

a = abs(a); b= abs(b);

while(b) {

r = a % b; a = b; b = r;

}

return a;

}

// Lcm - Lowest Common Multiple

int Lcm(int a , int b ) { return a\*(b/Gcd(a,b)); }

Frac Frac::Reduce() {

int d = Gcd(Num,Den);

Num /= d; Den /= d;

return \*this;

}

Frac operator +(Frac x, Frac y) {

int m = Lcm(x.Den,y.Den);

Frac z(x.Num\*(m/x.Den)+y.Num\*(m/y.Den), m);

return z.Reduce();

}

Frac operator -(Frac x, Frac y) {

int m = Lcm(x.Den,y.Den);

Frac z(x.Num\*(m/x.Den)-y.Num\*(m/y.Den), m);

return z.Reduce();

}

Frac operator \*(Frac x, Frac y) {

Frac z(x.Num\*y.Num, x.Den\*y.Den);

return z.Reduce();

}

Frac operator /(Frac x, Frac y) {

if (y.Num == 0) {

cout << endl << " Err: Divide by zero. ";

cin.get(); exit(2);

}

int s = (y.Num < 0) ? -1 : 1;

Frac z(s\*x.Num\*y.Den, s\*x.Den\*y.Num);

return z.Reduce();

}

**// Fractest.cpp**

// FracTest.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "Frac.h"

#include <iostream>

using namespace std;

int main(){

Frac x(5), y(3,4), z;

x.RPrint("\n x = ");

y.RPrint("\n y = ");

z.RPrint("\n z = ");

z = x\*y;

z.RPrint("\n Now z = (++x)\*y = ");

cout << endl << " Fini ";

cin.get();

return 0;

}

**typedefs**

**A typedef provides a new name for an existing type. You can think of a typedef simply as syntax for introducing a synonym for an existing type name. typedefs do not create new types — they only provide a new way to refer to an old type. You can use the new type name and the old type name interchangeably. Variables created with the new type name are completely compatible with those created with the original type name.**

**You might be surprised at the simplicity of the previous paragraph’s definition for typedefs. You’ve probably used typedefs in your code, or at least seen code that uses them, and they didn’t seem that easy.**

**However, if you examine all the uses, you will see that they are simply providing alternate typenames.**

**The most common use of typenames is to provide manageable names when the real typenames become too unwieldy. This situation commonly arises with templates. For example, suppose you want to use the Grid template from Chapter 11 to create a spreadsheet, which is a Grid of SpreadsheetCells. Without typedefs, anytime you want to refer to the type of this Grid, for declaring variables, specifying function parameters, and so on, you would have to write**

**Grid<SpreadsheetCell>:**

**int main(int argc, char\*\* argv)**

**{**

**Grid<SpreadsheetCell> mySpreadsheet;**

**// Rest of the program . . .**

**}**

**void processSpreadsheet(const Grid<SpreadsheetCell>& spreadsheet)**

**{**

**// Body omitted**

**}**

**Initialization order of nonlocal variables in different source files is undefined.**

**One tricky aspect of typedefs is that the typenames can include the scope qualifiers. For example, in Chapter 9, you saw this typedef:**

**typedef Spreadsheet::SpreadsheetCell SCell;**

**This typedef creates a short name SCell to refer to the SpreadsheetCell type inside the Spreadsheet scope.**

**The STL uses typedefs extensively to provide shorter names for types. For example, string is actually a typdef that looks like this:**

**typedef basic\_string<char> string;**

**Casts**

**As explained in Chapter 1, the old-style C casts with () still work in C++. However, C++ also provides four new casts: static\_cast, dynamic\_cast, const\_cast, and reinterpret\_cast. You should use**

**the C++ style casts instead of the old C-style casts because they perform more type checking and stand out better syntactically in your code.**

**This section describes the purposes for each cast and specifies when you would use each of them.**

**const\_cast**

**The const\_cast is the most straightforward. You can use it to cast away const-ness of a variable. It is the only cast of the four that is allowed to cast away const-ness. Theoretically, of course, there should be no need for a const cast. If a variable is const, it should stay const. In practice, however, you sometimes find yourself in a situation where a function is specified to take a const variable, which it must**

**then pass to a function that takes a non-const variable. The “correct” solution would be to make const consistent in the program, but that is not always an option, especially if you are using third-party**

**libraries. Thus, you sometimes need to cast away the const-ness of a variable. Here is an example:**

**void g(char\* str)**

**{**

**// Function body omitted for brevity**

**}**

**g(const\_cast<char\*>(str));**

**// Function body omitted for brevity**

**}**

**static\_cast**

**You can use the static\_cast to perform explicitly conversions that are supported directly by the lan-**

**guage. For example, if you write an arithmetic expression in which you need to convert an int to a double in order to avoid integer division, use a static\_cast:**

**int i = 3;**

**double result = static\_cast<double>(i) / 10;**

**You can also use static\_cast to perform explicitly conversions that are allowed because of user-**

**defined constructors or conversion routines. For example, if class A has a constructor that takes an object**

**of class B, you can convert a B object to an A object with a static\_cast. In most situations where you**

**want this behavior, however, the compiler will perform the conversion automatically.**

**Another use for the static\_cast is to perform downcasts in an inheritance hierarchy. For example:**

**class Base**

**{**

**public:**

**Base() {};**

**virtual ~Base() {}**

**};**

**class Derived : public Base**

**{**

**public:**

**Derived() {}**

**virtual ~Derived() {}**

**};**

**int main(int argc, char\*\* argv)**

**{**

**Base\* b;**

**Derived\* d = new Derived();**

**b = d; // Don’t need a cast to go up the inheritance hierarchy**

**d = static\_cast<Derived\*>(b); // Need a cast to go down the hierarchy**

**Base base;**

**Derived derived;**

**Base& br = base;**

**Derived& dr = static\_cast<Derived&>(br);**

**return (0);**

**}**

**These casts work with both pointers and references. They do not work with objects themselves.**

**339**

**Understanding C++ Quirks and Oddities**

**15\_574841 ch12.qxd 12/15/04 3:43 PM Page 339Note that these casts with static\_cast do not perform runtime type checking. They allow you to con-**

**vert any Base pointer to a Derived pointer or Base reference to a Derived reference, even if the Base**

**really isn’t a Derived at run time. To perform the cast safely, with runtime type checking, use the**

**dynamic\_cast.**

**static\_casts are not all-powerful. You can’t static\_cast pointers of one type to pointers of another**

**unrelated type. You can’t static\_cast pointers to ints. You can’t static\_cast directly objects of one**

**type to objects of another type. You can’t static\_cast a const type to a non-const type. Basically, you**

**can’t do anything that doesn’t make sense according to the type rules of C++.**

**reinterpret\_cast**

**The reinterpret\_cast is a bit more powerful, and concomitantly less safe, than the static\_cast.**

**You can use it to perform some casts that are not technically allowed by C++ type rules, but which might**

**make sense to the programmer in some circumstances. For example, you can cast a pointer type to any**

**other pointer type, even if they are unrelated by an inheritance hierarchy. Similarly, you can cast a refer-**

**ence to one type to a reference to another type, even if the types are unrelated. You can also cast pointers**

**to ints and ints to pointers. Here are some examples:**

**class X {};**

**class Y {};**

**int main(int argc, char\*\* argv)**

**{**

**int i = 3;**

**X x;**

**Y y;**

**X\* xp;**

**Y\* yp;**

**// Need reinterpret cast to perform pointer conversion from unrelated classes**

**// static\_cast doesn’t work.**

**xp = reinterpret\_cast<X\*>(yp);**

**// Need reinterpret\_cast to go from pointer to int and from int to pointer**

**i = reinterpret\_cast<int>(xp);**

**xp = reinterpret\_cast<X\*>(i);**

**// Need reinterpret cast to perform reference conversion from unrelated classes**

**// static\_cast doesn’t work.**

**X& xr = x;**

**Y& yr = reinterpret\_cast<Y&>(x);**

**return (0);**

**}**

**You should be very careful with the reinterpret\_cast because it “reinterprets” raw bits as a different**

**type without performing any type checking.**

**340**

**Chapter 12**

**15\_574841 ch12.qxd 12/15/04 3:43 PM Page 340dynamic\_cast**

**As mentioned in the discussion of static\_cast, the dynamic\_cast provides a run-time check on casts**

**within an inheritance hierarchy. You can use it to cast pointers or references. dynamic\_cast checks the**

**runtime type information of the underlying object at run time. If the cast doesn’t make sense,**

**dynamic\_cast returns NULL (for the pointer version) or throws a bad\_cast exception (for the reference**

**version).**

**Note that the runtime-type information is stored in the vtable of the object. Therefore, in order to use**

**dynamic\_cast, your classes must have at least one virtual function.**

**Here are some examples:**

**#include <typeinfo>**

**#include <iostream>**

**using namespace std;**

**class Base**

**{**

**public:**

**Base() {};**

**virtual ~Base() {}**

**};**

**class Derived : public Base**

**{**

**public:**

**Derived() {}**

**virtual ~Derived() {}**

**};**

**int main(int argc, char\*\* argv)**

**{**

**Base\* b;**

**Derived\* d = new Derived();**

**b = d;**

**d = dynamic\_cast<Derived\*>(b);**

**Base base;**

**Derived derived;**

**Base& br = base;**

**try {**

**Derived& dr = dynamic\_cast<Derived&>(br);**

**} catch (bad\_cast&) {**

**cout << “Bad cast!\n”;**

**}**

**return (0);**

**}**

**341**

**Understanding C++ Quirks and Oddities**

**15\_574841 ch12.qxd 12/15/04 3:43 PM Page 341In the preceding example, the first cast should succeed, while the second should throw an exception.**

**Chapter 15 covers the details of exception handling.**

**Note that you can perform the same casts down the inheritance hierarchy with a static\_cast or**

**reinterpret\_cast. The difference with dynamic\_cast is that it performs runtime (dynamic) type**

**checking.**

**Summary of Casts**

**The following table summarizes the casts you should use for difference situations.**

**Situation Cast**

**Remove const-ness const\_cast**

**Explicit cast supported by language static\_cast**

**(e.g., int to double, int to bool)**

**Explicit cast supported by user-defined**

**constructors or conversions static\_cast**

**Object of one class to object of another Can’t be done**

**(unrelated) class**

**Pointer-to-object of one class to pointer-to-object of static\_cast**

**another class in the same inheritance hierarchy or dynamic\_cast**

**Reference-to-object of one class to reference-to-object static\_cast or**

**of another class in the same inheritance hierarchy dynamic\_cast**

**Pointer-to-type to unrelated pointer-to-type reinterpret\_cast**

**Reference-to-type to unrelated reference-to-type reinterpret\_cast**

**Pointer to int/ int to pointer reinterpret\_cast**

**Pointer-to-function to pointer-to-function reinterpret\_cast**

**342**

**Chapter 12**

**15\_574841 ch12.qxd 12/15/04 3:43 PM Page 342**

**void \* memset(void \* a,int c,size\_t n);**

**memset(a,0,sizeof(a)); // gán toàn 0**

**void \*memmove(void \*dest, const void \*src, size\_t n);**

**// <mem.h>, <string.h> gán toàn c cho mọi byte**

**memcpy(b,a,sizeof(a)); Copy tu a sang b**

**main() {**

**char x[200], y[200];**

**strcpy(x,"abcd"); // x = "abcd"**

**strcpy((y+1), x); // y = "?abcd"**

**memmove((x+2),x,strlen(x)+1); // x = "??abcd"**

**return 0;**

**}**

**Friends**

**C++ allows classes to declare that other classes or nonmember functions are friends, and can access**

**protected and private data members and methods. For example, the SpreadsheetCell class could**

**specify that the Spreadsheet class is its “friend” like this:**

**class SpreadsheetCell**

**{**

**public:**

**friend class Spreadsheet;**

**// Remainder of the class omitted for brevity**

**};**

**Now all the methods of the Spreadsheet class can access the private and protected data and mem-**

**bers of the SpreadsheetCell class.**

**Similarly, you can specify that one or more functions or members of another class are friends. For**

**example, you might want to write a function to verify that the value and the string of a**

**SpreadsheetCell object are really in synch. You might want this verification routine to be outside the**

**SpreadsheetCell class to model an external audit, but the function should be able to access the inter-**

**nal data members of the object in order to check it properly. Here is the SpreadsheetCell class defini-**

**tion with a friend checkSpreadsheetCell() function:**

**class SpreadsheetCell**

**{**

**public:**

**// Omitted for brevity**

**friend bool checkSpreadsheetCell(const SpreadsheetCell &cell);**

**// Omitted for brevity**

**};**

**The friend declaration in the class serves as the function’s prototype. There’s no need to write the**

**prototype elsewhere (although it’s harmless to do so).**

**Here is the function definition:**

**bool checkSpreadsheetCell(const SpreadsheetCell &cell)**

**{**

**return (SpreadsheetCell::stringToDouble(cell.mString) == cell.mValue);**

**}**

**You write this function just like any other function, except that you can directly access private and**

**protected data members of the SpreadsheetCell class. You don’t repeat the friend keyword on the**

**function definition.**

**friend classes and methods are easy to abuse; they allow you to violate the principle of abstraction by**

**exposing internals of your class to other classes or functions. Thus, you should use them only in limited**

**circumstances such as operator overloading.**

**Char \* s = "abc";**

**char s[100] = "abc";**

File operations

Doc 1 dong

**ofstream f(fn,ios::app); // Open for appending**

**while ((b = fgetc(f)) != EOF)**

**f.getline(s,mn,'\n');**

**// doc toi da mn ki tu ke ca dau cach , gap \n, doi thanh \0**

**FILE \* f;**

**f = fopen("balls.out","\wt");**

**if (f==NULL) cout << endl << " Failed"; else cout << "OK";**

**…**

**fclose(f);**

**Đọc ghi theo C**

**FILE \*f;**

**f = fopen(fn,”rb”);**

**fread(d,N,M,f): doc tu file f vao mang d, M don vi,**

**moi don vi co kich thuoc N bytes**

**return: so bytes thuc doc**

**f = fopen(fn,”wb”);**

**fwrite(d,N,M,f): Ghi mang d vao file f, M don vi,**

**moi don vi co kich thuoc N bytes**

**return: so bytes thuc ghi**

**Examples**

**FILE \* f, \* g;**

**const int ms = 100000;**

**char a[ms], b[ms];**

**int sf,sg;**

**f = fopen(fn, "rb");**

**sf = fread(a,sizeof(char),ms,f);**

**g = fopen(gn,"rb");**

**sg = fread(b,sizeof(char),ms,g);**

**fclose(f); fclose(g);**

**FILE \*f;**

**f = fopen(bmp,"wb");**

**fwrite((void \*)data, sizeof(Byte), h.Fsize, f);**

**fclose(f);**

**void ReadFile(char \* fn) {**

**ifstream f(fn);**

**char c;**

**while (!f.eof())**

**cout << (c = f.get()); //cout << c;**

**f.close();**

**}**

# Ghi string msg vao file

# Char smg[2000];

# // Nap du lieu vao smg, ket ‘\0’

# ofstream f(txt); f << msg; f.close();

Example

// Add prime numbers to a file named fn

int Expand(char \*fn)

**{**

**int x,d=0;**

**ifstream f(fn);**

**while (true) // scan to the end of file**

**{ f >> x;**

**++d; cout << x << bl;**

**if (f.eof()) break;**

**}**

**f.close();**

**cout << endl << "Total " << d << " number(s)" << endl;**

**ofstream g(fn,ios::app); // open for appending**

**do**

**{**

**x = NextPrime(x);**

**++d;**

**g << x << endl; // write x to file**

**cout << x << bl;**

**} while (!kbhit());**

**g.close();**

**cout << endl << "The last prime number is " << x;**

**cout << endl << "Total " << d << " prime number(s) in file " << fn;**

**}**

Example

**// Read two lines:**

**// The first line: a number n, and**

**// The second line: a string s[500]**

**void Doc() // doc so n va string s[500]**

**{**

**ifstream f(fn);**

**f >> n; f.get();**

**// f.get() moves the file pointer to the next line**

**f.getline(s,mn,'\n'); // doc den khi gap \n, doi thanh \0**

**f.close();**

**}**

Example

**//Read all lines of a file**

**#include <fstream>**

**#include <iostream>**

**using namespace std;**

**main() {**

**char s[2000];**

**const int mn = 2000;**

**ifstream f("tt.cpp");**

**while (f.getline(s,mn,'\n') != NULL) cout << s << endl;**

**f.close();**

**cout << endl << " Fini ";**

**cin.get();**

**return 0;**

**}**

## Writing text files

A text file is a file that stores words and sentences in plain text. You must include the fstream.h header file before you can use files. You must then create a ofstream object which will let you open a file and output data to it. When creating the object you must put the name of the file you want to open in brackets. If the file does not exist it will be created and if it already exists it will be cleared.

ofstream f("test.txt");

You write to a file in a similar way as you write to the screen. All you have to do is replace cout with the name of the ofstream object.

f << "Hello";

Use the close method of the ofstream object to close the file when you are finished working with it. If you don't close a file then some data may not be written to it.

f.close();

## Appending to files

Files are cleared by default when you open them. If you want to add things to an existing file then you must open it for appending by using ios::app when opening the file.

ofstream f("test.txt",ios::app);

## Reading text files

You must declare an ifstream object instead of an ofstream when reading from files.

ifstream f("test.txt");

You need to declare a character array to store the data read when reading from a file. The character array can be any size as long as it is big enough to store what you are reading in.

char s[50];

Reading from a file is similar to using cin. Just replace cin with the name of the ifstream object.

f >> s;

## Writing data files

A data file is a file that is usually used to store structures. First we will create a structure called teststruct.

struct teststruct  
{  
   int a;  
   int b;  
};

Now let's declare a struct of the teststruct type and set its values.

teststruct ts;  
ts.a = 5;  
ts.b = 6;

You open a data file in the same way as a text file but you should give the file a .dat extension.

ofstream f("test.dat");

Writing to a data file is a bit more complicated. You must use the write method of the ofstream object. The first parameter of the write method is a pointer to the data structure to be written. The second parameter is the size of the object that you are writing.

f.write((char \*)(&ts),sizeof(ts));

Remember to close the file when you are finished writing to it.

## Reading data files

To read from a data file we will first create an ifstream object and a structure of the teststructure type.

ifstream f("test.dat");  
teststructure ts;

The read method of the ifstream object looks similar to the write method of the ofstream.

f.read((char \*)(&ts),sizeof(ts));

You should then test to see if the values were read properly by writing them to the screen.

cout << c.a << endl;  
cout << c.b << endl;

// Ghi 100 so float to File roi doc lai

void WriteFile(char \* fn) {

float v = 0.12345;

ofstream f(fn);

int i;

for (i = 0; i < 100; ++i) {

f.write((char \*)(&v),sizeof(v));

v += 1.0;

}

f.close();

ifstream g(fn);

for (i = 0; i < 100; ++i) {

g.read((char \*)(&v),sizeof(v));

cout << v << " ";

}

g.close();

}

Random

**#include <time.h>**

**void Gen(int a[], int n)**

**{ srand((unsigned int)time(NULL));**

**for (int i = 0; i < n; ++i)**

**a[i] = rand() % 100;**

**}**

Time counter

**<time.h>**

**time\_t t1,t2;**

**t1 = time(NULL);**

**// devc ko co ham delay(2000); // 2 sec**

**…**

**t2 = time(NULL);**

**float d = difftime(t2,t1);// sec.**

string

**<string.h> has the folowing operations:**

**==, !=, <, <= , >, >=, = , +, +=.**

**string x = "abc"; string y = "1234"; x += y;**

**if (x <= y)….**

**string và char \* có những điểm khác nhau.**

**Declaration: char s[100];**

**Assignment: strcpy(s,"This is a string");**

Access variable-argument lists.

**#include <stdarg.h>**

**void List(int n,...) {**

**int sum = 0;**

**int i,j;**

**va\_list p;**

**va\_start(p,n);**

**cout << endl << " List of " << n << " element(s): ";**

**for (i = 0; i < n; ++i) {**

**j = va\_arg(p,int);**

**sum += j;**

**cout << j << " ";**

**}**

**va\_end(p);**

**cout << endl << " Total Sum = " << sum;**

**}**

**using namespace std;**

**// D A T A A N D V A R I A B L E**

**int List(int i,...) // ending by -1**

**{**

**int d = 0;**

**va\_list p;**

**va\_start(p,i); // Start at the first element of the list**

**while ( i != -1)**

**{**

**++d;**

**cout << i << " ";**

**i = va\_arg(p,int); // Next**

**}**

**va\_end(p); /\* Reset variable arguments. \*/**

**return d;**

**}**

**int main()**

**{**

**List(1,2,3,4,5,-1);**

**cout << endl << " Fini "; cin.get();**

**return 0;**

**}**

**<string.h>**

**Constructors:**

**String(const char \* s = "");**

**String(const String& sourceString);**

**~String();**

**Member Functions:**

**String::hashValue String::isA**

**String::isEqual String::isLessThan**

**String::nameOf String::operator =**

**String::operator char \* String::printOn**

**String::String String::~String**

**See Also:**

**Container Classes**

**memcmp memcpy memicmp memmove memset**

**movedata movmem setmem stpcpy strcat**

**strchr strcmp strcmpi strcpy strcspn**

**strdup \_strerror strerror stricmp strlen**

**strlwr strncat strncmp strncmpi strncpy**

**strnicmp strnset strpbrk strrchr strrev**

**strset strspn strstr strtok strxfrm**

**strupr**

▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrstr, ▐ <STRING.H>

▌strstr ▐

▀▀▀▀▀▀▀▀▀▀▀▀

Finds the first occurrence of a substring in another string

Declaration:

■ char \*strstr(const char \*s1, const char \*s2);

■ char far \* far \_fstrstr(const char far \*s1, const char far \*s2);

Remarks:

strstr and \_fstrstr scan s1 for the first

occurrence of the substring s2.

Return Value:

■ On success, strstr returns a pointer to the element in s1 where s2

begins (points to s2 in s1).

■ On error (if s2 does not occur in s1), strstr returns null.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strstr ║ Yes │ Yes │ Yes │ ║

\_fstrstr ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

Example:

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*str1 = "Borland International", \*str2 = "nation", \*ptr;

ptr = strstr(str1, str2);

printf("The substring is: %s\n", ptr);

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrrchr, ▐

▌strrchr ▐ <STRING.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀

Finds the last occurrence of c in s

Declaration:

■ char \*strrchr(const char \*s, int c);

■ char far \* far \_fstrrchr(const char far \*s, int c);

Remarks:

strrchr scans a string in the reverse

direction, looking for a specific character.

\_fstrrchr is the far version.

strrchr finds the last occurrence of the

character c in the string s.

The null-terminator is considered to be part

of the string.

Return Value:

■ On success, strrchr returns a pointer to

the last occurrence of the character c.

■ If c does not occur in s, strrchr returns

null.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strrchr ║ Yes │ Yes │ Yes │ ║

\_fstrrchr ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

Example:

#include <string.h>

#include <stdio.h>

int main(void)

{

char string[15];

char \*ptr, c = 'r';

strcpy(string, "This is a string");

ptr = strrchr(string, c);

if (ptr)

printf("The character %c is at position: %d\n", c, ptr-string);

else

printf("The character was not found\n");

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrchr,▐ <STRING.H>

▌ strchr ▐

▀▀▀▀▀▀▀▀▀▀▀

Scans a string for the first occurence of a given character

Declaration:

■ char \*strchr(const char \*s, int c);

■ char far \* far \_fstrchr(const char far \*s, int c);

Remarks:

strchr scans a string in the forward

direction, looking for a specific character.

\_fstrchr is the far version.

These functions find the first occurrence of

the character c in the string s.

The null-terminator is considered to be part

of the string; for example,

strchr(strs, 0)

returns a pointer to the terminating null

character of the string strs.

Return Value:

■ On success, returns a pointer to the

first occurrence of the character c in

string s.

■ On error (if c does not occur in s),

returns null.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strchr ║ Yes │ Yes │ Yes │ ║

\_fstrchr ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

Example:

#include <string.h>

#include <stdio.h>

int main(void)

{

char string[15];

char \*ptr, c = 'r';

strcpy(string, "This is a string");

ptr = strchr(string, c);

if (ptr)

printf("The character %c is at position: %d\n", c, ptr-string);

else

printf("The character was not found\n");

return 0;

}

▌\_fstrcat,▐

▌strcat ▐ <STRING.H>

▀▀▀▀▀▀▀▀▀▀▀

Appends one string to another

Declaration:

■ char \*strcat(char \*dest, const char \*src);

■ char far \* far \_fstrcat(char far \*dest, const char far \*src);

Remarks:

strcat appends a copy of src to the end of

dest. The length of the resulting string is

strlen(dest) + strlen(src).

\_fstrcat is the far version.

Return Value:

strcat returns a pointer to the concatenated

strings.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strcat ║ Yes │ Yes │ Yes │ ║

\_fstrcat ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

Example:

#include <string.h>

#include <stdio.h>

int main(void)

{

char destination[25];

char \*blank = " ", \*c = "C++", \*turbo = "Turbo";

strcpy(destination, turbo);

strcat(destination, blank);

strcat(destination, c);

printf("%s\n", destination);

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstricmp, \_fstrcmp, ▐ <STRING.H>

▌strcmp, strcmpi, stricmp ▐

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ \_fstrcmp and strcmp compare two strings

■ strcmpi (a macro) compares two strings without case sensitivity

■ \_fstricmp and stricmp compare two strings without case sensitivity

Declaration:

■ int strcmp(const char \*s1, const char\*s2);

■ int strcmpi(const char \*s1, const char \*s2)

■ int stricmp(const char \*s1, const char \*s2);

■ int far \_fstrcmp(const char far \*s1, const char far \*s2);

■ int far \_fstricmp(const char far \*s1, const char far \*s2);

Remarks:

■ strcmp performs an unsigned comparison of s1

to s2. \_fstrcmp is the far version.

■ strcmpi (implemented as a macro that calls

stricmp) performs an unsigned comparison of s1

to s2, without case sensitivity.

■ stricmp performs an unsigned comparison of

s1 to s2, without case sensitivity. \_fstricmp

is the far version.

The string comparison starts with the first

character in each string and continues with

subsequent characters until the corresponding

characters differ or until the end of the

strings is reached.

To use strcmpi, you must include STRING.H.

This macro is provided for compatibility with

other C compilers.

Return Value:

These routines return an int value that is

■ < 0 if s1 < s2

■ == 0 if s1 == s2

■ > 0 if s1 > s2

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strcmp ║ Yes │ Yes │ Yes │ ║

strcmpi ║ Yes │ │ │ ║

strnicmp ║ Yes │ │ │ ║

\_fstrcmp ║ Yes │ │ │ ║

\_fstrnicmp ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

\_fstrncmp and \_fstrnicmp

strncmp, strncmpi, and strnicmp

Examples:

strcmp example strcmpi example

stricmp example

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstricmp, \_fstrcmp, ▐ <STRING.H>

▌strcmp, strcmpi, stricmp ▐

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ \_fstrcmp and strcmp compare two strings

■ strcmpi (a macro) compares two strings without case sensitivity

■ \_fstricmp and stricmp compare two strings without case sensitivity

Declaration:

■ int strcmp(const char \*s1, const char\*s2);

■ int strcmpi(const char \*s1, const char \*s2)

■ int stricmp(const char \*s1, const char \*s2);

■ int far \_fstrcmp(const char far \*s1, const char far \*s2);

■ int far \_fstricmp(const char far \*s1, const char far \*s2);

Remarks:

■ strcmp performs an unsigned comparison of s1

to s2. \_fstrcmp is the far version.

■ strcmpi (implemented as a macro that calls

stricmp) performs an unsigned comparison of s1

to s2, without case sensitivity.

■ stricmp performs an unsigned comparison of

s1 to s2, without case sensitivity. \_fstricmp

is the far version.

The string comparison starts with the first

character in each string and continues with

subsequent characters until the corresponding

characters differ or until the end of the

strings is reached.

To use strcmpi, you must include STRING.H.

This macro is provided for compatibility with

other C compilers.

Return Value:

These routines return an int value that is

■ < 0 if s1 < s2

■ == 0 if s1 == s2

■ > 0 if s1 > s2

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strcmp ║ Yes │ Yes │ Yes │ ║

strcmpi ║ Yes │ │ │ ║

strnicmp ║ Yes │ │ │ ║

\_fstrcmp ║ Yes │ │ │ ║

\_fstrnicmp ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

\_fstrncmp and \_fstrnicmp

strncmp, strncmpi, and strnicmp

Examples:

strcmp example strcmpi example

stricmp example

▄▄▄▄▄▄▄▄▄

▌strcoll▐ <STRING.H>

▀▀▀▀▀▀▀▀▀

Compares two strings

Declaration: int strcoll(char \*s1, char \*s2);

Remarks:

strcoll compares the string \*s1 to the string

\*s2, according to the collating sequence set

by setlocale.

Return Value:

strcoll returns a value that is

■ < 0 if s1 < s2

■ == 0 if s1 == s2

■ > 0 i f s1 > s2

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

║ Yes │ │ Yes │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

strcmp strcmpi stricmp strncmp

strncmpi strnicmp strxfrm

Example:

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*two = "International";

char \*one = "Borland";

int check;

check = strcoll(one, two);

if (check == 0)

printf("The strings are equal\n");

if (check < 0)

printf("%s comes before %s\n", one, two);

if (check > 0)

printf("%s comes before %s\n", two, one);

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrcpy,▐

▌strcpy ▐ <STRING.H>

▀▀▀▀▀▀▀▀▀▀▀

Copies string src to dest

Declaration:

■ char \*strcpy(char \*dest, const char \*src);

■ char far \* \_fstrcpy(char far \*dest, const char far \*src);

Remarks:

Copies string src to dest, stopping after the

terminating null character has been moved.

\_fstrcpy is the far version.

Return Value: dest

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strcpy ║ Yes │ Yes │ Yes │ ║

\_fstrcpy ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

strncpy

Example:

#include <stdio.h>

#include <string.h>

int main(void)

{

char string[10];

char \*str1 = "abcdefghi";

strcpy(string, str1);

printf("%s\n", string);

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrcspn, \_fstrspn,▐ <STRING.H>

▌strcspn, strcspn ▐

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ \_fstrcspn and strcspn scan a string for a segment that DOES NOT contain

a subset of a set of characters

■ \_fstrspn and strspn scan a string for a segment that IS a subset of a

set of characters

Declaration:

■ size\_t strcspn(const char \*s1, const char \*s2);

■ size\_t strspn(const char \*s1, const char \*s2);

■ size\_t far \_fstrcspn(const char \*s1, const char far \*s2);

■ size\_t far \_fstrspn(const char far \*s1, const char far \*s2);

Remarks:

■ strcspn and \_fstrcspn find the initial

segment of string s1 that consists entirely of

characters NOT from string s2.

■ strspn and \_fstrspn find the initial segment

of string s1 that consists entirely of

characters from string s2.

Return Value:

■ strcspn and \_fstrcspn: the length of the

initial segment found (see Remarks).

■ strspn and \_fstrspn: the length of the

initial segment found (see Remarks).

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strcspn ║ Yes │ Yes │ Yes │ ║

\_fstrcspn ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

Examples:

strcspn example strspn example

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstricmp, \_fstrcmp, ▐ <STRING.H>

▌strcmp, strcmpi, stricmp ▐

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ \_fstrcmp and strcmp compare two strings

■ strcmpi (a macro) compares two strings without case sensitivity

■ \_fstricmp and stricmp compare two strings without case sensitivity

Declaration:

■ int strcmp(const char \*s1, const char\*s2);

■ int strcmpi(const char \*s1, const char \*s2)

■ int stricmp(const char \*s1, const char \*s2);

■ int far \_fstrcmp(const char far \*s1, const char far \*s2);

■ int far \_fstricmp(const char far \*s1, const char far \*s2);

Remarks:

■ strcmp performs an unsigned comparison of s1

to s2. \_fstrcmp is the far version.

■ strcmpi (implemented as a macro that calls

stricmp) performs an unsigned comparison of s1

to s2, without case sensitivity.

■ stricmp performs an unsigned comparison of

s1 to s2, without case sensitivity. \_fstricmp

is the far version.

The string comparison starts with the first

character in each string and continues with

subsequent characters until the corresponding

characters differ or until the end of the

strings is reached.

To use strcmpi, you must include STRING.H.

This macro is provided for compatibility with

other C compilers.

Return Value:

These routines return an int value that is

■ < 0 if s1 < s2

■ == 0 if s1 == s2

■ > 0 if s1 > s2

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strcmp ║ Yes │ Yes │ Yes │ ║

strcmpi ║ Yes │ │ │ ║

strnicmp ║ Yes │ │ │ ║

\_fstrcmp ║ Yes │ │ │ ║

\_fstrnicmp ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

\_fstrncmp and \_fstrnicmp

strncmp, strncmpi, and strnicmp

Examples:

strcmp example strcmpi example

stricmp example

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrlwr, \_fstrupr, ▐

▌strlwr, strupr ▐ <STRING.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

Converts s to all lowercase

Declaration:

■ char \*strlwr(char \*s);

■ char far \* far \_fstrlwr(char far \*s);

■ char \*strupr(char \*s);

■ char far \* far \_fstrupr(char far \*s);

Remarks:

■ strlwr and \_fstrlwr convert uppercase

letters (A to Z) in string s to lowercase (a

to z).

■ strupr and \_fstrupr convert lowercase

letters (a to z) in string s to uppercase (A

to Z).

No other characters are changed.

Return Value:

A pointer to the string s.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

tolower toupper

Examples:

strlwr example strupr example

▄▄▄▄▄▄▄▄▄▄▄▄▄

▌\_fstrncat, ▐

▌strncat ▐ <STRING.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀

Appends a portion of one string to another

Declaration:

■ char \*strncat(char \*dest, const char \*src, size\_t maxlen);

■ char far \* far \_fstrncat(char far \*dest, const char far \*src,

size\_t maxlen);

Remarks:

strncat copies at most maxlen characters of

src to the end of dest and then appends a null

character.

\_fstrncat is the far version.

The maximum length of the resulting string is

strlen(dest) + maxlen.

Return Value: dest

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

strncat ║ Yes │ Yes │ Yes │ ║

\_fstrncat ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

Example:

#include <string.h>

#include <stdio.h>

int main(void)

{

char destination[25];

char \*source = " States";

strcpy(destination, "United");

strncat(destination, source, 7);

printf("%s\n", destination);

return 0;

}

**int memcmp (const void \*s1, const void \*s2, size\_t n);**

compares the first n bytes of strings s1 and s2 as unsigned chars; return < 0, 0, > 0

**int memicmp(const void \*s1, const void \*s2, size\_t n);**

compares the first n bytes of strings s1 and s2, ignoring case, return < 0, 0, > 0

**DevCPP khong co ham nay, chi co memmove**

**void movmem(void \*src, void \*dest, unsigned length); <mem.h>**

Moves a block of length bytes from src to dest. Even if the source and destination blocks overlap, the move direction is chosen so that the data is always moved correctly.

Example:

**#include <mem.h>**

**#include <alloc.h>**

**#include <stdio.h>**

**#include <string.h>**

**int main(void)**

**{**

**char \*source = "Borland International";**

**char \*destination;**

**int length;**

**length = strlen(source);**

**destination = (char \*) malloc(length + 1);**

**movmem(source, destination, length);**

**printf("%s\n", destination);**

**return 0;**

**}**

**▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄**

**▌\_fmemccpy, \_fmemcpy,▐ <MEM.H, STRING.H>**

**▌memccpy, memcpy, ▐**

**▌memmove ▐**

**▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀**

**Copies a block of n bytes from src to dest**

**Declaration:**

**■ void \*memccpy(void \*dest, const void \*src, int c, size\_t n);**

**■ void \*memcpy (void \*dest, const void \*src, size\_t n);**

**■ void \*memmove(void \*dest, const void \*src, size\_t n);**

**■ void far \* far \_fmemccpy(void far \*dest, const void far \*src,**

**int c, size\_t n);**

**■ void far \* far \_fmemcpy (void far \*dest, const void far \*src,**

**size\_t n);**

**Remarks:**

**Each of these functions copies a block of n bytes from src to dest.**

**■ With memccpy, the copying stops as soon as either of the following occurs:**

**■ the character c is first copied into dest**

**■ n bytes have been copied into dest**

**■ With memcpy, if src and dest overlap, the behavior is undefined.**

**■ With memmove, even when the src and dest blocks overlap, bytes in the**

**overlapping locations are copied correctly.**

**Return Value:**

**■ If c was copied, memccpy returns a**

**pointer to the byte in dest immediately**

**following c.**

**■ Otherwise, memccpy returns null.**

**■ memcpy and memmove return dest.**

**Portability:**

**╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗**

**memccpy ║ Yes │ Yes │ │ ║**

**memcpy ║ Yes │ Yes │ Yes │ ║**

**memmove ║ Yes │ Yes │ Yes │ ║**

**\_fmemccpy ║ Yes │ │ │ ║**

**\_fmemcpy ║ Yes │ │ │ ║**

**╚═════╧══════╧════════╧══════════╝**

**#include <string.h>**

**#include <stdio.h>**

**int main()**

**{**

**char x[100];**

**strcpy(x, "1234567890"); cout << endl << x;**

**//memmove(&x[8], x, 10);**

**memmove((char \*) (x+3),x,10);**

**cout << endl << x;**

**strcpy(x,"Borland International");**

**cout << endl << x;**

**//memmove(&x[8], x, 10);**

**memmove((char \*) (x+3),x,10);**

**cout << endl << x;**

**cout << endl << " Fini "; cin.get();**

**return 0;**

**}**

**Chu y: neu viet**

**char \* x = "1234567890"; memmove(&x[1], x, 10); // ERR**

# Thiết kế OOP

# // MACH DIEN.CPP

# // DEVCPP n(n+2) moves

# #include <string.h>

# #include <fstream>

# #include <iostream>

# #include <stdarg.h>

# using namespace std;

# // D A T A A N D V A R I A B L E

# class Lamp {

# public:

# int State;

# Lamp(int s = 0) {

# if (s != 0 && s != 1) {

# cout << endl << " State must be 0/1";

# cin.get(); exit(1);

# State = s; }

# }

# void SetState(int s = 0) {

# if (s != 0 && s != 1) {

# cout << endl << " State must be 0/1";

# cin.get(); exit(1); }

# State = s;

# }

# int GetState() { return State; }

# void Print() {

# if (State == 0) cout << " Darg ";

# else cout << " Light ";

# }

# };

# class Switch {

# public:

# int PNum;

# int Pos;

# Switch(int pn = 2, int p = 1) {

# SetPNum(pn); SetPos(p);

# }

# void Press() {

# if (Pos == PNum) Pos = 1; else ++Pos;

# }

# void TestRange(int p) {

# if (p < 1 || p > PNum) {

# cout << endl << " Position must be between 1.." << PNum << "! ";

# cin.get(); exit(1);

# }

# }

# int GetPos() { return Pos; }

# void SetPos(int p = 1) {

# TestRange(p);

# Pos = p;

# }

# void SetPNum(int pn = 1) {

# if (pn < 2) {

# cout << endl << " PNum must be greater or equal 2! ";

# cin.get(); exit(1);

# }

# PNum = pn;

# }

# void Set(int pn = 2, int p = 1) {

# SetPNum(pn); SetPos(p);

# }

# void Print() { cout << Pos << " / " << PNum; }

# };

# class Scheme {

# public:

# static const int mn = 10;

# int Con[mn][mn];

# Lamp L;

# Switch A;

# Switch B;

# Scheme(int s = mn-1, int p1 = 1, int p2 = 1) {

# int st = (s > mn-1) ? mn-1 : s;

# memset(Con,0,sizeof(Con));

# A.Set(st, p1); B.Set(st, p2);

# L.SetState(0);

# }

# void Connect(int i, int j) {

# A.TestRange(i); B.TestRange(j);

# Con[i][j] = 1;

# L.SetState (Con[A.GetPos()][B.GetPos()]);

# }

# void State() {

# cout << endl << " Switch A: " << A.GetPos();

# cout << endl << " Switch B: " << B.GetPos();

# cout << endl << " Connection: " << Con[A.GetPos()][B.GetPos()];

# L.SetState (Con[A.GetPos()][B.GetPos()]);

# L.Print();

# }

# };

# 

# void Test() {

# char ch;

# Scheme s(2,1,2);

# s.Connect(1,1); s.Connect(1,2);

# cout << endl << endl << "------------------------------" << endl;

# do {

# s.State();

# cout << endl << "Press A/B: ";

# do {

# ch = toupper(cin.get());

# if (ch == '.') return;

# } while (ch != 'A' && ch != 'B');

# cout << ch << " pressed; Lamp ";

# if (ch == 'A') s.A.Press(); else s.B.Press();

# } while (1);

# }

# main() {

# Test();

# cout << endl << " Fini "; cin.get();

# }

# /\*

# Name: Data Hiding

# Copyright: Nguyen Xuan Huy

# Author:

# Date: 12/05/10 10:53

# Description: Giau tin trong anh bitmap

# 2 mau (den, trang) va 16 mau

# \*/

# #include <string.h>

# #include <fstream>

# #include <iostream>

# #include <stdio.h>

# using namespace std;

# // D A T A A N D V A R I A B L E

# typedef unsigned int UI, UL;

# typedef unsigned short SI; // 2 bytes

# typedef unsigned short USI; // 2 bytes

# typedef unsigned char UC, Byte;

# const UI \_MaxLenData = 1000000; // chieu dai toi da cua file anh (bytes)

# const UI \_HeaderSize = 54; // kich thuoc heder file anh (54 bytes)

# const UI \_BP = 8; // so diem anh trong 1 Byte

# const UI \_BlockSize = 256; // Bloc Size

# UI segment; // so Byte cua 1 block trong data (32)

# // Thong tin trong Bitmap Header (54 Byte)

# const int \_MaxStrLen = 65536; // kich thuoc toi da cua text file

# typedef struct Bmp\_Header {

# USI ID;// dang file BM hay 19778, 2 Bytes: 0-1

# UI Fsize; // kich thuoc toan tep anh (Bytes), 4 Bytes: 2-5

# UI Reserved;// du phong tam chua toan 0, 4 Bytes: 6-9

# UI OffsetBit; // Byte dau cua vung du lieu, 4 Bytes: 10-13

# UI Isize; // So Byte cho vùng info, 4 Bytes: 14-17

# UI Width; // chieu rong anh (so diem anh tren 1 dong, 4 Bytes: 18-21

# UI Height;// so dong anh, 4 Bytes: 22-25

# USI Planes; // So planes màu (co dinh la 1), 2 Bytes: 26-27

# USI BitCount;// So bit cho mot pixel, 2 Bytes: 28-29

# UI Compression;// Kieu nen du lieu, 0: ko nen,

# // 1: runlen 8bits/pixel,

# // 2: runlen 4bits/pixel, 4 Bytes:30-33

# UI ImSize;// Tong so Byte luu anh (Bytes), 4 Bytes: 34-37

# // ImSize\*8 = Width \* Heigth

# UI XPelsPerMeter; // Do phan giai ngang, 4 Bytes: 38-41, Tính bang pixels/meter

# UI YPelsPerMeter;//Do phan giai doc, 4 Bytes: 42-45, Tính bang pixels/meter

# UI ColorsUsed; // So mau dung trong anh, 4 Bytes: 46-49

# UI ColorsImportant;// So mau su dung khi hien anh, 4 Bytes: 50-53

# } Head, BmpHeader;

# BmpHeader h;

# Byte data[\_MaxLenData]; // mang lu tru data

# Byte dutru[\_MaxLenData]; // du phong

# Byte block[\_BlockSize]; // mot khoi chua \_BlockSize=256 diem anh lien tiep

# UI lineSize; // so bytes tren 1 dong anh = ImSize/Height

# int maxLenMsg; // So ki tu giau toi da troing anh

# int soMau; // so mau cua anh bmp, 2 hoac 16

# char imageName[30]; // ten file anh hien doc

# // P R O T O T Y P E S

# void Size();

# Byte GetBit(Byte, int);

# Byte GetBit(Byte [], int);

# void Get2Bit(Byte , UC & , UC & );

# void Inv(UC &, UI );

# void Inv(UC [], UI );

# void Print(Byte [], int);

# void ReadImage(const char \*);

# void WriteImage(const char \*);

# UI GetNum(Byte [], int, int);

# void GetInfo(Byte [], BmpHeader &);

# void ShowInfo();

# void CopyImage(char \*, char \*);

# void ImToText1(char \*, char \*, char, char);

# void Hid(const char \*, const char \*, const char \*);

# void Hidc1(UI , Byte [], UI);

# void Hid1(const char \*, const char \*);

# void Hidc4(UI , Byte [], UI);

# void Hid4(const char \*, const char \*);

# bool GetBlock1(Byte [], UI);

# char \* Burn(const char \*, char \*);

# char Burnc(Byte [], UI );

# char \* Burn1(char \*);

# char \* Burn4(char \*);

# bool GetBlock4(Byte [], UI);

# void HidFile(const char \*txt, const char \*ibmp, const char \* obmp);

# void BurnFile(const char \* bmp, const char \*txt);

# int Find(Byte [], UI , UI , UI & , UI & );

# // I M P L E M E N T A T I O N

# /\* C A C H A M P H U T R O \*/

# // Hien thi kich thuoc cua mot so kieu du lieu co ban

# void Size() { // Size of some data type

# cout << endl << " hsize: " << sizeof(BmpHeader);

# cout << endl << " L: " << sizeof(long);

# cout << endl << " UI: " << sizeof(UI);

# cout << endl << " SI: " << sizeof(short);

# cout << endl << " USI: " << sizeof(unsigned short);

# cout << endl << " Byte: " << sizeof(unsigned char);

# cout << endl << " Char: " << sizeof(char);

# cout << endl << " data: " << sizeof(data);

# }

# // lay bit j = 0..7 trong byte x

# Byte GetBit(Byte x, int j) { return (x >> j) & (Byte)1; }

# // lay bit j trong day bit data[]

# Byte GetBit(Byte data[] , int j) {

# int byte = j/8, pos = j%8;

# return (data[byte] >> pos) & (Byte)1;

# }

# // Form a unsigned long integer from segment s[d..c]

# UI GetNum(Byte s[], int d, int c) {

# UI x = 0;

# int i;

# for (i = c; i >= d; --i) x = (x << 8) | (UI)s[i]; // x = x\*256 + (UI)s[i]

# return x;

# }

# // Hien thi mang

# void Print(Byte s[], int n) {

# for (int i = 0; i < n; ++i) cout << (int)s[i];

# }

# // dao bit p trong byte x

# void Inv(UC & x, UI p) {

# UC one = (UC)1;

# x ^= (one << p);

# }

# // dao bit thu i trong day bit x

# void Inv(UC x[], UI i) {

# UC one = (UC)1;

# UI byte = i/8, pos = i%8;

# x[byte] ^= (one << pos);

# }

# /\* C A C H A M C H I N H \*/

# // Doc thong tin anh bmp vao mang data

# // trich 54 bytes dau cho header

# void ReadImage(const char \*bmp) {

# ifstream f(bmp);

# if (!f) {

# cout << "\n Khong tim thay file anh " << bmp;

# cin.get(); exit(1);

# }

# memcpy(imageName,bmp,strlen(bmp)+1);

# f.read((char \*)data,\_MaxLenData);

# f.close();

# GetInfo(data,h);

# }

# // Get information to Header: trich 54 bytes dau cho header

# void GetInfo(Byte s[], BmpHeader & h) {

# h.ID = (USI)GetNum(s,0,1);//dang file BM hay 19778, 2 Bytes: 0-1

# h.Fsize = GetNum(s,2,5); // kich thuoc toan tep anh (Bytes),

# // 4 Bytes: 2-5

# h.Reserved = GetNum(s,6,9); // du phong tam chua toan 0,

# // 4 Bytes: 6-9

# h.OffsetBit = GetNum(s,10,13); // Byte dau cua vung du lieu,

# // 4 Bytes: 10-13

# h.Isize = GetNum(s,14,17); // So Byte cho vùng info,

# // 4 Bytes: 14-17

# h.Width = GetNum(s,18, 21); // chieu rong anh (so diem anh tren 1 dong,

# // 4 Bytes: 18-21

# h.Height = GetNum(s,22,25); // so dong anh, 4 Bytes: 22-25

# 

# h.Planes = (USI)GetNum(s,26,27); // So planes màu (co dinh la 1),

# // 2 Bytes: 26-27

# h.BitCount = (USI)GetNum(s,28,29);// So bit cho mot pixel, 2 Bytes: 28-29

# h.Compression = GetNum(s,30,33);// Kieu nen du lieu, 0: ko nen,

# // 1: runlen 8bits/pixel,

# // 2: runlen 4bits/pixel,

# // 4 Bytes:30-33

# h.ImSize = GetNum(s,34,37);// Dien tich anh (Bytes)

# // h.ImSize\*8 = Width\*Height

# // 4 Bytes: 34-37

# h.XPelsPerMeter = GetNum(s,38,41); // Do phan giai ngang, 4 Bytes: 38-41,

# // Tính bang pixels/meter

# h.YPelsPerMeter = GetNum(s,42,45);//Do phan giai doc, 4 Bytes: 42-45,

# // Tính bang pixels/meter

# h.ColorsUsed = GetNum(s,46,49); // So mau dung trong anh, 4 Bytes: 46-49

# h.ColorsImportant = GetNum(s,50,53);// So mau su dung khi hien anh,

# // Tinh cac tham bien phu tro

# lineSize = h.ImSize/h.Height; // So byte tren 1 dong anh

# //Moi egment giau 1 ki tu

# maxLenMsg = h.Width\*h.Height / \_BlockSize - 2;

# if (maxLenMsg < 0) maxLenMsg = 0;

# // segment: so byte cho 1 khoi anh

# if (h.BitCount==1) {

# soMau = 2; segment = 32;

# }

# else if (h.BitCount==4) { // so mau dung cho anh

# soMau = 16; segment = 128;

# }

# }

# // Hien thi thong tin

# void ShowInfo() {

# cout << "\n\n File " << imageName;

# cout << endl << " ID (Loai anh): " << h.ID << " " << soMau << " mau";

# 

# cout << endl << " FSize (Kich thuoc toan file): " << h.Fsize;

# cout << endl << " Width: " << h.Width;

# cout << endl << " Height: " << h.Height;

# cout << endl << " ISize: " << h.Isize;

# cout << endl << " BitCount: " << h.BitCount;

# cout << endl << " Compression: " << h.Compression;

# cout << endl << " Image Size (So Bytes cua data, ImSize): " << h.ImSize;

# cout << endl << " ImSize\*8 = " << h.ImSize\*8;

# 

# cout << endl << " So diem anh = Width\*Height = " << h.Width\*h.Height;

# cout << endl << " So Byte tren 1 dong, lineSize = " << lineSize;

# 

# cout << endl << " OffsetBit = " << h.OffsetBit;

# cout << endl << " Byte dau tien chua du lieu, start: " << h.OffsetBit;

# cout << endl << " So ki tu co the giau toi da: " << maxLenMsg;

# 

# }

# // Copy anh tu file ibmp sang file obmp

# void CopyImage(char \*ibmp, char \*obmp) {

# ReadImage(ibmp);

# WriteImage(obmp);

# }

# // Ghi anh vao file bmp

# void WriteImage(const char \*bmp) {

# ofstream f(bmp);

# f.write((char \*)data, h.Fsize);

# f.close();

# }

# // Chuyen 1 dong anh sang dang text ghi vao file g

# void TextLine1(ofstream & g, UI s, char black, char white) {

# UI i,j,d = 0;

# int k;

# for (j = 0;(d < h.Width) && (j < lineSize) ; ++j)

# for (k = 7; (d < h.Width) && (k >= 0); --k, ++d)

# if (GetBit(data[s+j],k)) g << white; else g << black;

# g << '\n';

# }

# // Chuyen anh sang dang text file

# void ImToText1(char \*bmp, char \* txt, char black, char white) {

# UI s, i;

# ReadImage(bmp);

# s = h.OffsetBit;

# ofstream g(txt);

# for (i = 0; i < h.Height; ++i, s += lineSize)

# TextLine1(g, s,black, white ); // viet dong i

# g.close();

# }

# // Tim 2 bit can sua v1, v2 trong khoi diem anh 0/1 a

# int Find(Byte a[], UI N, UI c, UI & v1, UI & v2) {

# // Compute t = sum { a[i] | i = 1,...,N-1} mod N

# UI t = 0, i;

# Byte one = (Byte)1;

# for (i = 1; i < N; ++i) t += i\*a[i];

# t %= N; // % Toán t? mod

# int b = (t >= c) ? t-c : N-(c-t);

# if (b == 0) return 0;

# if (a[b] == one) { // b > 0, a[b] = 1

# v1 = b;

# return 1;

# }

# if (a[N-b] == 0) { // b > 0, a[b] = 0, a[N-b] = 0

# v1 = N-b;

# return 1;

# }

# // b > 0, a[b] = 0, a[N-b] = 1

# v2 = b;

# do {

# v1 = v2;

# v2 = (v2 + b)%N;

# } while (a[v2] != one);

# return 2;

# }

# // Giau tin msg vao anh bitmap ibmp ghi vao anh obmp

# void Hid(const char \* msg, const char \*ibmp, const char \*obmp) {

# ReadImage(ibmp); ShowInfo();

# if (maxLenMsg == 0) return;

# if (h.BitCount==1) Hid1(msg, obmp); // anh 2 mau den trang

# else if (h.BitCount==4) Hid4(msg, obmp); // anh 16 mau

# }

# 

# // Giau ki tu c vao khoi anh 0/1 s

# void Hidc1(UI c, Byte s[], UI b) {

# UI v1, v2, d = b\*8;

# switch (Find(s,\_BlockSize,c,v1,v2)) {

# case 0: break;

# case 1: Inv(data,d+v1); break;

# case 2: Inv(data,d+v1); Inv(data,d+v2); break;

# }

# }

# // Giau tin msg vao anh bitmap 0/1 ibmp ghi vao anh obmp

# void Hid1(const char \* msg, const char \*obmp) {

# Byte s[\_BlockSize];

# UI b; // Byte dang xet

# UI len = (UI)strlen(msg);

# if (len > maxLenMsg) len = maxLenMsg;

# 

# b = h.OffsetBit;

# int i;

# UI mask = 0x000000ff;

# // Giau 2 bytes cua len

# GetBlock1(s,b); Hidc1(len & mask,s,b); // giau byte thap cua len

# b += segment;

# GetBlock1(s,b); Hidc1((len >> 8) & mask,s,b); // giau byte cao cua len

# b += segment;

# 

# for ( i = 0; i < len; ++i) {

# if (!GetBlock1(s,b)) break;

# Hidc1((UI)msg[i],s,b);

# b += segment;

# }

# WriteImage(obmp);

# }

# // Tric khoi anh 0/1 tu anh den trang

# bool GetBlock1(Byte block[], UI b) { // lay 256 = 32\*8 bit tinh tu data[b]

# UI i, d = b\*8; // dua vao block

# if (b >= h.Fsize) return false;

# for (i = 0; i < \_BlockSize; ++i)

# block[i] = GetBit(data, d+i);

# return true;

# }

# // Lay 2 bit LSB tu 2 diem anh 16 mau

# void Get2Bit(UC x , UC & y, UC & z) {

# UC one = (UC)1;

# y = x & one;

# z = (x >> 4) & one;

# }

# // Tric khoi anh 0/1 tu anh 16 mau

# bool GetBlock4(Byte block[], UI b) { // lay 256 = 32\*8 bit tinh tu data[b]

# UI i;

# int j = 3; // dua vao block

# if (b >= h.Fsize) return false;

# for (i = 0; i < \_BlockSize; i += 2, b++)

# Get2Bit(data[b], block[i],block[i+1]);

# return true;

# }

# // Giau tin msg vao anh bitmap 16 mau ibmp ghi vao anh obmp

# void Hidc4(UI c, Byte s[], UI b) {

# UI v1, v2, byte1, byte2, pos1, pos2;

# 

# int k = Find(s,\_BlockSize,c,v1,v2);

# pos1 = 4\*(v1 % 2); pos2 = 4\*(v2 % 2);

# byte1 = b + v1/2; byte2 = b + v2/2;

# switch (k) {

# case 0: break;

# case 1: Inv(data[byte1],pos1); break;

# case 2: Inv(data[byte1],pos1); Inv(data[byte2],pos2); break;

# }

# }

# // anh 2 mau den trang

# void Hid4(const char \* msg, const char \*obmp) {

# Byte s[\_BlockSize];

# UI b; // Byte dang xet

# UI len = (UI)strlen(msg);

# if (len > maxLenMsg) len = maxLenMsg;

# b = h.OffsetBit;

# int i;

# UI mask = 0x000000ff;

# // Giau len vao 2 byte

# GetBlock4(s,b); Hidc4(len & mask,s,b); // giau len

# b += segment;

# GetBlock4(s,b); Hidc4((len >> 8) & mask,s,b); // giau len

# b += segment;

# for ( i = 0; i < len; ++i) {

# if (!GetBlock4(s,b)) break;

# Hidc4((UI)msg[i],s,b);

# b += segment;

# }

# WriteImage(obmp);

# }

# // Lay thong tin tu anh ibmp dua vao string str

# char \* Burn(const char \*ibmp, char \* str) {

# ReadImage(ibmp); ShowInfo();

# if (maxLenMsg == 0) return NULL;

# if (h.BitCount==1) return Burn1(str); // anh 2 mau den trang

# else if (h.BitCount==4) return Burn4(str); // anh 16 mau

# }

# // trich 1 ki tu tu khoi anh a chua n bytes

# char Burnc(Byte a[], UI n) {

# UI t = 0, i;

# for (i = 1; i < n; ++i) t += i\*(UI)a[i];

# return (char)(t%n);

# }

# // Trich tin mat str tu file anh den trang

# char \* Burn1(char \* str) {

# UI b; // Byte dang xet

# UI i;

# Byte s[\_BlockSize];

# UI len; // so ki tu giau trong file

# b = h.OffsetBit; // xet tu data[b]

# 

# // Doc byte thap cua len

# if (!GetBlock1(s,b)) return NULL;

# len = UI(Burnc(s, \_BlockSize));

# // Doc byte cao cua len

# b += segment;

# if (!GetBlock1(s,b)) return NULL;

# len = len | (UI(Burnc(s, \_BlockSize)) << 8) ;

# b += segment;

# 

# for (i = 0; i < len; ++i){

# if (!GetBlock1(s,b)) break;

# str[i] = Burnc(s, \_BlockSize);

# // cout << str[i];

# b += segment;

# }

# str[i] = '\0';

# return str;

# }

# // Trich tin mat str tu file anh 16 mau

# 

# char \* Burn4(char \* str) {

# UI b; // Byte dang xet

# UI i;

# Byte s[\_BlockSize];

# UI len; // so ki tu giau trong file

# b = h.OffsetBit; // xet tu data[b]

# // Doc byte thap cua len

# if (!GetBlock4(s,b)) return NULL;

# len = UI(Burnc(s, \_BlockSize));

# // Doc byte cao cua len

# b += segment;

# if (!GetBlock4(s,b)) return NULL;

# len = len | (UI(Burnc(s, \_BlockSize)) << 8) ;

# b += segment;

# for (i = 0; i < len; ++i){

# if (!GetBlock4(s,b)) break;

# str[i] = Burnc(s, \_BlockSize);

# b += segment;

# }

# str[i] = '\0';

# return str;

# }

# // Giau file txt vao anh ibmp thu duoc anh obmp

# void HidFile(const char \*txt, const char \*ibmp, const char \* obmp){

# char msg[\_MaxStrLen];

# ReadImage(ibmp); //ShowInfo();

# if (maxLenMsg == 0) return;

# ifstream f(txt); f.read(msg, \_MaxStrLen); f.close();

# cout << endl << " Hid" << endl << msg << "| len = " << strlen(msg);

# if (h.BitCount==1) Hid1(msg, obmp); // anh 2 mau den trang

# else if (h.BitCount==4) Hid4(msg, obmp); // anh 16 mau

# }

# // Lay tin tu anh bmp ghi vao file txt

# void BurnFile(const char \* bmp, const char \*txt) {

# char msg[\_MaxStrLen];

# ReadImage(bmp); //ShowInfo();

# if (maxLenMsg == 0) return;

# if (h.BitCount==1) Burn1(msg); // anh 2 mau den trang

# else if (h.BitCount==4) Burn4(msg); // anh 16 mau

# ofstream f(txt); f << msg; f.close();

# cout << endl << " Burn" << endl << msg<<"| len = " << strlen(msg);

# }

# int main() // int argc, char \* argv[])

# {

# 

# char s[2000];

# //HidFile("Hid.cpp","tulips16.bmp","tulip16new.bmp");

# BurnFile("tulip16new.bmp","Hidnew.cpp");

# //ifstream f("test.txt");

# // f.read(s,2000);

# // //f.getline(s,2000,'\n');

# //f.close();

# // cout << s << endl << " Len = " << strlen(s);

# // cout << "\n Begin hiding ... ";

# // Hid(s,"tulips16.bmp","newtulips16.bmp");

# // cout << "\n\n Burn Newtulips16: " << Burn("newtulips16.bmp",s);

# //cout << endl;

# // ReadImage("tulips16.bmp"); ShowInfo();

# //Hid("Nguyen Xuan Huy so huu anh nay","tulips16.bmp","tulip16new.bmp");

# //cout << endl << Burn("tulip16new.bmp",s);

# // cout << s;

# //ReadImage("mickay.bmp"); ShowInfo();

# //Hid(s,"mickay.bmp","newmickay.bmp");

# //cout << "\n\n Burn NewMickay: " << Burn("newmickay.bmp",s);

# cout << endl << endl << " Fini" << endl;

# cin.get();

# return 0;

# }

math

▄▄▄▄▄▄▄▄▄▄▄▄▄

▌fmod, fmodl▐ <MATH.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀

Calculates x modulo y, the remainder of x/y

Declaration:

■ double fmod(double x, double y);

■ long double fmod(long double (x), long double (y));

Remarks:

fmod and fmodl calculate x modulo y. This is

defined as the remainder f, where

x = (ay + f) for some integer a

and

0 < f < y.

Return Value:

■ Where x = ay + f and 0 < f < y,

fmod and fmodl return the remainder f.

■ Where y = 0, fmod and fmodl return 0.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

fmod ║ Yes │ Yes │ Yes │ ║

fmodl ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

ceil floor modf

Example:

#include <stdio.h>

#include <math.h>

int main(void)

{

double x = 5.0, y = 2.0;

double result;

result = fmod(x,y);

printf("The remainder of (%lf / %lf) is %lf\n", x, y, result);

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌ceil, ceill, floor, floorl▐ <MATH.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ ceil and ceill round up

■ floor and floorl round down

Declaration:

■ double ceil(double x);

■ double floor(double x);

■ long double ceill(long double (x));

■ long double floorl(long double (x));

Remarks:

■ ceil finds the smallest integer not < x.

■ ceill finds the smallest (long double) integer not < x.

■ floor finds the largest integer not > x.

■ floorl finds the largest (long double) integer not > x.

Return Value:

Both ceil and floor return the integer found

as a double; ceill and floorl return the

integer found as a long double.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

ceil ║ Yes │ Yes │ Yes │ ║

ceill ║ Yes │ │ │ ║

floor ║ Yes │ Yes │ Yes │ ║

floorl ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

fmod

Example (for both ceil and floor):

#include <math.h>

#include <stdio.h>

int main(void)

{

double number = 123.54;

double down, up;

down = floor(number);

up = ceil(number);

printf("original number %5.2lf\n", number);

printf("number rounded down %5.2lf\n", down);

printf("number rounded up %5.2lf\n", up);

return 0;

}

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌frexp, frexpl▐ <MATH.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ frexp splits a double number into mantissa and exponent

■ frexpl splits a long double number into mantissa and exponent

Declaration:

■ double frexp(double x, int \*exponent);

■ long double frexp(long double (x), int \*(exponent));

Remarks:

Given x (a double), frexp calculates the

mantissa m (a double) and n (an integer), such

that

x = m \* (2\*\*n)

where

0.5 =< m < 1

frexpl calculates the mantissa and integer for

a long double x.

frexp and frexpl store n in the integer

\*exponent.

Return Value:

frexp and frexpl return the mantissa m.

You can use the function matherr to modify

error handling for frexp, or \_matherrl for

frexpl.

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

fexp ║ Yes │ Yes │ Yes │ ║

fexpl ║ Yes │ │ │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

exp ldexp

Example:

#include <math.h>

#include <stdio.h>

int main(void)

{

double mantissa, number;

int exponent;

number = 8.0;

mantissa = frexp(number, &exponent);

printf("The number %lf is ", number);

printf("%lf times two to the ", mantissa);

printf("power of %d\n", exponent);

return 0;

}

math

▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌abs, ▐ <MATH.H, STDLIB.H, COMPLEX.H>

▌cabs, cabsl,▐ <MATH.H>

▌fabs, fabsl,▐ <MATH.H>

▌labs ▐ <MATH.H, STDLIB.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ abs (a macro) gets the absolute value of an integer

■ cabs and cabsl (macros) calculate the absolute value of a complex number

■ fabs and fabsl calculate the absolute value of a floating-point number

■ labs calculates the absolute value of a long number

Declaration:

■ abs

real: int abs(int x);

complex: double abs(complex x);

■ double cabs(struct complex z);

■ long double cabsl(struct \_complexl (z));

■ double fabs(double x);

■ long double fabsl(long double @E(x));

■ long int labs(long int x);

Remarks:

All of these routines return the absolute

value of their argument. abs, cabs, and cabsl

are macros; fabs and labs are functions.

│ Returns absolute

Routine │ value of...

═════════╪═══════════════════════════════════

abs │ x, an integer

cabs │ z, a complex number

cabsl │ z, a complex number

fabs │ x, a double

labs │ x, a long

abs

▀▀▀

If abs is called when STDLIB.H has been

included, it is treated as a macro that

expands to inline code.

If you want to use the abs function instead of

the macro, include #undef abs in your program,

after the #include <STDLIB.H>.

cabs and cabsl

▀▀▀▀▀▀▀▀▀▀▀▀▀▀

Calling cabs or cabsl is equivalent to calling

sqrt with the real and imaginary components of

z:

sqrt(z.x \* z.x + z.y \* z.y)

In C++, use abs with the complex type defined

in COMPLEX.H.

Return Value:

■ abs:

■ (real): An integer in the range 0 to 32,767;

an argument of -32,768 is returned as -32,768

■ (complex): A double

■ cabs and cabsl:

■ On success, the absolute value of z, a double

■ On overflow, they return HUGE\_VAL and set errno to

ERANGE (result out of range)

■ fabs: A double

■ labs: A long

Error handling for cabs and cabsl can be

modified via matherr and \_matherrl.

Portability ╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

Real abs ║ Yes │ Yes │ Yes │ ║

Complex abs ║ Yes │ │ │ Yes ║

cabs ║ Yes │ Yes │ Yes │ ║

cabsl ║ Yes │ │ │ ║

fabs ║ Yes │ Yes │ Yes │ ║

fabsl ║ Yes │ │ │ ║

labs ║ Yes │ Yes │ Yes │ ║

╚═════╧══════╧════════╧══════════╝

Examples:

abs example cabs example fabs example

labs example

▄▄▄▄▄▄▄▄▄▄▄▄▄▄

Header Files

▀▀▀▀▀▀▀▀▀▀▀▀▀▀

ALLOC.H ASSERT.H BCD.H

BIOS.H COMPLEX.H CONIO.H

CTYPE.H DIR.H DIRENT.H

DOS.H ERRNO.H FCNTL.H

FLOAT.H FSTREAM.H GENERIC.H

GRAPHICS.H IO.H IOMANIP.H

IOSTREAM.H LIMITS.H LOCALE.H

MALLOC.H MATH.H MEM.H

PROCESS.H SETJMP.H SHARE.H

SIGNAL.H STDARG.H STDDEF.H

STDIO.H STDIOSTR.H STDLIB.H

STREAM.H STRING.H STRSTREA.H

SYS\STAT.H SYS\TIMEB.H SYS\TYPES.H

TIME.H VALUES.H

See Also

▀▀▀▀▀▀▀▀

Pre-compiled headers

printf

▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄

▌...printf functions▐ <CONIO.H, STDIO.H>

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

■ cprintf sends formatted output to the text window on the screen

■ fprintf sends formatted output to a stream

■ printf sends formatted output to stdin

■ sprintf sends formatted output to a string

■ vfprintf sends formatted output to a stream, using an argument list

■ vprintf sends formatted output to stdin, using an argument list

■ vsprintf sends formatted output to a string, using an argument list

Declaration:

■ int cprintf ( const char \*format [, argument, ...]);

■ int fprintf (FILE \*stream, const char \*format [, argument, ...]);

■ int printf ( const char \*format [, argument, ...]);

■ int sprintf (char \*buffer, const char \*format [, argument, ...]);

■ int vfprintf(FILE \*stream, const char \*format, va\_list arglist);

■ int vprintf ( const char \*format, va\_list arglist);

■ int vsprintf(char \*buffer, const char \*format, va\_list arglist);

Remarks:

All these functions are declared in STDIO.H,

except cprintf, which is declared in CONIO.H.

The ...printf functions do the following:

■ Accept a series of arguments

■ Apply to each argument a format specifier

contained in the format string \*format

■ Output the formatted data (to the screen,

a stream, stdin, or a string)

These functions apply the first format

specifier to the first argument, the second

specifier to the second argument, the third to

the third, etc., to the end of the format.

┌────────────────────────────────────────┐

│ NOTE │

├────────────────────────────────────────┤

│ There must be enough arguments for the │

│ format. │

│ │

│ If there are not, the results will be │

│ unpredictable and likely disastrous. │

│ │

│ Excess arguments (more than required │

│ by the format) are merely ignored. │

└────────────────────────────────────────┘

Argument│ Functions │ What Argument Is/Does

═════════╪═══════════╪══════════════════════

arglist │ v...printf│ Pointer to a list of

│ │ arguments

argument│ cprintf, │ One of a series of

│ fprintf, │ arguments to which

│ printf, │ the functions apply

│ sprintf │ a format specifier

│ │ contained in \*format

buffer │ sprintf, │ Buffer where function

│ vsprintf │ writes the string

format │ (all) │ Format string

stream │ fprintf, │ Stream where the

│ vfprintf │ functions output the

│ │ formatted data

cprintf

▀▀▀▀▀▀▀

With cprintf, the string is written either

directly to screen memory or by way of a BIOS

call, depending on the value of directvideo.

cprintf does not translate linefeed characters

(\n) into carriage-return/linefeed character

pairs (\r\n).

The v...printf functions

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

The v...printf functions are known as

alternate entry points for the ...printf

functions.

They behave exactly like their ...printf

counterparts, except they accept a pointer to

a list of arguments (va\_list arglist) instead

of accepting an actual list of arguments ([,

address, ...]).

■ NOTE: When you use the SS!=DS flag, vprintf

assumes that the address being passed is in

the SS segment.

Return Value:

■ On success, the ...printf functions return the number of bytes output

■ cprintf returns the number of characters output

■ sprintf does not include the terminating null byte in the count

■ On error, these functions return EOF

Portability:

╔ DOS ╤ UNIX ╤ ANSI C ╤ C++ Only ╗

cprintf ║ Yes │ │ │ ║

fprintf ║ Yes │ Yes │ Yes │ ║

printf ║ Yes │ Yes │ Yes │ ║

sprintf ║ Yes │ Yes │ Yes │ ║

vfprintf ║ Yes │ Yes │ Yes │ ║

vprintf ║ Yes │ Yes │ Yes │ ║

vsprintf ║ Yes │ Yes │ Yes │ ║

╚═════╧══════╧════════╧══════════╝

See Also:

atof fscanf getc getche

putc putch va\_arg va\_end

va\_start

More ...printf topics

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

Format String

Format Specifiers

Format Specifier Conventions

Flag Characters

Input-size Modifiers

Precision Specifiers

Type Characters

Width Specifiers

Examples:

cprintf example fprintf example

printf example sprintf example

vfprintf example vprintf example

vsprintf example

cprintf example

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

#include <conio.h>

int main(void)

{

/\* clear the screen \*/

clrscr();

/\* create a text window \*/

window(10, 10, 80, 25);

/\* output some text in the window \*/

cprintf("Hello world\r\n");

/\* wait for a key \*/

getch();

return 0;

}

format

cprintf example

▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

#include <conio.h>

int main(void)

{

/\* clear the screen \*/

clrscr();

/\* create a text window \*/

window(10, 10, 80, 25);

/\* output some text in the window \*/

cprintf("Hello world\r\n");

/\* wait for a key \*/

getch();

return 0;

}