I/O Redirection in C++

Difficulty Level : Medium • Last Updated : 22 Jun, 2022

In C, we could use the function freopen() to redirect an existing FILE pointer to another stream. The prototype for freopen() is given as

```
FILE * freopen ( const char * filename, const char * mode, FILE * stream );
```

For Example, to redirect the stdout to say a textfile, we could write :

```
freopen ("text_file.txt", "w", stdout);
```

While this method is still supported in C++, this article discusses another way to redirect I/O streams.
C++ being an object-oriented programming language, gives us the ability to not only define our own streams but also redirect standard streams. Thus, in C++, a stream is an object whose behavior is defined by a class. Thus, anything that behaves like a stream is also a stream.
Streams Objects in C++ are mainly of three types :

- **istream :** Stream object of this type can only perform input operations from the stream
- **ostream :** These objects can only be used for output operations.
- **iostream :** Can be used for both input and output operations

All these classes, as well as file stream classes, derived from the classes: ios and streambuf. Thus, filestream and IO stream objects behave similarly.
All stream objects also have an associated data member of class streambuf. Simply put, streambuf object is the buffer for the stream. When we read data from a stream, we don't read it directly from the source, but instead, we read it from the buffer which is linked to the

source. Similarly, output operations are first performed on the buffer, and then the buffer is flushed (written to the physical device) when needed.
C++ allows us to set the stream buffer for any stream, So the task of redirecting the stream simply reduces to changing the stream buffer associated with the stream. Thus, to redirect a Stream A to Stream B we need to do:-

1. Get the stream buffer of A and store it somewhere
2. Set the stream buffer of A to the stream buffer of B
3. If needed to reset the stream buffer of A to its previous stream buffer

We can use the function ios::rdbuf() to perform below two operations.

```
1) stream_object.rdbuf(): Returns pointer to the stream buffer of
stream_object
2) stream_object.rdbuf(streambuf * p): Sets the stream buffer to the object
pointed by p
```

Here is an example program below to show the steps

CPP

```cpp
// Cpp program to redirect cout to a file
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

int main()
{
    fstream file;
    file.open("cout.txt", ios::out);
    string line;

    // Backup streambuffers of  cout
    streambuf* stream_buffer_cout = cout.rdbuf();
    streambuf* stream_buffer_cin = cin.rdbuf();
```

```
    // Get the streambuffer of the file
    streambuf* stream_buffer_file = file.rdbuf();

    // Redirect cout to file
    cout.rdbuf(stream_buffer_file);

    cout << "This line written to file" << endl;

    // Redirect cout back to screen
    cout.rdbuf(stream_buffer_cout);
    cout << "This line is written to screen" << endl;

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

Output:

```
This line is written to screen
Contents of file cout.txt:
This line written to file
```

Time Complexity: O(1)

Space Complexity: O(1)

Note:

The above steps can be condensed into a single step

```
auto cout_buf = cout.rdbuf(file.rdbuf())

// sets couts streambuffer and returns the old
streambuffer back to cout_buf
```

References:

CPP IOS

138

Related Articles

1. C++ Error - Does not name a type

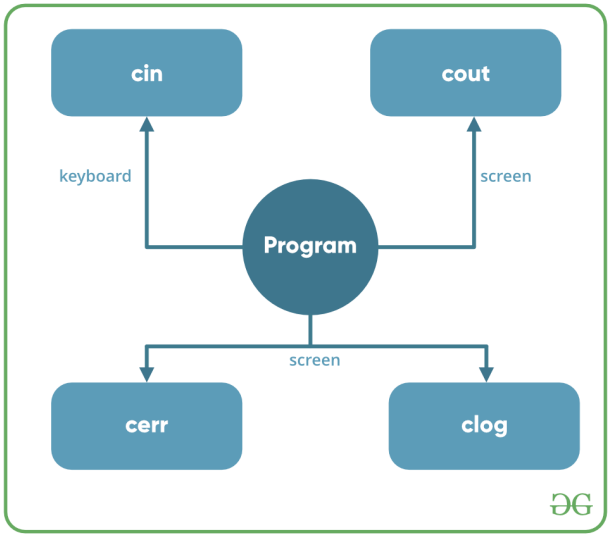
2. Execution Policy of STL Algorithms in Modern C++

Basic Input / Output in C++

Difficulty Level : Easy • Last Updated : 25 Jan, 2023

C++ comes with libraries that provide us with many ways for performing input and output. In C++ input and output are performed in the form of a sequence of bytes or more commonly known as **streams**.

- **Input Stream:** If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.



Header files available in C++ for Input/Output operations are:

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.
2. **iomanip**: iomanip stands for input-output manipulators. The methods declared in these files are used for manipulating streams. This file contains definitions of setw,

setprecision, etc.

3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.
4. **bits/stdc++:** This header file includes every standard library. In programming contests, using this file is a good idea, when you want to reduce the time wasted in doing chores; especially when your rank is time sensitive. To know more about this header file refer this article.

In C++ after the header files, we often use '*using namespace std;*'. The reason behind it is that all of the standard library definitions are inside the namespace std. As the library functions are not defined at global scope, so in order to use them we use *namespace std*. So, that we don't need to write STD:: at every line (eg. STD::cout etc.). To know more refer this article.

The two instances **cout in C++** and **cin in C++** of iostream class are used very often for printing outputs and taking inputs respectively. These two are the most basic methods of taking input and printing output in C++. To use cin and cout in C++ one must include the header file *iostream* in the program.

This article mainly discusses the objects defined in the header file *iostream* like the cin and cout.

- **Standard output stream (cout)**: Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(**<<**).

C++

```
#include <iostream>

using namespace std;
```

```
int main()
{
    char sample[] = "GeeksforGeeks";

    cout << sample << " - A computer science portal for geeks";

    return 0;
}
```

Output:

```
GeeksforGeeks - A computer science portal for geeks
```

Time Complexity: O(1)
Auxiliary Space: O(1)

In the above program, the insertion operator(**<<**) inserts the value of the string variable **sample** followed by the string "A computer science portal for geeks" in the standard output stream **cout** which is then displayed on the screen.

- **standard input stream (cin)**: Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard.
  The extraction operator(**>>**) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keyboard.

C++

```
#include <iostream>
using namespace std;

int main()
{
    int age;

    cout << "Enter your age:";
    cin >> age;
    cout << "\nYour age is: " << age;

    return 0;
}
```

Input :

```
18
```

Output:

```
Enter your age:
Your age is: 18
```

Time Complexity: O(1)
Auxiliary Space: O(1)

The above program asks the user to input the age. The object cin is connected to the input device. The age entered by the user is extracted from cin using the extraction operator(**>>**) and the extracted data is then stored in the variable **age** present on the right side of the extraction operator.

- **Un-buffered standard error stream (cerr)**: The C++ cerr is the standard error stream that is used to output the errors. This is also an instance of the iostream class. As cerr in C++ is un-buffered so it is used when one needs to display the error message immediately. It does not have any buffer to store the error message and display it later.
- The main difference between cerr and cout comes when you would like to redirect output using "cout" that gets redirected to file if you use "cerr" the error doesn't get stored in file.(This is what un-buffered means ..It cant store the message)

C++

```
#include <iostream>

using namespace std;

int main()
{
    cerr << "An error occurred";
    return 0;
}
```

Output:

```
An error occurred
```

Time Complexity: O(1)
Auxiliary Space: O(1)

- **buffered standard error stream (clog)**: This is also an instance of ostream class and used to display errors but unlike cerr the error is first inserted into a buffer and is stored in the buffer until it is not fully filled. or the buffer is not explicitly flushed (using flush()). The error message will be displayed on the screen too.

C++

```
#include <iostream>

using namespace std;

int main()
{
    clog << "An error occurred";

    return 0;
}
```

Output:

```
An error occurred
```

Time Complexity: O(1)
Auxiliary Space: O(1)

Related Articles:

- cout << endl vs cout << "\n" in C++
- Problem with scanf() when there is fgets()/gets()/scanf() after it
- How to use getline() in C++ when there are blank lines in input?
- Cin-Cout vs Scanf-Printf

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Clearing The Input Buffer In C/C++

Difficulty Level : Medium • Last Updated : 30 Oct, 2022

What is a buffer?

A temporary storage area is called a buffer. All standard input and output devices contain an input and output buffer. In standard C/C++, streams are buffered. For example, in the case of standard input, when we press the key on the keyboard, it isn't sent to your program, instead of that, it is sent to the buffer by the operating system, till the time is allotted to that program.

How does it affect Programming?

On various occasions, you may need to clear the unwanted buffer so as to get the next input in the desired container and not in the buffer of the previous variable. For example, in the case of C after encountering "scanf()", if we need to input a character array or character, and in the case of C++, after encountering the "cin" statement, we require to input a character array or a string, we require to clear the input buffer or else the desired input is occupied by a buffer of the previous variable, not by the desired container. On pressing "Enter" (carriage return) on the output screen after the first input, as the buffer of the previous variable was the space for a new container(as we didn't clear it), the program skips the following input of the container.

In the case of C Programming

C

```c
// C Code to explain why not
// clearing the input buffer
// causes undesired outputs
#include <stdio.h>
int main()
{
    char str[80], ch;

    // Scan input from user -
    // GeeksforGeeks for example
    scanf("%s", str);

    // Scan character from user-
    // 'a' for example
    ch = getchar();

    // Printing character array,
    // prints “GeeksforGeeks”)
    printf("%s\n", str);

    // This does not print
    // character 'a'
    printf("%c", ch);

    return 0;
}
```

Input:

```
GeeksforGeeks
a
```

Output:

```
GeeksforGeeks
```

Time Complexity: O(1)

In the case of C++

C++

```cpp
// C++ Code to explain why
// not clearing the input
// buffer causes undesired
```

```
// outputs
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    int a;
    char ch[80];

    // Enter input from user
    // - 4 for example
    cin >> a;

    // Get input from user -
    // "GeeksforGeeks" for example
    cin.getline(ch,80);

    // Prints 4
    cout << a << endl;

    // Printing string : This does
    // not print string
    cout << ch << endl;

    return 0;
}
```

Input:

```
4
GeeksforGeeks
```

Output:

```
4
```

Time Complexity: O(1)

In both of the above codes, the output is not printed as desired. The reason for this is an occupied Buffer. The "\n" character goes remains there in the buffer and is read as the next input.

How can it be resolved?

In the case of C:

1. Using " while ((getchar()) != '\n'); ": Typing "while ((getchar()) != '\n');" reads the buffer characters till the end and discards them(including newline) and using it after the "scanf()" statement clears the input buffer and allows the input in the desired container.

C

```c
// C Code to explain why adding
// "while ( (getchar()) != '\n');"
// after "scanf()" statement
// flushes the input buffer
#include<stdio.h>

int main()
{
    char str[80], ch;

    // scan input from user -
    // GeeksforGeeks for example
    scanf("%s", str);

    // flushes the standard input
    // (clears the input buffer)
    while ((getchar()) != '\n');

    // scan character from user -
    // 'a' for example
    ch = getchar();

    // Printing character array,
    // prints “GeeksforGeeks”)
    printf("%s\n", str);

    // Printing character a: It
    // will print 'a' this time
    printf("%c", ch);

    return 0;
}
```

Input:

```
GeeksforGeeks
a
```

Output:

```
GeeksforGeeks
a
```

Time Complexity: O(n), where n is the size of the string.

2. Using “ fflush(stdin) ”: Typing “fflush(stdin)” after “scanf()” statement, also clears the input buffer but generally it’s use is avoided and is termed to be “undefined” for input

stream as per the C++11 standards.

In the case of C++:

1. Using " cin.ignore(numeric_limits::max(),'\n'); " :- Typing "cin.ignore(numeric_limits::max(),'\n');" after the "cin" statement discards everything in the input stream including the newline.

C++

```
// C++ Code to explain how
// "cin.ignore(numeric_limits
// <streamsize>::max(),'\n');"
// discards the input buffer
#include <iostream>

// for <streamsize>
#include <ios>

// for numeric_limits
#include <limits>
using namespace std;

int main()
{
    int a;
    char str[80];

    // Enter input from user
    // - 4 for example
    cin >> a;

    // discards the input buffer
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    // Get input from user -
    // GeeksforGeeks for example
    cin.getline(str, 80);

    // Prints 4
    cout << a << endl;

    // Printing string : This
    // will print string now
    cout << str << endl;

    return 0;
}
```

Input:

```
4
GeeksforGeeks
```

Output:

```
4
GeeksforGeeks
```

Time Complexity: O(1)

2. Using " cin.sync() ": Typing "cin.sync()" after the "cin" statement discards all that is left in the buffer. Though "cin.sync()" **does not work** in all implementations (According to C++11 and above standards).

C++

```
// C++ Code to explain how " cin.sync();"
// discards the input buffer
#include <ios>
#include <iostream>
#include <limits>

using namespace std;

int main()
{
    int a;
    char str[80];

    // Enter input from user
    // - 4 for example
    cin >> a;

    // Discards the input buffer
    cin.sync();

    // Get input from user -
    // GeeksforGeeks for example
    cin.getline(str, 80);

    // Prints 4
    cout << a << endl;

    // Printing string - this
    // will print string now
    cout << str << endl;

    return 0;
}
```

Input:

```
4
GeeksforGeeks
```

Output:

```
4
```

Time Complexity: O(1)

3. Using " cin >> ws ": Typing "cin>>ws" after "cin" statement tells the compiler to ignore buffer and also to discard all the whitespaces before the actual content of string or character array.

C++

```
// C++ Code to explain how "cin >> ws"
// discards the input buffer along with
// initial white spaces of string

#include<iostream>
#include<vector>
using namespace std;

int main()
{
    int a;
    string s;

    // Enter input from user -
    // 4 for example
    cin >> a;

    // Discards the input buffer and
    // initial white spaces of string
    cin >> ws;

    // Get input from user -
    // GeeksforGeeks for example
    getline(cin, s);

    // Prints 4 and GeeksforGeeks :
    // will execute print a and s
    cout << a << endl;
    cout << s << endl;

    return 0;
}
```

Input:

```
4
GeeksforGeeks
```

Output:

```
4
GeeksforGeeks
```

Time Complexity: O(1)

4.Using " fflush(stdin) ": Typing "fflush(stdin)" after taking the input stream by "cin" statement also clears the input buffer by prompting the '\n' to the nextline literal but generally it is avoided as it is only defined for the C++ versions below 11 standards.

C++

```cpp
// C++ Code to explain working of "flush(stdin);"
// discards the input buffer
#include <cstdio> //fflush(stdin) is available in cstdio header files
#include <ios>
#include <iostream>
using namespace std;

int main()
{
    int value;
    string letters;

    // Enter an integer input from user
    // 98 for example
    cin >> value;

    // Discards the input buffer
    fflush(stdin);

    // Get input from user -
    // GeeksforGeeks for example
    getline(cin, letters);

    // Prints 98
    cout << value << endl;

    // Printing user entered string - this
    // will print string now
    cout << letters << endl;

    return 0;
}
```

```
// This code is contributed by im_impossible_ksv
```

Input:

```
369
geeksforgeeks
```

Output:

```
369
geeksforgeeks
```

Time Complexity: O(1)

This article is contributed by **Manjeet Singh**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

290

Related Articles

1. main Function in C
2. printf in C
3. C Program to Implement Max Heap
4. C Program to Implement Min Heap
5. C++ Error - Does not name a type
6. Execution Policy of STL Algorithms in Modern C++
7. C++ Program To Print Pyramid Patterns
8. Jagged Arrays in C++