Redirecting the Standard Input and Output Streams

In this lesson, we will learn how the standard inputs and outputs of programs can be redirected to files or piped to other programs.

WE'LL COVER THE FOLLOWING

All of the programs that we have seen so far have interacted through `stdin` and `stdout`, the standard input and output streams. Input and output functions like `readf` and `writeln` operate on these streams by default. While using these streams, we assumed that the standard input comes from the keyboard and that the standard output goes to the screen.

We will start writing programs that deal with files in the next lesson. We will see that, just like the standard input and output streams, files are character streams as well; they are used in almost the same way as `stdin` and `stdout`. But before seeing how files are accessed from within programs, we will get familiar with how the standard inputs and outputs of programs can be redirected to files or piped to other programs.

Using `>` to redirect standard output to file

When starting the program from the terminal, typing a `>` character and a file name at the end of the command line redirects the standard output of that program to the specified file. Everything that the program displays to its standard output will be written to that file instead.

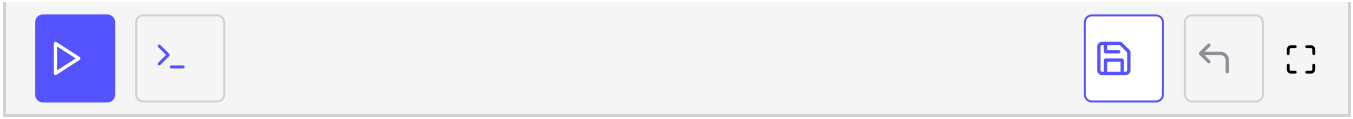
Let's test this with a program that reads a floating point number from its input, multiplies that number by two and displays the result to its standard

output:

```
import std.stdio;

void main() {
    double number;
    readf(" %s", &number);

    writeln(number * 2);
}
```



Printing the result to standard output

If the name of the program is by_two, its output will be written to a file named `by_two_result.txt` when the program is started on the command line, as in the following line:

```
./by_two > by_two_result.txt
```

For example, if we enter 1.2 at the terminal, the result 2.4 will appear in `by_two_result.txt`.

> **Note:** Although the program does not display a prompt like “Please enter a number,” it still expects a number to be entered.

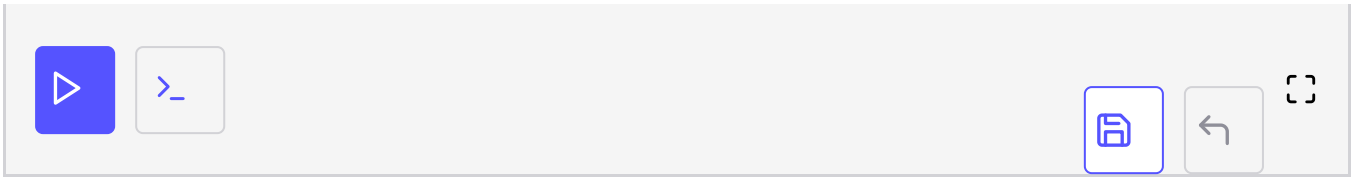
Using `<` to redirect standard input from file

Similar to redirecting the standard output by using the `>` operator, the standard input can be redirected to be read from a file using `<` operator. In this case, the program reads from the specified file instead of from the keyboard.
To test this, let’s use a program that calculates one-tenth of a number:

```
import std.stdio;

void main() {
    double number;
    readf(" %s", &number);

    writeln(number / 10);
}
```



one_tenth

Assuming that the file `by_two_result.txt` still exists and contains 2.4 from the previous output and that the name of the new program is `one_tenth`, we can redirect the new program's standard input from that file as in the following line:

```
./one_tenth < by_two_result.txt
```

This time the program will read from `by_two_result.txt` and display the result to the terminal as 0.24.

Redirecting both standard streams

The operators `>` and `<` can be used at the same time:

```
./one_tenth < by_two_result.txt > one_tenth_result.txt
```

This time the standard input will be read from `by_two_result.txt`, and the standard output will be written to `one_tenth_result.txt`.

> **Note:** `by_two_result.txt` acts as an intermediary between the two programs; `by_two` writes to it and `one_tenth` reads from it.

Piping programs with operator |

The `|` operator pipes the standard output of the program that is on its left-hand side to the standard input of the program that is on its right-hand side without the need for an intermediary file. For example, when programs by_two, and one-tenth, discussed above are piped together as shown below, they collectively produce one-fifth of the input:

```
./by_two | ./one_tenth
```

First `by_two` reads a number from its input. Then `by_two` writes the result to

First, `by_two` reads a number from its input. Then `by_two` writes the result to its standard output. This result of `by_two` will appear on the standard input of `one_tenth`, which in turn will calculate and write one-tenth of the result to its standard output.

> **Note**: Existing programs, without their source code being changed, can be made to print to files instead of the screen and be read from files instead of the keyboard. Although these features are not directly related to programming languages, they are useful tools that are available in nearly all modern shells.

The next lesson explores file handling concepts such as creating, reading from and writing to files in more detail.