CS332 Exercise I

Aidan Rutherford

There are many programming languages, each with their own benefits and drawbacks, better suited for certain tasks. The purpose of this paper is to demonstrate the use of these languages through simple input/output programs in old and new languages such as Fortran, Java, and C, all implemented in a Linux environment.

In each language, the general process was to print the "Hello" followed by the programming language that the instructions were written in. That was followed by a `for` loop that iterated the strings entered in the command line and printed them to the console using a user-defined function.

The first language demonstrated is C. The following code was for the hello program was copied below.

```c
#include <stdio.h>

void hello(char* name);

int main(int argc, char* argv[]) {
    int i = 0;
    fputs("Hello C!\n", stdout);
    for(i = 0; i < argc; ++i)
        hello(argv[i]);
    return 0;
}

void hello(char* name) {
    printf("Hello %s\n", name);
}
```

Figure 1. Hello example in C

For the example in C and C++, to pass values via the command line requires `argc` and `argv[]` as parameters to be passed into `main`. The first parameter `argc`, is the number of strings passed, and `argv[]` is the array of strings that was passed. The code was compiled with `gcc -o hello_c hello.c` which creates an executable file with the name `hello_c`. The executable file name entered in the command line along with the text `Aidan`, `Peter` and `Steve` produced the following output:

```
[ruthera1@prclab1:~/cs332/exercise1]$ hello_c Aidan Peter Steve
Hello C!
Hello hello_c
Hello Aidan
Hello Peter
Hello Steve
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 2. Hello example output with C

The C++ used two versions of the same program, one with templates and one without. The code samples copied are shown below:

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

void hello(string name);

int main(int argc, char* argv[]) {
    cout << "Hello C++!" << endl;
    for(int i = 0; i < argc; ++i)
        hello(string(argv[i]));
    return 0;
}

void hello(string name) {
    cout << "Hello " << name << endl;
}
```

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

template <typename T> void hello(T name);

int main(int argc, char* argv[]) {
    cout << "Hello C++!" << endl;
    for(int i = 0; i < argc; ++i)
        hello(string(argv[i]));
    hello(99);
    return 0;
}

template <typename T>
void hello(T name) {
    cout << "Hello " << name << endl;
}
```

Figure 3. Hello demonstration in C++ with templates (right) and without

Something unique about the example that uses templates is that the function accepts a value of type `T` which allows the function to accept any data type and still perform as expected regardless of the type passed. The command line instruction to compile each C++ file were `g++ -o hello_cpp hello.cpp` and `g++ -o hello_cpp_t hello_t.cpp` which made executable files for each program whose output is shown below.

```
[ruthera1@prclab1:~/cs332/exercise1]$ hello_cpp_t Aidan Carl Oscar
Hello C++!
Hello hello_cpp_t
Hello Aidan
Hello Carl
Hello Oscar
Hello 99
[ruthera1@prclab1:~/cs332/exercise1]$ hello_cpp Aidan Stan Glenn
Hello C++!
Hello hello_cpp
Hello Aidan
Hello Stan
Hello Glenn
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 4. Hello example output with C++ with templates (top) and without

The third language to be used was Python. The sample code is viewable below.

```
#!/usr/bin/python
import sys

def hello(name):
    print("Hello " + name)

print("Hello Python!")
for i in range(1, len(sys.argv)):
    hello(sys.argv[i])
```

Figure 5. Hello example in Python

The Python code follows the same structure as the other examples, except for `import sys` which allows the use of `argc` and `argv` since there is no `main` function to start at. Most of Python's capabilities comes from libraries which are typically written in C. Also, since Python is an interpreted language, to run it in Linux only requires the line `python <filename>` or in this case `python hello.py`. However, like compiling C or C++ with the `-o` prefix which creates an executable, the same can be done by entering `chmod +x <filename>` in the command line and including the line `#!/usr/bin/python` at the top of the script. The output of the Python script is shown below.

```
[ruthera1@prclab1:~/cs332/exercise1]$ hello.py Aidan George Grew
Hello Python!
Hello Aidan
Hello George
Hello Grew
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 6. Hello example outputs with Python

The next demonstration was done in Java as seen in the code sample below.

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Java!");
        Hello greeter = new Hello();
        for(int i = 0; i < args.length; ++i)
            greeter.greet(args[i]);
    }

    public void greet(String name) {
        System.out.println("Hello " + name + "!");
    }
}
```

Figure 7. Hello example in Java

In Java, all of the code goes inside a class, even the main function and everything is done inside the class declaration. Additionally, rather than having a typical print function, Java relies a series of libraries for everything. For example, the general print function is `system.out.println()`. To compile in the Linux environment, uses the `javac` command, or in this case `javac Hello.java`. Something else to note is that the name of the class must match the filename for the code to compile. Attempting to compile with the file name `hello.java` when the class name is `Hello` results is an error message. After compilation, another file with a `.class` extension is created. To run a Java program uses the line `java <class name>` followed by any particular command line arguments. For this example, the instruction to run was `java Hello Aidan Steve Mike`. The output can be seen below.

```
[ruthera1@prclab1:~/cs332/exercise1]$ java Hello Aidan Frank Jim
Hello Java!
Hello Aidan!
Hello Frank!
Hello Jim!
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 6. Hello example output in Java

The next demonstration was done in Fortran and the source code is displayed below.

```
PROGRAM HELLO
CHARACTER :: ARG*32

PRINT *, 'Hello Fortran!'
DO I = 1, IARGC()
CALL GETARG(I, ARG)
CALL GREET(ARG)
END DO
END

SUBROUTINE GREET(s)
CHARACTER :: S*32
PRINT *, 'Hello ' // s
END
```

Figure 7. Hello example in Fortran

Some things to note about Fortran is that the keyword `PROGRAM` is the equivalent to main and functions are referred to as subroutines. Fortran also uses a `do` loop rather than a `for` loop. Fortran is a compiled language which was compiled with the line `gfortran -o hello_f hello.f90` which makes an executable file. The output can be seen below.

```
[ruthera1@prclab1:~/cs332/exercise1]$ hello_f Aidan Robert Francis
 Hello Fortran!
 Hello Aidan
 Hello Robert
 Hello Francis
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 8. Hello example output with Fortran

Perl is a common language used for text processing, and is referred to as a Swiss Army lanaguage for immense power. Similar to PHP, variable names begin with `$` and functions are referred to as subroutines. Although Perl code is very difficult to read, it is very robust language given the two code samples below which both do the same thing.

```
[ruthera1@prclab1:~/cs332/exercise1]$ cat hellov2.pl
#!/usr/bin/perl

use strict;
use warnings;

sub hello {
    print("Hello " . $_[0] . "\n");
}

print("Hello Perl!\n");

while (<>) {
    my @words = split(/\s+/);
    foreach my $name (@words) {
        hello("$name");
    }
}
[ruthera1@prclab1:~/cs332/exercise1]$
```

```
#!/usr/bin/perl

use strict;
use warnings;
print("Hello Perl!\n");
while (<>) {
    print("Hello " . join("\nHello ", split(/\s+/)) . "\n");
}
```

Figure 9. Hello examples in Perl

To run the script, one could enter `perl <filename>` in the command line. However, to make an executable script the line `#!/usr/bin/perl` was added to the top and `chmod +x hello.pl` was used make an executable script. Perl does not accept command line arguments, so the values were entered after while the executable as seen below.

```
[ruthera1@prclab1:~/cs332/exercise1]$ hello.pl
Hello Perl!
Aidan
Hello Aidan
Greg
Hello Greg
Jeff
Hello Jeff
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 10. Hello example Output with Perl

The final language to be demonstrated was R, primarily used for statistical analysis. For functions in R, there is a definition, which is assigned to a name, which is seen in the code example below.

```
hello <- function(name) {
    cat("Hello", name, "\n")
}

main <- function() {
    cat("Hello World\n")
    args <- commandArgs(trailingOnly = TRUE)
    for(i in 1:length(args)) {
        hello(args[i])
    }
}

main();
```

Figure 11. Hello example in R

To run an R script, `R` had to be entered into the command line which brought up a new command line specific to the R environment. To actually run the script was `Rscript hello.r Aidan George Jerry` which actually failed in the R environment but worked in the Linux environment. Additionally, not entering `R` prior to using the `Rscript` command caused `Rscript` to do nothing. Despite the difficulty this result was eventually displayed below.

```
Hello R!
Hello Aidan
Hello George
Hello Jerry
[ruthera1@prclab1:~/cs332/exercise1]$
```

Figure 12. Hello example output in R

This exercise demonstrated the use of many languages, many of which are used for many different things, from general purpose programming, to text processing, to statistical analysis. Despite the oddities of each language whether it is syntax or efficiency, they all can serve a purpose depending on the application, some of which are far better at somethings than others.