Project Name	Hopeful - A First Programming Language
Author	Gillian Mullen
Supervisor	David Sinclair
Document Title	Language Guide
Summary	This project, Hopeful - A First Programming Language, will involve the development of a programming language. This language will be aimed at beginning programmers in an undergraduate setting. It will have a simple and clean syntax that will allow students to focus on the fundamentals of programming, instead of a complicated syntax.

Running Hopeful

Hopeful programs are saved under the file extension .hope. For example, hello_world.hope. Hopeful programs can be run with the following command:

```
java -jar HOPEFUL.jar file_name.hope
```

This command will both compile the program and execute the compiled program, and will print any output onto the terminal.
In order for the code to execute, LLVM is required on that machine, which can be found at the following link: http://releases.llvm.org/download.html.

Basic Syntax

All statements in Hopeful end in a semicolon. For example:

```
a = b + 1;
```

The main method may be contained within

```
main {
   //insert code here
}
```

but this is not required. Indentation is not required within the main method, functions, if statements and while loops.
Comments in Hopeful may be contained within /* */ and may span multiple lines. They may also begin with //, but are terminated by a new line character.

Variables

In Hopeful, variables must first be declared before they can be used. For example:

```
int a;
a = 1;
```

Variables may be of type int, string, or boolean. They are statically typed. This means that a variable declared as one type cannot have a value of another type. For example, the statements:

```
int a;
a = “hello”;
```

would be invalid because a was declared as an int, not a string.
Due to an issue with the scoping of variables, you may not have variables with the same names in multiple functions and the main method.

Types

Integers

The integer numbers (e.g. 2, 99, -4) are of the type int, and function much like integers in other languages. Expression syntax is also similar to other languages. The operators used are +, -, *, /, and %. Integers are represented by a sequence of one or more digits, meaning 0 to 9. Integers may begin with a minus sign, e.g. -123. Integers may not start with any leading 0s, e.g. 01.

Booleans

Booleans may have the value of either true or false, and are of type boolean. They also use the operators & and |.

Strings

String values is a sequence of characters and digits encased in double quotes, and are of type string.

Output

The print() function will write its arguments to the command line. Types of arguments include integers, strings, variables names, and integer expressions.

```
print(1);
print("hello world");
print(a); // variable name
print(3 * 8 / 4); // integer expression
```

While Booleans can be printed in Hopeful, true and false are represented by 1 and 0 respectively while printing onto the command line.

Control Flow

If Statements

These statements in Hopeful take on the following form:

```
if(condition) {
    //insert code here
}
else {
    //more code
}
```

The code inside the if block will execute if the condition is true, otherwise the else code will execute.

While Loops

These statements in Hopeful take on the following form:

```
while(condition) {
    //insert code here
}
```

The statements inside the loop will execute as long as the condition is true.

Conditions

The standard comparison operators are present in Hopeful: < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), == (equal to), and != (not equal to). The operators may only be used on integers or integer variables.

As with expressions, conditions require spaces between integers and operators, e.g. 4 < 5 instead of 4<5.

Functions

Functions in Hopeful take on the form:

```
def return_type function_name(parameter_list) {
    //insert code here;
    return (integer or string or boolean or variable);
}
```

where the parameter list is comma separated and each parameter takes the form of parameter_type parameter_name.

```
def void add(int a, int b) {
    print(a + b);
    return;
}
```

Functions may be called by function_name(argument_list), where the argument list is comma separated.

```
double(2, 1);
```

Code Examples

```
// program that prints “hello world”
print(“hello world”);

// program that prints the total points scored by a GAA team
int goals;
int points;
goals = 2;
points = 10;
print(goals * 3 + points); // result = 16

// program that prints pass or fail depending on whether a grade is above of below 40
int grade;
grade = 40;
if(grade > 40) {
    print(“pass”);
}
else {
    print(“fail”);
}

// program that prints the square of a number from 1 up to some integer n
int n;
int i;
n = 10;
i = 1;
while(i < n) {
    print(i * i);
    i = i + 1;
}

// program that contains a function that takes an integer and doubles it
def int double(int a) {
    return (a * 2);
}

main {
    int n;
    n = double(2); // result = 4
    print(n);
}
```