# BASIC

BASIC is a programming language designed at Dartmouth in the 1960's (really!) for learning. It stands for Beginners All-Purpose Symbolic Instruction Code. It was very popular in the 1980's – it came pre-installed on every computer.

BASIC (at least the version that we will be using) is **not** object oriented; it is procedural. This is a simple model of computation – everything is a number or a string. For simple programs, this is actually a very nice model – it is very easy to get started.

For example:

```
F=72
C = 5*(F-32)/9
PRINT C
```

While there were standards published, there were dozens of implementations, many of which varied greatly. For this course, we will use the set of features defined in this document.

## Data Types

Simple variables do not need to be pre-defined! In Java, you have to declare a variable before using it. Not in BASIC. This has advantages **and** disadvantages…

They are typed by their ending character:

$ = string  % = float  any other ending = integer

Example: myString$, percentOfPeople%, count

## Structure

BASIC doesn't require any setup (like Java with defining a class). It is literally just a list of commands for the computer to execute. This makes it easy to get started, but it makes growing a program more difficult. Classic BASIC has a number for each line of code. An example:

```
10 PRINT "Hello!"
20 GOTO 10
```

This, you will recognize, is an infinite loop. This is one place where we can improve – we will have "labels", like C. A label is a name followed by a colon. It must be the first item on a line. We can then reference that line from anywhere else in our program. Translating the above example:

```
beginning: PRINT "Hello!"
GOTO beginning
```

Labels are more intuitive than line numbers; line numbers are also terrible when you want to edit your program and insert new lines in the middle. That is why I numbered the lines by 10 above.

In the version of BASIC that we are implementing, there are no user-defined functions. There are built-in functions, but you cannot make your own. You can have subroutines. A subroutine is like a function, but it doesn't have parameters or return values. It uses global variables for communication. **Every variable is global!**

Example:

```
FtoC: C = 5*(F-32)/9
RETURN

F=72
GOSUB FtoC
PRINT C
```

## Flow Control

IF expression THEN label

If expression is true, GOTO label

Example: `IF x<5 THEN xIsSmall`

FOR variable = initialValue TO limit STEP increment

NEXT variable

Sets variable to initialValue, loops by adding increment to variable on each loop until limit is hit/surpassed. Note that the "STEP" and increment is optional – the step is assumed to be 1 if it is left off. NEXT variable marks the end of the "block"

Example:

```
FOR A = 0 TO 10 STEP 2
PRINT A
NEXT A
```

WHILE expression label

Loops from the current line to label until the expression is not true.

```
WHILE x < 5 endWhileLabel
x = x + 1
endWhileLabel:
```

END ends the program. We need this because we might want to have subroutines/functions. How can we make sure that we don't run those at the end of the program? By ending the program early!

## Dealing with Data

DATA – defines a list of constant data that can be accessed with READ

READ – reads the next item from DATA

Example:

```
DATA 10,"mphipps"
READ a, a$
```

If the data types don't match correctly (like the $ was on the wrong variable), that is a runtime error.

INPUT – expects a string, then any number of variables. Prints the string, then waits for the user to enter the inputs, comma separated.

Example:

```
INPUT "What is your name and age?", name$, age
PRINT "Hi ", name$, " you are ", age, " years old!"
```

PRINT – prints any number of values, separated by a comma

Example:

```
PRINT "hello. 5 + 3 = ", 5+3, " how do you like that ", name$, "?"
```

## Built-in Functions

RANDOM() – returns a random integer

LEFT$(string, int) – returns the leftmost N characters from the string

RIGHT$(string, int) – returns the rightmost N characters from the string

MID$(string,int, int) – returns the characters of the string, starting from the 2nd argument and taking the 3rd argument as the count; MID$(“Albany”,2,3) = “ban”

NUM$(int or float) – converts a number to a string

VAL(string) – converts a string to an integer

VAL%(string) – converts a string to a float