Chapter 1

Variables

1.1 C++ Variables

1.1.1 C++ Variables

Variables are containers for storing data values.

In C++, there are different types of variables (defined with different keywords), for example:

int	stores integers (whole numbers), without decimals, such as 123 or -123
double	stores floating point numbers, with decimals, such as 19.99 or -19.99
char	stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
string	stores text, such as "Hello World". String values are surrounded by double quotes
bool	stores values with two states: true or false

1.1.2 Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

```
type variable = value;
```

Listing 1.1: C++ example

Where type is one of C++ types (such as int), and variable is the name of the variable (such as x or myName). The equal sign is used to assign values to the variable.

To create a variable that should store a number, look at the following example:

Create a variable called myNum of type int and assign it the value 15:

```cpp
int myNum = 15;
cout << myNum;
```

Listing 1.2: C++ example

You can also declare a variable without assigning the value, and assign the value later:

```cpp
int myNum;
myNum = 15;
cout << myNum;
```

Listing 1.3: C++ example

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

```cpp
int myNum = 15;  // myNum is 15
myNum = 10;  // Now myNum is 10
cout << myNum;  // Outputs 10
```

Listing 1.4: C++ example

1.1.3 Constants

However, you can add the const keyword if you don't want others (or yourself) to override existing values (this will declare the variable as "constant", which means unchangeable and read-only):

```cpp
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

Listing 1.5: C++ example

1.1.4 Other Types

A demonstration of other data types:

```cpp
int myNum = 5;               // Integer (whole number without
   decimals)
double myFloatNum = 5.99;    // Floating point number (with
   decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;       // Boolean (true or false)
```

Listing 1.6: C++ example

1.1.5 Display Variables

The cout object is used together with the ¡¡ operator to display variables.

To combine both text and a variable, separate them with the ¡¡ operator:

```
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

Listing 1.7: C++ example

1.1.6 Add Variables Together

To add a variable to another variable, you can use the + operator:

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

Listing 1.8: C++ example

1.1.7 Declare Many Variables

To declare more than one variable of the same type, you can use a comma-separated list:

```
int x = 5, y = 6, z = 50;
cout << x + y + z;
```

Listing 1.9: C++ example

1.1.8 C++ Identifiers

All C++ variables must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code.

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (like C++ keywords, such as int) cannot be used as names

1.2 Python Variables

1.2.1 Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```python
x = 5
y = "John"
print(x)
print(y)
```

Listing 1.10: Python example

Variables do not need to be declared with any particular type and can even change type after they have been set.

```python
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Listing 1.11: Python example

String variables can be declared either by using single or double quotes:

```python
x = "John"
# is the same as
x = 'John'
```

Listing 1.12: Python example

1.2.2 Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Remember that variable names are case-sensitive

1.2.3 Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```python
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Listing 1.13: Python example

And you can assign the same value to multiple variables in one line:

```python
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Listing 1.14: Python example

1.2.4 Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

```python
x = "awesome"
print("Python is " + x)
```

Listing 1.15: Python example

You can also use the + character to add a variable to another variable:

```python
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

Listing 1.16: Python example

For numbers, the + character works as a mathematical operator:

```python
x = 5
y = 10
print(x + y)
```

Listing 1.17: Python example

If you try to combine a string and a number, Python will give you an error:

```python
x = 5
y = "John"
print(x + y)
```

Listing 1.18: Python example

1.2.5 Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function, and use it inside the function

```python
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

Listing 1.19: Python example

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```python
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

Listing 1.20: Python example

1.2.6 The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

```python
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Listing 1.21: Python example

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```python
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Listing 1.22: Python example

1.3 Ada Variables

1.3.1 Declaring a Variable

To declare a variable, in the line under procedure, use the following formula:

```ada
VariableName : DataType;
```

Listing 1.23: Ada example

The declaration starts with the name of the variable:

- The name of a variable must be in one word
- It must start with a letter
- It can include a combination of letters, digits, and underscores
- It must not contain special characters

The above formula is used to declare one variable. You can use it to declare various variables, each on its own line. This would be:

```
VariableName1, VariableName2 : DataType1;
```

Listing 1.24: Ada example

1.3.2 Initializing a Variable

Initializing a variable consists of assigning a value to it before using it. You have two options.

To initialize a variable when declaring it, after the name of the variable, type := followed by an appropriate value. This would be done as follows:

```
VariableName : DataType := Value;
```

Listing 1.25: Ada example

The option consists of assigning a value after declaring it. This can be done as follows:

```
VariableName : DataType
begin
    VariableName:= Value;
end
```

Listing 1.26: Ada example

1.3.3 The declare Keyword

We saw that, to declare a variable, you can use the section under the procedure. As another option, in the body of the procedure, use the declare keyword, then declare the variable(s). Before using the variable(s), start a section with begin, include the necessary code, and end it with end;. Here is an example:

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Exercise is

begin
    declare
        -- Declarations
    begin
        -- Initializations
    end;
end Exercise;
```

Listing 1.27: Ada example

1.3.4 Displaying the Value of a Variable

One way you can use a variable consists of displaying its value to the user. To do this, you can use Put_Line(). If the variable represents a word or sentence (a string), you can write the name of the variable in the parentheses. Otherwise, we will see other ways of displaying it.

Characters

A character is a a letter, a symbol, or a digit. To declare a variable that can hold a variable, use the character keyword. To initialize it, include the value is single-quotes. Here is an example:

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Welcome is
   gender : character := 'M';
begin
   Put_Line("Gender = " & character'image(gender));
end Welcome;
```

Listing 1.28: Ada example

Strings

A string is a combination of characters. To represent strings, Ada uses the String data type. When declaring the variable, to initialize it, include its value in double-quotes.

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Exercise is
   sentence : String := "Welcome to the wonderful world of Ada
   programming!";

begin

end Exercise;
```

Listing 1.29: Ada example

To declare a string variable, in the parentheses of Put_Line(), include the name of the variable. Here is an example:

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Exercise is
   sentence : String := "Welcome to the wonderful world of Ada
   programming!";

begin
    Put_Line(sentence);
end Exercise;
```

Listing 1.30: Ada example

1.4 Pascal

1.4.1 Variable Declaration in Pascal

All variables must be declared before we use them in Pascal program. All variable declarations are followed by the var keyword. A declaration specifies a list of variables, followed by a colon (:) and the type. Syntax of variable declaration is -

```
var
variable_list : type;
```

Listing 1.31: Pascal example

Here, type must be a valid Pascal data type including character, integer, real, boolean, or any user-defined data type, etc., and variable_list may consist of one or more identifier names separated by commas. Some valid variable declarations are shown here -

```
var
age, weekdays : integer;
taxrate, net_income: real;
choice, isready: boolean;
initials, grade: char;
name, surname : string;
```

Listing 1.32: Pascal example

1.4.2 Variable Initialization in Pascal

Variables are assigned a value with a colon and the equal sign, followed by a constant expression. The general form of assigning a value is -

```
variable_name := value;
```

Listing 1.33: Pascal example

By default, variables in Pascal are not initialized with zero. They may contain rubbish values. So it is a better practice to initialize variables in a program. Variables can be initialized (assigned an initial value) in their declaration. The initialization is followed by the var keyword and the syntax of initialization is as follows -

```
var
variable_name : type = value;
```

Listing 1.34: Pascal example

Some examples are -

```
age: integer = 15;
taxrate: real = 0.5;
grade: char = 'A';
name: string = 'John Smith';
```

Listing 1.35: Pascal example

Let us look at an example, which makes use of various types of variables discussed so far -

```
Live Demo
program Greetings;
const
message = ' Welcome to the world of Pascal ';

type
name = string;
var
firstname, surname: name;

begin
   writeln('Please enter your first name: ');
   readln(firstname);

   writeln('Please enter your surname: ');
   readln(surname);

   writeln;
   writeln(message, ' ', firstname, ' ', surname);
end.
```

Listing 1.36: Pascal example

1.5 Lisp

In LISP, each variable is represented by a symbol. The variable's name is the name of the symbol and it is stored in the storage cell of the symbol.

1.5.1 Global Variables

Global variables have permanent values throughout the LISP system and remain in effect until a new value is specified.

Global variables are generally declared using the defvar construct.

```
(defvar x 234)
(write x)
```

Listing 1.37: Lisp example

Since there is no type declaration for variables in LISP, you directly specify a value for a symbol with the setq construct.

```
->(setq x 10)
```

Listing 1.38: Lisp example

The above expression assigns the value 10 to the variable x. You can refer to the variable using the symbol itself as an expression.

1.5.2 Example

Create new source code file named main.lisp and type the following code in it.

```lisp
(setq x 10)
(setq y 20)
(format t "x = ~2d y = ~2d ~%" x y)

(setq x 100)
(setq y 200)
(format t "x = ~2d y = ~2d" x y)
```

Listing 1.39: Lisp example

1.5.3 Local Variables

Like the global variables, local variables can also be created using the setq construct.
There are two other constructs - let and prog for creating local variables.

Example

```lisp
(let ((y 1)
      (z y))
      (list y z))
```

Listing 1.40: Lisp example

Result → (1 2)

Example

```lisp
(let ((str "Hello, world!"))
      (string-upcase str))
```

Listing 1.41: Lisp example

Result → "HELLO, WORLD!"

1.6 Variables - Summary

1.6.1 C

Local variables are generally called auto variables in C. Variables must be declared before use. The declaration of a variable, without assigning a value takes the form

$< typename >< variablename >$;

Some common types are: char, short, int, long, float, double and unsigned.

```c
int j;
```

Listing 1.42: C example

Multiple variables may be defined in a single statement as follows:

```c
double double1, double2, double3;
```

Listing 1.43: C example

It is possible to initialize variables with expressions having known values when they are defined. The syntax follows the form

$< typename >< variablename >=< initializingexpression >$;

```c
short b1 = 2500;
long elwood = 3*BSIZE, jake = BSIZE -2;
```

Listing 1.44: C example

Strings in C are arrays of char terminated by a 0 or NULL character. To declare space for a string of up to 20 characters, the following declaration is used.

```c
char mystring[21];
```

Listing 1.45: C example

1.6.2 C++

Much like C, C++ variables are declared at the very start of the program after the headers are declared. To declare a as an integer you say: the type of variable; then the variable followed by a semicolon ";".

```cpp
int a;
```

Listing 1.46: C++ example

1.6.3 Python

Names in Python are not typed .

```python
# these examples, respectively, refer to integer, float, boolean, and string objects
example1 = 3
example2 = 3.0
example3 = True
example4 = "hello"

# example1 now refers to a string object.
example1 = "goodbye"
```

Listing 1.47: Python example

1.6.4 Ada

```ada
Name: declare   -- a local declaration block has an optional name
   A : constant Integer := 42;  -- Create a constant
   X : String := "Hello"; -- Create and initialize a local variable
   Y : Integer;           -- Create an uninitialized variable
   Z : Integer renames Y: -- Rename Y (creates a view)
   function F (X: Integer) return Integer is
     -- Inside, all declarations outside are visible when not hidden: X, Y, Z are global with respect to F.
     X: Integer := Z;  -- hides the outer X which however can be referred to by Name.X
   begin
     ...
   end F;  -- locally declared variables stop to exist here
begin
   Y := 1; -- Assign variable
   declare
     X: Float := -42.0E-10;  -- hides the outer X (can be referred to Name.X like in F)
   begin
     ...
   end;
end Name; -- End of the scope
```

Listing 1.48: Ada example

1.6.5 Pascal

```
var
  i: Integer;
  s: string;
  o: TObject;
begin
  i := 123;
  s := 'abc';
  o := TObject.Create;
  try
    // ...
  finally
    o.Free;
  end;
end;
```

Listing 1.49: Pascal example

1.6.6 Lisp

Special variables may be defined with defparameter. Special variables are wrapped with asterisks (called 'earmuffs').

```
(defparameter *x* nil "nothing")
```

Listing 1.50: Lisp example

We may also use defvar, which works like defparameter except that defvar won't overwrite the value of the variable that has already been bound.

```
(defvar *x* 42 "The answer.")
```

Listing 1.51: Lisp example

For local varibles, we use let:

```
(let ((jenny (list 8 6 7 5 3 0 9))
      hobo-joe)
      (apply #'+ jenny))
```

Listing 1.52: Lisp example

We use setf to modify the value of a symbol.

```
(setf *x* 625)
```

Listing 1.53: Lisp example

We can also modify multiple symbols sequentially:

```
(setf *x* 42 *y* (1+ *x*))
```

Listing 1.54: Lisp example

Result → 43

We can use psetf to set variables in parallel:

```
(setf *x* 625)
(psetf *x* 42 *y* (1+ *x*))
```

Listing 1.55: Lisp example

1.7 Pointers

1.7.1 C++

The following code creates a pointer to an int variable

```cpp
int var = 3;
int *pointer = &var;
```

Listing 1.56: C++ example

Access the integer variable through the pointer:

```cpp
int v = *pointer; /* sets v to the value of var (i.e. 3) */
*pointer = 42; /* sets var to 42 */
```

Listing 1.57: C++ example

Change the pointer to refer to another object

```cpp
int othervar;
pointer = &othervar;
```

Listing 1.58: C++ example

Change the pointer to not point to any object

```cpp
pointer = NULL; /* needs having stddef.h included */
```

Listing 1.59: C++ example

or

```cpp
pointer = 0; /* actually any constant integer expression
   evaluating to 0 could be used, e.g. (1-1) will work as well */
```

Listing 1.60: C++ example

or

```cpp
pointer = (void*)0; /* C only, not allowed in C++ */
```

Listing 1.61: C++ example

Get a pointer to the first element of an array:

```cpp
int array[10];
pointer = array;
/* or alternatively: */
pointer = &array[0];
```

Listing 1.62: C++ example

Move the pointer to another object in the array

```cpp
pointer += 3; /* pointer now points to array[3] */
pointer -= 2; /* pointer now points to array[1] */
```

Listing 1.63: C++ example

1.7.2 Python

Python does not have pointers and all Python names (variables) are implicitly references to objects. Python is a late-binding dynamic language in which ”variables” are untyped bindings to objects. (Thus Pythonistas prefer the term name instead of ”variable” and the term bind in lieu of ”assign”).

```python
# Bind a literal string object to a name:
a = "foo"
# Bind an empty list to another name:
b = []
# Classes are "factories" for creating new objects: invoke class name as a function:
class Foo(object):
    pass
c = Foo()
# Again, but with optional initialization:
class Bar(object):
    def __init__(self, initializer = None)
        # "initializer is an arbitrary identifier, and "None" is an arbitrary default value
        if initializer is not None:
           self.value = initializer
d = Bar(10)
print d.value
# Test if two names are references to the same object:
if a is b: pass
# Alternatively:
if id(a) == id(b): pass
# Re-bind a previous used name to a function:
def a(fmt, *args):
    if fmt is None:
        fmt = "%s"
     print fmt % (args)
# Append reference to a list:
b.append(a)
# Unbind a reference:
del(a)
# Call (anymous function object) from inside a list
b[0]("foo")  # Note that the function object we original bound to the name "a" continues to exist
             # even if its name is unbound or rebound to some other object.
```

Listing 1.64: Python example

1.7.3 Ada

In Ada pointer types are called access types.

```
type Int_Access is access Integer;
Int_Acc : Int_Access := new Integer'(5);
```

Listing 1.65: Ada example

1.7.4 Pascal

Delphi (Object Pascal) fully supports both typed and untyped pointers.
Simple untyped pointer variable:

```
pMyPointer : Pointer ;
```

Listing 1.66: Pascal example

Simple pointer to a predefined type:

```
pIntPointer : ^Integer ;
```

Listing 1.67: Pascal example

A pointer to a Record. This is the equivalent to a Struct in C

```
MyRecord = Record
            FName : string[20];
            LName : string[20];
           end;

pMyRecord : ^MyRecord ;
```

Listing 1.68: Pascal example

Dereferencing a Pointer -
Pointers are dereferenced using the caret ”ˆ symbol.

```
IntVar := pIntPointer^ ;
```

Listing 1.69: Pascal example

1.7.5 Lisp

Listing 1.70: Lisp example