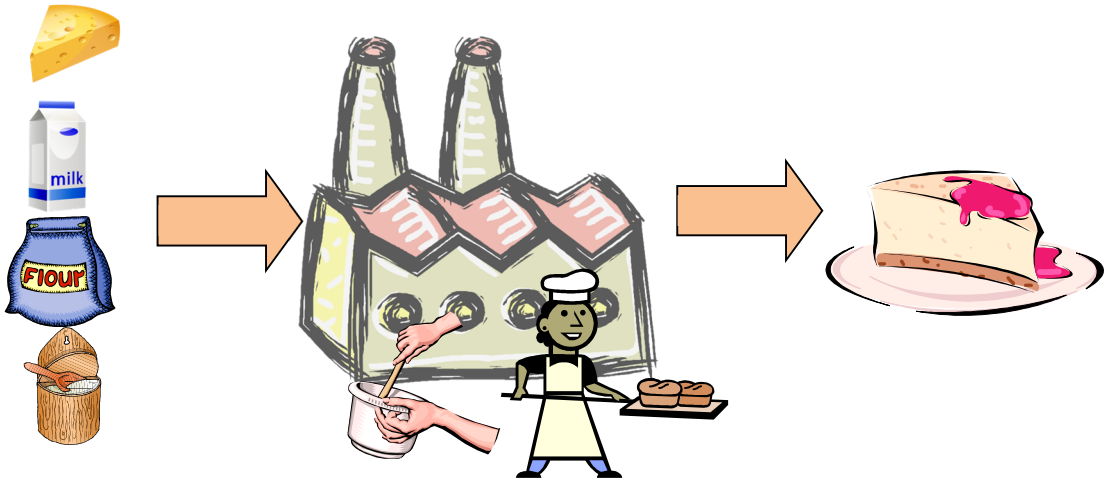
Level 2: Functions

Consider a cheesecake factory. Cheese, milk, flour, sugar etc… are supplied to the factory; factory bakes mixing them and produces cheesecake. **Functions** are like factories; they get *input*, makes some *operations*, and produce *output*.



Input: Cheese, milk, flour, sugar	→	**Factory** *Operations*: Mix and bake	→	*Output*: Cheesecake

For instance, suppose we have a function called power. It receives two values *x* and *y* then returns the value x^y. We can call the function with the values 10 and 3, and put the return value of function in a variable *pow* as follows:

```
pow = Power(10,3);
```

Here, *Power* function will return $10^3 = 1000$ and this value is put in the variable *pow*. We can write the entire program including the *Power* function as follows:

```
1.   #include <iostream>
2.   using namespace std;
3.
4.   int Power(int x, int y)
5.   {
6.     int prod = 1;
7.     for (int i = 1; i <= y; ++i)
8.       prod *= x;
9.     return prod;
10.  }
11.
12.  int main ()
13.  {
14.    int pow = Power(10, 3);
15.    cout << pow << endl;
16.  }
```

The function has to be defined above *main*. In this code, *Power* function is defined between lines 4 and 10. Input and output types, and the name of the function are defined in line 4: it receives to integers, x and y, and returns an integer value as indicated just before 'power'. We write the operations in the body of the function which is between the braces; lines 6 to 9.

Note: The names of the functions are case sensitive as it is in the variables.

Example: Write a function that returns the number of letters in a given string.

Solution:

```
int NumLetters(string s)
{
  int count = 0;
  for (int i = 0; i < s.length(); ++i)
    if ((s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= 'a' && s[i] <= 'z'))
      ++count;
  return count;
}
```

This code checks all the letters in string *s* within the `for` loop. If the current character is between 'A' and 'Z' (uppercase) or 'a' and 'z' (lowercase) then increments *count*.

Exercise 1: Write a function named *UpperCase* that converts all letters to uppercase in a given string. For instance,

```
cout << UpperCase("I8-10caKes!") << endl;
```

will write "`I8-10CAKES!`" on the screen.

Exercise 2: Write a function named *SameWords* that checks if two givens words are the same regardless of the case differences. If the strings are equal, function returns 1, otherwise 0.

For instance,

```
cout << SameWords("Trivial", "Trival") << endl;
```

will write "`0`" whereas

```
cout << SameWords("tRiViAl", "TrIvIaL") << endl;
```

will write "`1`" on the screen.

Exercise 3: Write a function named *Reverse* that reverses a given string. For instance,

```
cout << Reverse("!esreveR") << endl;
```

will write "`Reverse!`" on the screen.

How to Call Functions

We can call functions with values. For instance, the function *Power* at line 14 in the first example is called with two values 10 and 3:

```
  int pow = Power(10, 3);
```

On the other hand, we can call functions using variables. For instance, the following program writes the value of 10^z to the screen where *z* is read from the keyboard.

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

int Power(int x, int y)
{
  int prod = 1;
  for (int i = 1; i <= y; ++i)
    prod *= x;
  return prod;
}

int main ()
{
  int z;
  cin >> z;
  int pow = Power(10, z);
  cout << pow << endl;
}
```

In the program above, note that instead of writing,

```
  int pow = Power(10, z);
  cout << pow << endl;
```

we can directly write,

```
  cout << Power(10, z) << endl;
```

without defining the variable `pow`.

We can also call the functions with expressions. For instance, the following program writes the value of 10^{z*t+2} to the screen where *z* and *t* is read from the keyboard.

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

int Power(int x, int y)
{
  int prod = 1;
  for (int i = 1; i <= y; ++i)
    prod *= x;
  return prod;
}

int main ()
{
```

```
  int z, t;
  cin >> z >> t;
  cout << Power(10, z * t + 2) << endl;
}
```

In a function '`return`' command immediately returns the specified value, and the rest of the code in the function is not executed. For instance, *Power* function below immediately returns 0 if the value of *y* (the exponent) is a negative number.

```
int Power(int x, int y)
{
  if (y < 0) return 0;

  int prod = 1;
  for (int i = 1; i <= y; ++i)
    prod *= x;
  return prod;
}
```

Exercise 4: Write a program that reads three strings from the keyboard and checks if the number of vowels in these strings are equal.

input:	*output*:
`ThReE GOOD VoWeLs`	`Same number of vowels.`
`ThReE equal VoWeLs`	`Not same number of vowels.`

We can call functions inside a function. The important thing is the called function has to above the caller function. For example,

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

int Power(int x, int y)
{
  int prod = 1;
  for (int i = 1; i <= y; ++i)
    prod *= x;
  return prod;
}

int PowerOf10(int k)
{
  return Power(10, k);
}

int main ()
{
  int z;
  cin >> z;
  cout << PowerOf10(z) << endl;
}
```

In the code above *PowerOf10* function calls *Power* function hence *Power* function is above it.

Remark: As you noticed, functions are especially very useful when we have to do a process many times. They also organize a messy code, makes it more understandable.

`void` Functions

Sometimes we don't need a return value from a function. In that case, we indicate that the function returns nothing by writing 'void' before the definition of the function name.

Example: Write a program that reads two strings and writes the number of digits in these strings on the screen.

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

void NumOfDigits(string s)
{
  int digits = 0;
  for (int i = 0; i < s.length(); ++i)
    if (s[i] >= '0' && s[i] <= '9')
      ++digits;
  cout << digits << endl;
}

int main ()
{
  string s1, s2;
  cin >> s1 >> s2;
  NumOfDigits(s1);
  NumOfDigits(s2);
}
```

Exercise 5: Write a program that reads two numbers *x*, *y* and a character *c* from the keyboard. The program outputs two squares with the sizes *x* and *y* filled with character *c*.

input:	*output*:

`3 5 #`	`###`
	`###`
	`###`
	
	`#####`
	`#####`
	`#####`
	`#####`
	`#####`

Local / Global Variables

Variables defined in a function can only be used in that function. It is the same idea in the condition and loop blocks. Look at the example below:

```
1.   #include <iostream>
2.   using namespace std;
3.
4.   int Power(int x, int y)
5.   {
6.     int z = 1;
7.     for (int i = 1; i <= y; ++i)
8.       z *= x;
9.     return z;
10.  }
11.
12.  int main ()
13.  {
14.    int x, z = 11;
15.    cin >> x;
16.    cout << Power(10, x) << endl;
17.    cout << z << endl;
18.  }
```

In this example, variable *y* defined in line 4 belongs to the *Power* function hence it can't be used in *main*. The variable *x* is defined in lines 4 and 14. *x* in line 4, belongs to the *Power* whereas the other one belongs to *main*. Be careful that in line 16, the value of *x* in *main* goes to the variable *y* in the *Power* function. *x* of the *Power* function receives the value 10 from line 16. Similarly, *z* in line 14 belongs to *main* and it is not affected from *z* in the *Power* function. Hence, line 17 will print '11' for the value of *z* since it is initialized with '11' in line 14. We can consider *x*'s in *main* and *Power* as two different boxes with the same names in two different rooms. Same goes for *z*.

Exercise 6: What is the output of the following program if user enters '`I8cakes`' and '`try911`' from the keyboard respectively.

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

int NumOfDigits(string s, int x)
{
  int digits = 0;
  for (int i = 0; i < s.length(); ++i)
    if (s[i] >= '0' && s[i] <= '9')
      ++digits;
  return digits + x;
}

int main ()
{
  int x = 3;
  string s, t;
  cin >> s >> t;
  cout << NumOfDigits(t, x + x) << endl;
}
```

We can define variables outside *main* and the functions. Then *main* and all functions can use that variable. For instance,

```
1.   #include <iostream>
2.   using namespace std;
3.
4.   int prod;
5.
6.   void Power(int x, int y)
7.   {
8.     prod = 1;
9.     for (int i = 1; i <= y; ++i)
10.      prod *= x;
11.  }
12.
13.  int main ()
14.  {
15.    int x;
16.    cin >> x;
17.    Power(10, x);
18.    cout << prod << endl;
19.  }
```

prod is defined in line 4. This time, *prod* is the same box that is used by both *main* and *Power*. These type of variables are called **global** variables whereas the others are **local**.

Remark: There is another crucial difference between global and local variables; global variables are initialized by 0 (or "" – empty for *strings*) by default whereas local variables don't have that property. That means, we have to be sure that all the local variables are initialized correctly.

For example, the following program reads n numbers from the keyboard and outputs their sum.

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

int main ()
{
  int sum = 0, n, number;

  cin >> n;
  for (int i = 0; i < n; ++i)
  {
    cin >> number;
    sum += number;
  }
  cout << sum << endl;
}
```

Note that *sum* needs to have an initial value of '0', but it is not a global variable. Hence we initialize it with '0'.

Note: Avoid defining a global variable with the same name of a local variable. That makes more confusion in the code.

Exercise 7: What is the output of the following program if user enters '`I8cakes`' and '`try911`' from the keyboard respectively.

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

int y;

int NumOfDigits(string s, int x)
{
  int digits = 0;
  x += y;
  for (int i = 0; i < s.length(); ++i)
    if (s[i] >= '0' && s[i] <= '9')
      ++digits;
  return digits + x;
}

int main ()
{
  int x = 3;
  string s, t;
  cin >> s >> t;
  cout << NumOfDigits(t, 19) << endl;
  y = x * 2;
}
```

Remark: It is better to define local variables as much as possible. We mostly define global variables when we need to use the same variable in many functions or we want to use the advantage of initialization for arrays and matrices.