2 Statistical Thermodynamics and the Boltzmann Distribution

This set of exercises brings together the theory of statistical mechanics with a practical example, the harmonic oscillator. After a brief (re)introduction to statistical mechanics, you will write a code that computes the probability distribution of the energy of a system using Boltzmann statistics.

2.1 Statistical Thermodynamics

The average behaviour of a macroscopic system, the state of which is uncertain, can be studied using probability theory (*statistical mechanics*). However, such a probabilistic description cannot make thermodynamic properties accessible, unless a link between thermodynamic (free energy, pressure, entropy) and mechanical (position, momenta) properties can be introduced on the microscopic scale. Statistical thermodynamics (or *equilibrium statistical mechanics*) is concerned with the description of *classical thermodynamics* from a microscopic picture in terms of the particles that constitute the thermodynamical system. Statistical thermodynamics thus provide a link between the system's microscopic properties and its macroscopic behaviour. The following constitutes a brief summary of the most important concepts in statistical mechanics (phase space, entropy as a phase space volume, partition functions). The discussion starts from the classical description of the microcanonical ensemble and is then extended onto the canonical and grandcanonical ensemble, with a link between quantum mechanics and the canonical ensemble being provided at the end of this chapter.

2.1.1 Phase Space and the Microcanonical Ensemble

Any system that consists of N particles can be characterised in terms of a 6N-dimensional set of variables: 3N coordinates $\mathbf{q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_N\}$ and 3N momenta $\mathbf{p} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$. At every instant, the system is described by the values of $\mathbf{p}$ and $\mathbf{q}$ within a 6N-dimensional space, the *phase space* $\Gamma = (\mathbf{p}, \mathbf{q})$. The point in phase space that characterises its current state identifies a point referred to as the *representative point*. The system (*i.e.* its representative point) will evolve within phase space according to its Hamiltonian:

$$H = \sum_{i=1}^{N} \left[\frac{\mathbf{p}_i^2}{2m} + U\left(\mathbf{q}_1, \ldots, \mathbf{q}_N\right) \right]. \tag{1}$$

Note that in the classical limit, the Hamiltonian is not an operator, but a sum over kinetic and potential energy terms. The evolution of the system in phase space is governed by Hamilton's equations of motion:

$$\frac{\mathrm{d}\mathbf{p}_i}{\mathrm{d}t} = -\frac{\partial H}{\partial \mathbf{q}_i} \tag{2}$$

$$\frac{\mathrm{d}\mathbf{q}_i}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}_i}. \tag{3}$$

As is evident from eq. 3, this formulation is an equivalent of Newtonian mechanics. Given a representative point, the evolution of the system in phase space is thus fully determined by Hamilton's equations of motion. For an isolated system of constant particle number N, constant volume V and constant energy E, the trajectory of the system will lie on a hypersurface defined by:

$$H = E = \text{const.} \tag{4}$$

Such a constant-NVE system of particles is described by a statistical ensemble referred to as the *microcanonical ensemble*. In the microcanonical ensemble, a system will move on the 6N dimensional hypersurface spanned by coordinates and momenta that lead to the same total energy E. Each of the possible representative points $\mathbf{p}, \mathbf{q}$ that give rise to the same energy E correspond to one *microstate* of the system. In general, for a thermodynamic state that is characterised by a set of extensive variables $\{X_0, \ldots, X_r\}$, the $\{X_i\}$ will be a function over microscopic states $\{\mathbf{p}, \mathbf{q}\}$, such that $X(\mathbf{r}, \mathbf{q}) = X(\Gamma)$. The region of phase space that is accessible to the system $X(\mathbf{p}, \mathbf{q})$ is thus determined and limited by the values of X. Such an 'accessible region' in phase space is (within the error of the varibles X) denoted by $W(E)$, with the corresponding volume element being $|W(E)|$. According to the fundamental postulate of statistical mechanics, since every microstate has the same energy, the system is equally likely to be found in any of its microstates. In terms of the volume element in phase space, the probability to find the system in one of its microstates is:

$$P(E) = \frac{1}{|W(E)|}. \tag{5}$$

Based on $W(E)$, *any* thermodynamical property and observable is now accessible as a *phase-space* or *ensemble average*. For a microcanonical ensemble, the expectation value of an observable O is simply:

$$\langle O \rangle = \int_{\Gamma \in W(E)} \frac{1}{|W(E)|} O(\Gamma) \mathrm{d}\Gamma, \tag{6}$$

where the integration is over the 6N variables within the phase space volume elements $d\Gamma$ that are contained in $W(E)$. The above equation is nothing but the *probability density* of the system to be in a macroscopic state with the property O (recall the general form of a probability density). Macroscopic observables are therefore linked to microscopic quantities using simple tools of *statistics*.

The Hamiltonian quantifies the kinetic and potential energy of the system, but the concept of entropy has been absent from the discussion so far. According to Boltzmann's fundamental postulate of statistical thermodynamics, the entropy of a system is linked to the accessible volume $W(E)$ in phase space, *i.e.* the entropy of a system in state $X(\Gamma)$ is related to the volume accessible to X:

$$S = k_B \ln |W(E)|, \tag{7}$$

where S is the entropy as introduced for macroscopic systems, and k_B is Boltzmann's constant. Thus, a link between thermodynamic quantities and the system's microscopic states is established, allowing for the prediction of *any* thermodynamic observable. However, the 6N dimensionality of the problem restricts the analytic expressions of the above equations to small model systems; thermodynamic properties *e.g.* of a protein are in practice impossible to predict (although the exact Hamiltonian of the system is of course known). The great value of the techniques thaught in this course lies in them providing alternative approaches to calculate thermodynamical properties as ensemble averages, but without explicitly relying on eq. 6.

2.1.2 The Canonical Ensemble and the Canonical Partition Function

The statistical concepts considered so far can be extended onto systems of constant particle number N, constant volume V and constant temperature T, rather than constant energy. The derivation of the properties of an NVT-ensemble starts from the microcanonical ensemble. The NVT system is considered to be in contact with a much larger *thermal bath*, the purpose of which it is to exchange energy with the NVT-subsystem such that the subsystem's temperature remains constant, without the energy of the overall system (*i.e.* including the bath) changing due to its enormous size. An NVT system of particles is represented by the *canonical ensemble*. According to Boltzmann and Gibbs, the expectation value of an observable is again an ensemble average,

$$\langle O \rangle = \int_\Gamma \frac{1}{Z} O(\Gamma) e^{-\beta E(\Gamma)} \mathrm{d}\Gamma \tag{8}$$

but now weighted by some exponential factor - the *Boltzmann factor* - that takes into account the energy of every possible microstate. *En passant*, we have introduced the *canonical partition function* $Z(N, V, T)$ as the counterpart of the microcanonical W, as well as the inverse of the temperature, the *thermodynamic beta*, $\beta = \frac{1}{k_B T}$. Again, the expression is a simple probability density function. It can be shown that for the above to be a valid normalised probability distribution, $Z(N, V, T)$ must be of the form:

$$Z(N, V, T) = \frac{1}{N!\, h^{mN}} \int_\Gamma e^{-\beta E(\Gamma)} \mathrm{d}\Gamma, \tag{9}$$

i.e. the partition function is nothing but the integral over all possible microstates that the system can assume. Since classical states are indistinguishable and uncountable (they form a continuum), the partition function has to be normalised to avoid overcounting ($N!$) and to account for Z being dimensionless (h^{mN}, where m is the dimensionalty of the integral over position and momenta). The microstates do not share an equal probability as in the microcanonical ensemble, and the integral is over *all* phase space rather than over a restricted region. According to the above definitions, the probability of observing a microstate s is given by its energetic weight:

$$P(E_s) = \frac{1}{Z} g_s e^{-\beta E_s}, \tag{10}$$

where g_s takes into account possible degeneracies. In general, the classical case does not hold and the microstates do not form a continuum, but are discrete and as such countable. Therefore, the integrals in eqs 8 and 9 may be recast as sums:

$$\langle O \rangle = \sum_s \frac{1}{Z(T)} g_s O_s e^{-\beta E_s}, \tag{11}$$

where the canonical partition function is now a sum over states:

$$Z(N, V, T) = \sum_i g_i e^{-\beta E_i}. \tag{12}$$

2.1.3 The Quantum Mechanical Partition Function and the Density Operator

This section briefly relates the concepts treated so far with the density matrix formalism introduced in the course 'Introduction to Electronic Structure Methods'. The density operator (called the density matrix in position basis) is at the base of the quantum mechanical formulation of statistical mechanics, since it casts the problem into a basis-independent form. Recall that a density operator in *state basis*

$$\hat{\rho} = \sum_i p_i \hat{\gamma}_i \tag{13}$$

$$= \sum_i p_i \left|\Psi_i\right\rangle \left\langle\Psi_i\right| \tag{14}$$

can be used to describe both mixed ($p_0 < 1$) and pure states ($p_0 = 1$, where the density operator becomes a projector), and that its diagonal elements play the role of a probability if the states are orthogonal. Note that $Tr\left(\hat{\rho}\right) = 1$ applies for non-orthogonal states as well. The γ_i are pure-state density matrices that admit some eigenbasis $\{|i\rangle\}$. The great benefit of the density operator formalism lies in its intrinsic ability to describe statistical mixtures of states if they are entangled. In this case, the Hilbert spaces of the respective Hamiltonians form a composite system of Hilbert spaces $\mathbb{H}_1 \otimes \mathbb{H}_2$, and either $\mathbb{H}_1$ or $\mathbb{H}_2$ will be inaccessible to any observation. Density matrices allow for the determination of the properties of either subsystem by means of the *partial trace* over the other respective Hilbert space, incorporating statistical uncertainties.

The *canonical* density matrix is defined as:

$$\hat{\rho} = \frac{1}{Z} e^{-\beta \hat{\mathrm{H}}}, \tag{15}$$

and the *canonical partition function* is:

$$Z(N, V, T) = \mathrm{Tr}\left(e^{-\beta \hat{\mathrm{H}}}\right). \tag{16}$$

In the eigenbasis $\{|\Psi_s\rangle\}$ of $\hat{\mathrm{H}}$ (a specific rather than a general case), one may write

$$\hat{\rho} = \frac{1}{Z} \sum_s e^{-\beta \hat{\mathrm{H}}} \left|\Psi_s\right\rangle \left\langle\Psi_s\right| \tag{17}$$

$$= \frac{1}{Z} \sum_s \left|\Psi_s\right\rangle e^{-\beta E_s} \left\langle\Psi_s\right|. \tag{18}$$

Note that the trace is only defined in some orthonormal basis $\{Ket\Psi\}$, and that the basis may be chosen to diagonalise $\hat{\rho}$ (but not forcibly $\hat{\mathrm{H}}$). The above definitons are still unambiguous, since the trace is invariant under a change of basis. In the eigenbasis of the Hamiltonian (*i.e.* in the absence of coherence), the partition function becomes:

$$Z(N,V,T) = \sum_i e^{-\beta E_i}, \tag{19}$$

which recovers the form of eq. 12, and $p_s = \frac{1}{Z} e^{-\beta E_s}$. Hence, given a system's density matrix, all of its thermodynamic properties can be derived:

$$\langle \hat{\mathrm{O}} \rangle = \mathrm{Tr}\left(\hat{\mathrm{O}}\hat{\rho}\right), \tag{20}$$

and, if $\left[\hat{\mathrm{H}}, \hat{\mathrm{O}}\right] = 0$, then:

$$\langle \hat{\mathrm{O}} \rangle = \frac{1}{Z} \sum_s \mathrm{O}_s e^{-\beta E_s}, \tag{21}$$

which recovers the expression for the classical probability distribution function presented in the preceeding section.

2.1.4 The Isothermal-Isobaric Ensemble

The isothermal-isobaric or NpT-ensemble is of great practical relevance in chemistry. The isothermal-isobaric partition function is a weighted sum over canonical partition functions, where the definition of the weighting factor is ambiguous:

$$\Delta(N,p,T) = \int Z(N,V,T) e^{-\beta p V} \frac{N}{V} \mathrm{d}V, \tag{22}$$

or

$$\Delta(N,p,T) = \int Z(N,V,T) e^{-\beta p V} \beta p \ \mathrm{d}V. \tag{23}$$

Both definitions become equal if $N \to \infty$, $V \to \infty$ and $\frac{N}{V} = \text{const}$, the *thermodynamic limit*.

2.1.5 The Grand Canonical Ensemble

If the particle number N is not constant anymore, *i.e.* if the system is not only in contact with a temperature reservoir, but also able to exchange particles, its possible states are described by the grand canonical ensemble. Instead of the particle number, the chemical potential μ is introduced as the constant (μVT-ensemble). If you wish to obtain more ample information on thermodynamic ensembles, why not consult one of the excellent introductory books in the field (*e.g.* 'Introduction to Modern Statistical Mechanics' by David Chandler, published by Oxford University Press).

2.1.6 Thermodynamic Quantities as Derivatives of the Canonical Partition Function

Thermodynamic properties can be directly derived from the partition function. In the canonical ensemble, the energy of the system is given by

$$E = -\frac{\partial}{\partial \beta} \ln Z, \tag{24}$$

whereas the Helmholtz free energy ($A \equiv U - TS$) is obtained from

$$A = -kT \ln Z, \tag{25}$$

and, finally, the entropy can be derived according to:

$$S = k_B \frac{\partial}{\partial T} \left(T \ln Z \right). \tag{26}$$

The probability of finding a particle in a microstate s is given by the Boltzmann distribution:

$$P(E_s) = \frac{N_s}{N} = \frac{g_s e^{-\beta E_s}}{\sum_i g_i e^{-\beta E_i}} = \frac{1}{Z} g_s e^{-\beta E_s}, \tag{27}$$

where N_s are the number of particles in the microstate s, N is the total number of particles and E_s is the energy of microstate s.

2.2 Computing the Boltzmann Distribution of a Fictitious Harmonic Oscillator

In this exercise you will apply the Boltzmann distribution (eq. 27) to determine the occupancy of states for a fictitious harmonic oscillator. Figure 2.2 presents our system. Here individual states are separated by the energy ϵ with zero-point energy ϵ_0. From this point onwards it is practical to use reduced temperature such that $k_B = 1$, and define that the energy levels $\epsilon_0, \epsilon_1, \epsilon_2, \ldots, \epsilon_N$ of the harmonic oscillator take the values $1, 2, 3, \ldots, N$. We are particularly interested in the evolution of the distribution of microstates upon changing the temperature or degeneracy, thus in this exercise we will build a C++ tool to discover the Boltzmann distribution characteristics of the system.

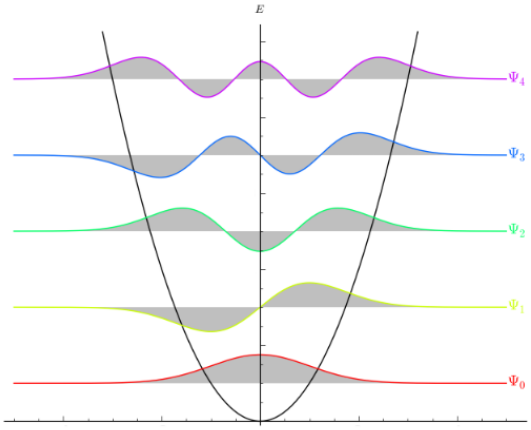

Figure 1: Harmonic Oscillator with states $\Psi = \{\psi_0, \psi_1, \psi_2, \ldots, \psi_N\}$ and associated energies $E = \{\epsilon_0, \epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$.

2.2.1 Designing the Boltzmann Distribution C++ Program

In a directory of your choosing, create a file called '`boltzmann.cpp`' and open it in a text-editor. Within this file, begin by adding the following:

```
1  #include <iostream>
2  #include <fstream>
3  #include <math.h>
4  #include <cstdlib>
5  #include <string>
6
7  using namespace std;
8
9  double calculateStateOccupancy(double temperature, double stateEnergy);
10 void outputToFile(double *distribution, string outputFile);
11
12 int numberOfEnergyLevels = 0;
```

```
13 double reducedTemperature = 0.0;
14 double partitionFunction = 0.0;
15
16 double *distribution;
17
18 int main(int argc, char * argv []) {
19
20   return 0;
21 }
```

This program will calculate the Boltzmann distribution of the fictitious system at a given temperature and degeneracy. The variables you require to perform this calculation are `numberOfEnergyLevels`, `reducedTemperature`, `normalise` and `distribution`.

2.2.2 Parsing Command-line Arguments

The number of energy levels you wish to calculate the distribution over (`numberOfEnergyLevels`) as well as the reduced temperature of your system (`reducedTemperature`) will need to be passed into your main function. This can be achieved using the following code within the main function:

```
1   numberOfEnergyLevels = atoi(argv[1]);
2   reducedTemperature = atof(argv[2]);
```

2.2.3 Storing and Accessing the Boltzmann Distribution

A structure to hold the distribution also needs to be created. Write the following within your main function, once the value of `numberOfEnergyLevels` has been set:

```
1   distribution = new double [numberOfEnergyLevels];
```

This code creates an array of rational numbers of size `numberOfEnergyLevels`. The function `new` allocates a memory block whose purpose is to hold the contents of this array. This function also assigns the address of the begining of this memory block to the pointer `distribution`. For example if the array contains the values $[0.9, 0.05, 0.03, 0.02]$, then these individual values can be accessed via `distribution[j]` or in pointer-arithmetic by `*(distribution+j)`. Pointers and arrays are discussed in more detail within the C++ glossary in section 2.4.

2.2.4 Calculating the Boltzmann Distribution

You are now ready to design the Boltzmann distribution implementation within your main function. Enter the following code in an appropriate place, i.e. once the array `distribution` has been allocated:

```
  for(int i = 0; i < numberOfEnergyLevels; i++)
  {
    distribution[i]=calculateStateOccupancy(reducedTemperature,  i);
    partitionFunction+=stateOccupancy;
  }
```

Recall that the calculation of the Boltzmann distribution requires the calculation of the partition function, Z. Hence, within this loop you calculate the partition function (`partitionFunction`) iteratively by summing each `stateOccupancy` value.

2.2.5 Outputting the Results

To finish the flow of the program, you will need to output the results of your calculation to a file and terminate the program. This can be achieved by calling the `outputToFile()` function. Place the following code after the for-loop:

```
outputToFile(distribution, "results.dat");
delete [] distribution;
```

It is worthwhile to note that since you dynamically allocated a new (un-managed) memory block within your program earlier, you must also delete it once you are finished. To free the memory associated with this array you must call `delete [] distribution`.

You will also need to provide an implementation for the function `outputToFile()`. Enter the following code below the main function:

```
void outputToFile(double *distribution, string outputFile) {
  ofstream output;
  output.open(outputFile);

  for (int i = 0; i < numberOfEnergyLevels; i++) {
    output << i << " " << distribution[i]/partitionFunction << "\n";
  }
  output.close();
}
```

The purpose of this function is to write the Boltzmann distribution contained within the array `distribution` to the file specified by the path `outputFile`. If this file does not exist, it will automatically create it and otherwise will entirely overwrite all contents contained therein. Writing to files is discussed further in the glossary (section 2.4). Also note that the normalisation of the Boltzmann distribution with the partition function occurs within the writing step.

2.2.6 Calculating State Occupancy

Finally, place the following code somewhere below your main function. You must now modify this function to complete this Boltzmann distribution program. Hint: recall that

you are using reduced temperature.

```
1 double calculateStateOccupancy(double temperature, int i) {
2   double stateOccupancy = 0.0;
3   //Calculate the occupancy for state i
4   return stateOccupancy;
5 }
```

Note: The directive `#include <math.h>` at the begining of your code includes the math library. This library contains the function `exp()` which you can use to calculate e^x by calling `exp(x)`.

2.2.7 Compiling Your C++ Code

To compile the program, navigate your terminal focus to the directory which contains `boltzmann.cpp`, and type the following:

```
g++ -std=c++11 boltzmann.cpp -o boltzmann.x
```

This will produce an executable file called `boltzmann.x` in your current directory. You can execute this program using the following command-line arguments:

```
./boltzmann.x numberOfEnergyLevels reducedTemperature
```

2.3 Exercises

2.3.1 Statistical Mechanics

a) A quantum harmonic oscillator has energy levels:

$$E_n = \left(n + \frac{1}{2}\right)\hbar\omega. \tag{28}$$

Write down the corresponding canonical partition function $Z(N, V, T)$. Note that in this case, the partition function forms an infinite geometric series, and can be rewritten in terms of the $n \to \infty$ limit of the series. From the result you obtain, derive the expectation value of the energy. Use the limit of the geometric series for Z, rather than the sum-based form.

b) Derive the Boltzmann distribution, eq. 27, from eq. 11, using the expectation value of the particle number in state s, N_s.

c) **Bonus**: Show that, based on eq. 20, the canonical partition function (eq. 21) is obtained from eq. 15 if the Hamiltonian admits an orthonormal eigenbasis $\{|\Psi_i\rangle\}$ and the commutator $\left[\hat{\mathrm{H}}, \hat{\mathrm{O}}\right]$ vanishes. (There is no need to use the commutator itself, applying conditions that follow from vanishing commutators is sufficient. If you wish to provide an extended derivation in a general basis, recall that the time-independent case applies.)

d) **Bonus**: Show from eqs 14 and 15 that the p_i are indeed equal to $\frac{1}{Z}e^{-\beta E_i}$ for pure states, given that there exists a common eigenbasis $\{|\Psi_i\rangle\}$ to the total Hamiltonian and explain the origin of this restriction. Explain why the density operator and the expression for the expectation value of an observable assume a general form (eqs 15 and 20), rather than being defined directly in terms of eqs 18 and 21. (You may link your answer to the assumptions made in c)).

2.3.2 Boltzmann Distribution

To refresh your memory on how to use the gnuplot program to generate graphs, refer to section 2.3.3.

a) Modify the code presented in section 2.2.6 to calculate the occupancy of each state within the harmonic oscillator system. Present the entire '`boltzmann.cpp`' file within your report and comment upon the main features.

b) Calculate using your program the occupancy of each state within the harmonic oscillator at the reduced temperatures of 0.5, 1, 2 and 3, with `numberOfEnergyLevels` set to 10 (recall $\epsilon = 1$). Using the output file the code generates, plot the distribution and present the graphs in your report. What do you note at higher temperature?

c) Change the `calculateStateOccupancy()` function such that the degeneracies $(s+1)$ and $(s+2)$ are considered, where s is the index of the energy level (use `numberOfEnergyLevels` = 10 and `reducedTemperature` = 0.5). Plot the results and include the graphs in your report. What can you infer from this trend?

d) Modify your program such that the state occupancy and partition function are calculated for a linear rotor with moment of inertia I. Compare your results at different temperatures with the approximate result:

$$Z = \frac{2I}{\beta\hbar^2}, \tag{29}$$

using

$$\frac{I}{\hbar^2} = 1. \tag{30}$$

Note that the energy levels of a linear rotor are:

$$U = J(J+1)\frac{\hbar^2}{2I} \tag{31}$$

with $J = 0, 1, 2, \ldots, \infty$ and the degeneracy of level J is $2J + 1$.

2.3.3 GNUPlot: Displaying Results

A convenient way to quickly graph the data contained within your results file is to use the `gnuplot` program. Open a terminal and navigate the terminal focus to the directory containing your results file. Enter the command `gnuplot` to start the program.

Plotting Your results file only contains 2 columns thus you can simply plot the graph directly using the following command:

```
plot 'results.dat'
```

If however you have multiple columns from which to plot, you can use the `using` specifier as follows:

```
plot 'results.dat' using 2:3
```

which will plot the third column (y-axis) against the second (x-axis). Extending this, to plot multiple graphs on one image, you can type the following:

```
plot 'results1.dat' using 1:2 title 'graph1' with lines, \

'results2.dat' using 1:2 title 'graph2' with lines, \

'results3.dat' using 1:2 title 'graph3' with lines
```

In this example each data point is connected with a straight line using the `with lines` command. Note that `with lines` may be abbreviated as `w l`, `using` as `u`, and `title` as `t`.

Labelling Axes Axis labels are added with the `set` command. The following commands label the x-axis and y-axis respectively:

```
set xlabel 'Energy Level'

set ylabel 'Occupancy'
```

Output to File To write the graph to file with PNG format, the following commands can be used:

```
set term png

set output 'results.png'

replot

set term x11
```

2.4 C++ Glossary

2.4.1 Pointers and Arrays

Pointers Memory in a computer can be viewed simply as a succession of memory cells, each one byte in size, and each with a unique address. When a variable is declared, the memory block required to store its value is assigned at a specific location (its memory address). Pointers are simple structures which allow you to obtain the memory address of a particular variable, and are declared as follows:

```
type * name;
```

where `type` refers to the variable type this pointer is pointing to. The memory address of a variable can then be accessed with the address-of operator (&) as follows:

```
1 int bar = 1000;
2 int * foo = &bar;
```

which assigns to the pointer `foo` the address to the memory block containing the contents of the variable `bar`. The value stored within the memory block addressed by the pointer is accessed by using the dereference operator (*):

```
1 int bar = 1000;
2 int * foo = &bar;
3 cout << foo << endl; //prints memory address of variable bar
4 cout << *foo << endl; //dereference foo to print 1000
```

Finally, when you create a dynamically allocated variable using the `new` operator, you must declare this statement with a pointer as follows:

```
1 Object * objectPointer = new Object;
2 Object * objectArray = new Object[3];
```

such that when its time to free the memory associated with this variable, you can use the `delete` operator (or `delete []` for arrays) to act on its pointer:

```
1 delete objectPointer;
2 delete[] objectArray;
```

Arrays The concept of an array is very much akin to that of a pointer. In C++ arrays are simply wrapped pointers which address the starting memory block of the contents contained within the array. This is illustrated in the following example:

```
1 int myarray [3];
2 int * arrayPointer = &myarray;
3 myarray[0] = 5;
4 myarray[1] = 6;
```

```
5  myarray[2] = 7; // array now contains {5, 6, 7}
6  cout << *(arrayPointer) << ", " << myarray[0] << endl; //prints 5, 5
7  cout << *(arrayPointer+1) << ", " << myarray[1] << endl; //prints 6, 6
```

2.4.2 Writing to Files

The classes `ofstream` and `ifstream` can be used for writing and reading files respectively. To make use of these classes you simply include the `<iostream>` and `<fstream>` libraries. To write to files, the following commands can be used:

```
1  ofstream myfile;
2  myfile.open("example.txt");
3  myfile << "Writing this to file.\n";
4  myfile.close();
```

Here an object of class `ofstream` is created with the name `myfile`. Next, the function `open()` is called to create a stream to the file `example.txt`. Characters are pushed onto this stream using the stream insertion operator `«` and finally the function `close()` is called to flush the stream and release any allocated memory.