Example of Quicksort as a Template by using Overload Operation.

Posted by Alejandro G. Carlstein Ramos Mejia on October 15, 2010 November 12, 2010 About Programming / Algorithms / C++

Example of Quicksort as a Template by using Overload Operation.

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given 'as is'. I do not take responsibilities of how they are used.

doublesort.cpp:

```
/**
  * @Author: Alejandro G. Carlstein
  */
#include <iostream>
#include <fstream>
#include <string>

#include 'Sort.cc'
#include 'Item.h'

using namespace std;

const int MAX_WORDS = 1000;

const string OUTPUT_ALPHA_SORT_FILE = 'short.alpha.txt';
const string OUTPUT_FREQ_SORT_FILE = 'sort.freq.txt';

template <typename T>
T *resize_array(T *array,
     int &num_elements) {
 int new_size = num_elements * 2;

    T* new_array = new T[new_size];

    //memcpy( new_array, array, num_elements * sizeof(T) );
 for (int i = 0; i < num_elements; i++){

  //new_array[i].data = array[i].data;
  //new_array[i].frequency = array[i].frequency;
  new_array[i] = array[i];

 }

    num_elements = new_size;
```

```
    delete [] array;

    //array = new_array;

 //return new_size;
 return new_array;
}

/**
 * main
 * @description: Lab 10
 * @due: November 24, 2009 1:00PM EST
 */
int main(int argc, char *argv[]){

 int str_len;

 string words[MAX_WORDS];

 Sort<string> str_sort;

 ofstream fout;

 str_len = -1;

 //1. Reads up to 1000 words separated by newline characters   (one at the time)
 for (int i = 0; !cin.fail() && i < MAX_WORDS; i++, str_len++)
  cin >> words[i];

 //2. Print out in alphabetical order to file called short.alpha.txt
 // 2a. one word per line with number of occurences of each word
 //        listed in parentheses after the word

 fout.open(OUTPUT_ALPHA_SORT_FILE.data());

 if (fout.is_open()){

  // Bonus A. Use an O(N log N) sorting algorithm, make sure that no
  //                 pre-processing steps exceed O(N log N)
  str_sort.quicksort(words, 0, str_len - 1);

  for (int i = 0; i < str_len; i++)
   cout << words[i] << endl;

  for (int i = 0; i < str_len; i++)
   fout << words[i] << endl;

 }else{
  cerr << '[X] ERROR: Couldn't open ' << OUTPUT_ALPHA_SORT_FILE << endl;
 }

 fout.close();
```

```
 //3. Print out in frequency order to file caled sort.freq.txt

 fout.open(OUTPUT_FREQ_SORT_FILE.data());

 if (fout.is_open()){

  int item_size = 2;

  Item* item = new Item[item_size];

  Sort<Item> item_sort;

  // Record their frequency
  int j = 0;

  item[0].frequency = 1;
  item[0].data = words[0];

  for (int i = 1; i < str_len; i++){

   if (item_size < i){

    // resize array
    item = resize_array(item, item_size);

   }

   if (item[j].data == words[i]){

    //Increase frequency
    item[j].frequency++;

   }else{

    j++;

    item[j].frequency = 1;

    item[j].data = words[i];

   }

  }

  //Order words by frequency
  // Bonus A. Use an O(N log N) sorting algorithm, make sure that no
  //                 pre-processing steps exceed O(N log N)
  item_sort.quicksort(item, 0, str_len);

  for (int i = 0; i < str_len; i++)
   if (item[i].frequency > 0) fout << '(' << item[i].frequency << ') ' << item[i].data
<< endl;
```

```
 }else{
  cerr << '[X] ERROR: Couldn't open ' << OUTPUT_ALPHA_SORT_FILE << endl;
 }

 fout.close();

 return 0;

}
```

Item.h:

```
/**

  * @Author: Alejandro G. Carlstein

  */

#ifndef ITEM_H

#define ITEM_H

#include <string>

using namespace std;

class Item{

 public:

  int frequency;

  string data;

  //Bonus B. Write a single sort function (but two different comparison

  //     functions/operators) to produce the two different

  //     sorted lists.

  // Compare two Items by their number

  friend bool operator == (const Item& Item1,

         const Item& Item2);

  // Compare two Items by their number

  friend bool operator != (const Item& Item1,

         const Item& Item2);

  // Compare two Items by their number
```

```
  friend bool operator <= (const Item& Item1,
         const Item& Item2);
  // Compare two Items by their number
  friend bool operator >= (const Item& Item1,
         const Item& Item2);
  // Compare two Items by their number
  friend bool operator < (const Item& Item1,
        const Item& Item2);
  // Compare two Items by their number
  friend bool operator > (const Item& Item1,
        const Item& Item2);
  Item(void);
  Item(const Item& new_item);
  ~Item(void);
};
#endif
```

Item.cpp:

```
/**
  * @Author: Alejandro G. Carlstein
  */

#include 'Item.h'

Example of Quicksort as a Template by using Overload Operation.

NOTIFICATION:  These  examples are provided for educational  purposes.   Using this
code is  under your own responsibility and risk. The  code is   given  ‘as is’. I do
not take responsibilities of how they are  used.

doublesort.cpp:

/**
  * @Author: Alejandro G. Carlstein
  */
#include <iostream>
#include <fstream>
#include <string>

#include 'Sort.cc'
```

```
#include 'Item.h'

using namespace std;

const int MAX_WORDS = 1000;

const string OUTPUT_ALPHA_SORT_FILE = 'short.alpha.txt';
const string OUTPUT_FREQ_SORT_FILE = 'sort.freq.txt';

template <typename T>
T *resize_array(T *array,
     int &num_elements) {
 int new_size = num_elements * 2;

    T* new_array = new T[new_size];

    //memcpy( new_array, array, num_elements * sizeof(T) );
 for (int i = 0; i < num_elements; i++){

  //new_array[i].data = array[i].data;
  //new_array[i].frequency = array[i].frequency;
  new_array[i] = array[i];

 }

    num_elements = new_size;

    delete [] array;

    //array = new_array;

 //return new_size;
 return new_array;
}

/**
 * main
 * @description: Lab 10
 * @due: November 24, 2009 1:00PM EST
 */
int main(int argc, char *argv[]){

 int str_len;

 string words[MAX_WORDS];

 Sort<string> str_sort;

 ofstream fout;

 str_len = -1;

 //1. Reads up to 1000 words separated by newline characters   (one at the time)
 for (int i = 0; !cin.fail() && i < MAX_WORDS; i++, str_len++)
  cin >> words[i];

 //2. Print out in alphabetical order to file called short.alpha.txt
 // 2a. one word per line with number of occurences of each word
 //        listed in parentheses after the word

 fout.open(OUTPUT_ALPHA_SORT_FILE.data());

 if (fout.is_open()){

  // Bonus A. Use an O(N log N) sorting algorithm, make sure that no
  //                 pre-processing steps exceed O(N log N)
  str_sort.quicksort(words, 0, str_len - 1);

  for (int i = 0; i < str_len; i++)
   cout << words[i] << endl;

  for (int i = 0; i < str_len; i++)
   fout << words[i] << endl;
```

```
 }else{
  cerr << '[X] ERROR: Couldn't open ' << OUTPUT_ALPHA_SORT_FILE << endl;
 }

 fout.close();

 //3. Print out in frequency order to file caled sort.freq.txt

 fout.open(OUTPUT_FREQ_SORT_FILE.data());

 if (fout.is_open()){

  int item_size = 2;

  Item* item = new Item[item_size];

  Sort<Item> item_sort;

  // Record their frequency
  int j = 0;

  item[0].frequency = 1;
  item[0].data = words[0];

  for (int i = 1; i < str_len; i++){

   if (item_size < i){

    // resize array
    item = resize_array(item, item_size);

   }

   if (item[j].data == words[i]){

    //Increase frequency
    item[j].frequency++;

   }else{

    j++;

    item[j].frequency = 1;

    item[j].data = words[i];

   }

  }

  //Order words by frequency
  // Bonus A. Use an O(N log N) sorting algorithm, make sure that no
  //                 pre-processing steps exceed O(N log N)
  item_sort.quicksort(item, 0, str_len);

  for (int i = 0; i < str_len; i++)
   if (item[i].frequency > 0) fout << '(' << item[i].frequency << ') ' << item[i].data << endl;

 }else{
  cerr << '[X] ERROR: Couldn't open ' << OUTPUT_ALPHA_SORT_FILE << endl;
 }

 fout.close();

 return 0;

}
```

Item.h:

```
/**
  * @Author: Alejandro G. Carlstein
  */
#ifndef ITEM_H
#define ITEM_H
#include <string>
using namespace std;
class Item{
 public:
  int frequency;
  string data;
  //Bonus B. Write a single sort function (but two different comparison
  //     functions/operators) to produce the two different
  //     sorted lists.
  // Compare two Items by their number
  friend bool operator == (const Item& Item1,
         const Item& Item2);
  // Compare two Items by their number
  friend bool operator != (const Item& Item1,
         const Item& Item2);
  // Compare two Items by their number
  friend bool operator <= (const Item& Item1,
         const Item& Item2);
  // Compare two Items by their number
  friend bool operator >= (const Item& Item1,
         const Item& Item2);
  // Compare two Items by their number
  friend bool operator < (const Item& Item1,
        const Item& Item2);
  // Compare two Items by their number
  friend bool operator > (const Item& Item1,
        const Item& Item2);
  Item(void);
  Item(const Item& new_item);
  ~Item(void);
};
#endif
```

Item.cpp:

```
/**
  * @Author: Alejandro G. Carlstein
  */

#include 'Item.h'

/**
 * operator ==
 * @description: Compare two Items by their number
 * @param: in, Item1, Item2
 * @return: istream&
 */
bool operator == (const Item& Item1,
      const Item& Item2){
  return  (Item1.frequency == Item2.frequency);
}

/**
 * operator !=
 * @description: Compare two Items by their number
 * @param: in, Item1, Item2
 * @return: istream&
 */
bool operator != (const Item& Item1,
      const Item& Item2){
     return  (Item1.frequency != Item2.frequency);
}

/**
 * operator <=
 * @description: Compare two Items by their number
 * @param: in, Item
 * @return: istream&
 */
bool operator <= (const Item& Item1,
      const Item& Item2){

    bool b_rtn = false;

 if (Item1.frequency == Item2.frequency){

  b_rtn = (Item1.data > Item2.data);

 }else{
  b_rtn = (Item1.frequency > Item2.frequency);
 }

 return  b_rtn;

}

/**
 * operator >=
 * @description: Compare two Items by their number
 * @param: in, Item1, Item2
 * @return: istream&
 */
bool operator >= (const Item& Item1,
      const Item& Item2){

 bool b_rtn = false;

 if (Item1.frequency == Item2.frequency){

  b_rtn = (Item1.data > Item2.data);

 }else{
```

```
  b_rtn = (Item1.frequency < Item2.frequency);
 }

 return  b_rtn;
}

/**
 * operator <
 * @description: Compare two Items by their number
 * @param: in, Item1, Item2
 * @return: istream&
 */
bool operator < (const Item& Item1,
     const Item& Item2){
 return (Item1.frequency > Item2.frequency);
}

/**
 * operator >
 * @description: Compare two Items by their number.
 * @param: in, Item1, Item2
 * @return: istream&
 */
bool operator > (const Item& Item1,
     const Item& Item2){
 return (Item1.frequency < Item2.frequency);
}

Item::Item(void):
   frequency(0),
   data(''){
}

Item::Item(const Item& new_item):
   frequency(new_item.frequency),
   data(new_item.data){
}

Item::~Item(void){
}
```

Sort.cc:

```
/**
  * @Author: Alejandro G. Carlstein
  * @Course: CS240
  */

#ifndef SORT_CC
#define SORT_CC

#include <iostream>
#include <cstdlib>

//#include 'functions.h'

using namespace std;

//////////////////////////////////////////////////////////////////
// TEMPLATE CLASS SORT   ////////////////////////////////////////
//////////////////////////////////////////////////////////////////

template <typename T>
class Sort{

 private:

  void swap(T* first,
      T* last);
```

```
  void choose_pivot(T array[],
        int first,
        int last);

  void partition(T array[],
        int first,
        int last,
        int& pivotIndex,
        bool order);

 public:

  Sort(void);

  void quicksort(T array[],
        int first,
        int last,
        bool order);

  void selection(T array[],
        int first,
        int last,
        bool order);

     void insertion(T array[],
        int first,
        int last,
        bool order);

  void bubble(T array[],
     int first,
     int last,
     bool order);

  ~Sort(void);
};

//////////////////////////////////////////////////////////////////////////////
//**** OVERLOAD OPERATORS ****
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
//**** PRIVATE METHODS ****
//////////////////////////////////////////////////////////////////////////////
template <typename T>
void Sort<T>::swap(T* first,
       T* last){
 T temp;
 temp = *first;
 *first = *last;
 *last = temp;
}

template <typename T>
void Sort<T>::choose_pivot(T array[],
        int first,
        int last){

 int index_pivot = ((first + last) / 2);

 swap (&array[first], &array[index_pivot]);
}

template <typename T>
void Sort<T>: artition(T array[],
      int first,
      int last,
      int& pivotIndex,
      bool order = false){
 int index_last;
 int index_unknown;
```

```
 if (!order){
  // Place pivot in array
  choose_pivot(array, first, last);

  // Copy pivot
  T pivot = array[first];

  // Index of last item
  index_last = first;

  // Index of next item after the first item
  index_unknown = first + 1;

  // Move one item at a time until region is empty
  for (; index_unknown <= last; index_unknown++){

   // Move item from unknown to proper region
   if (array[index_unknown] < pivot){

    index_last++;

    swap(&array[index_unknown], &array[index_last]);

   }// end if
  }// end for

  // Place pivot into proper position and indicate its location
  swap(&array[first], &array[index_last]);

  pivotIndex = index_last;

 }else{

 }// end if

}

//////////////////////////////////////////////////////////////////////////
//**** PUBLIC METHODS ****
//////////////////////////////////////////////////////////////////////////

//**** Constructors ****/

template <typename T>
Sort<T>::Sort(void){
 cout << '[Sort (default)]' << endl;
}

//**** Get Methods ****/

//**** Set Methods ****/

//**** Display Methods ****/

//****  Methods ****/

template <typename T>
void Sort<T>::quicksort(T array[],
        int first,
        int last,
        bool order = false){

 int pivotIndex = 0;

 if (!order){

  if (first < last){

   // Create partition
   partition(array, first, last, pivotIndex, order);

   // Sort regions
```

```
   quicksort(array, first, pivotIndex - 1, order);

   quicksort(array, pivotIndex + 1, last, order);
  }

 }else{

 }

}

template <typename T>
void Sort<T>::selection(T array[],
      int first,
      int last,
      bool order){

 int i, j, min;

 i = first;

 for (; i < last; i++){

  min = i;

  for (j = i + 1; j < last; j++){

   if (array[j] < array[min]){

    min = j;

   }// end if

  }//end for

  swap(array[i], array[min]);

 }//end for

}

template <typename T>
void Sort<T>::insertion(T array[],
      int first,
      int last,
      bool order){

 int i, j, min;

 T tmp_to_insert;

 for (i = first + 1; i < last; i++){

  tmp_to_insert = array[i];

  j = i;

  for (; j > first && array[ j - 1] > tmp_to_insert; j--){

   array[j] = array[j - 1];

  }// end for

  array[j] = tmp_to_insert;

 }//end for

}

template <typename T>
void Sort<T>::bubble(T array[],
      int first,
      int last,
```

```
     bool order){

 int i,j,min;

 i = last - 1;

 j = first + 1;

 for (; i>= first; i--){

  for (; j <= i; i++){

   if (array[j - 1] > array[j]){

    swap(array[j - 1], array[j]);

   }//end if

  }// end for

 }//end for

}

//**** Destructor ****/

template <typename T>
Sort<T>::~Sort(void){
 cout << '[~Sort(X)]' << endl;
}

#endif
```

input.txt:

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.

/** * operator == * @description: Compare two Items by their number * @param: in, Item1, Item2 * @return: istream& */ bool operator == (const Item& Item1, const Item& Item2){ return (Item1.frequency == Item2.frequency); } /** * operator != * @description: Compare two Items by their number * @param: in, Item1, Item2 * @return: istream& */ bool operator != (const Item& Item1, const Item& Item2){ return (Item1.frequency != Item2.frequency); } /** * operator <= * @description: Compare two Items by their number * @param: in, Item * @return: istream& */ bool operator <= (const Item& Item1, const Item& Item2){ bool b_rtn = false; if (Item1.frequency == Item2.frequency){ b_rtn = (Item1.data > Item2.data); }else{ b_rtn = (Item1.frequency > Item2.frequency); } return b_rtn; } /** * operator >= * @description: Compare two Items by their number * @param: in, Item1, Item2 * @return: istream& */ bool operator >= (const Item& Item1, const Item& Item2){ bool b_rtn = false; if (Item1.frequency == Item2.frequency){ b_rtn = (Item1.data > Item2.data); }else{ b_rtn = (Item1.frequency < Item2.frequency); } return b_rtn; } /** * operator < * @description: Compare two Items by their number * @param: in, Item1, Item2 * @return: istream& */ bool operator < (const Item& Item1, const Item& Item2){ return (Item1.frequency > Item2.frequency); } /** * operator > * @description: Compare two Items by their number. * @param: in, Item1, Item2 * @return: istream& */ bool operator > (const Item& Item1, const Item& Item2){ return (Item1.frequency < Item2.frequency); } Item::Item(void): frequency(0), data(''){ } Item::Item(const Item& new_item):

frequency(new_item.frequency),   data(new_item.data){ } Item::~Item(void){ }

Sort.cc:

```
/**
  * @Author: Alejandro G. Carlstein
  * @Course: CS240
  */

#ifndef SORT_CC
#define SORT_CC

#include <iostream>
#include <cstdlib>

//#include 'functions.h'

using namespace std;

//////////////////////////////////////////////////////////////////
// TEMPLATE CLASS SORT   ////////////////////////////////////////
//////////////////////////////////////////////////////////////////

template <typename T>
class Sort{

 private:

  void swap(T* first,
      T* last);

  void choose_pivot(T array[],
        int first,
        int last);

  void partition(T array[],
        int first,
        int last,
        int& pivotIndex,
        bool order);

 public:

  Sort(void);

  void quicksort(T array[],
        int first,
        int last,
        bool order);

  void selection(T array[],
        int first,
        int last,
        bool order);

     void insertion(T array[],
        int first,
        int last,
        bool order);

  void bubble(T array[],
     int first,
     int last,
     bool order);

  ~Sort(void);
};

//////////////////////////////////////////////////////////////////////////
//**** OVERLOAD OPERATORS ****
```

```
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
//**** PRIVATE METHODS ****
//////////////////////////////////////////////////////////////////////////////
template <typename T>
void Sort<T>::swap(T* first,
       T* last){
 T temp;
 temp = *first;
 *first = *last;
 *last = temp;
}

template <typename T>
void Sort<T>::choose_pivot(T array[],
        int first,
        int last){

 int index_pivot = ((first + last) / 2);

 swap (&array[first], &array[index_pivot]);
}

template <typename T>
void Sort<T>: artition(T array[],
      int first,
      int last,
      int& pivotIndex,
      bool order = false){
 int index_last;
 int index_unknown;

 if (!order){
  // Place pivot in array
  choose_pivot(array, first, last);

  // Copy pivot
  T pivot = array[first];

  // Index of last item
  index_last = first;

  // Index of next item after the first item
  index_unknown = first + 1;

  // Move one item at a time until region is empty
  for (; index_unknown <= last; index_unknown++){

   // Move item from unknown to proper region
   if (array[index_unknown] < pivot){

    index_last++;

    swap(&array[index_unknown], &array[index_last]);

   }// end if
  }// end for

  // Place pivot into proper position and indicate its location
  swap(&array[first], &array[index_last]);

  pivotIndex = index_last;

 }else{

 }// end if

}

//////////////////////////////////////////////////////////////////////////////
//**** PUBLIC METHODS ****
//////////////////////////////////////////////////////////////////////////////
```

```
//**** Constructors ****/

template <typename T>
Sort<T>::Sort(void){
 cout << '[Sort (default)]' << endl;
}

//**** Get Methods ****/

//**** Set Methods ****/

//**** Display Methods ****/

//****  Methods ****/

template <typename T>
void Sort<T>::quicksort(T array[],
        int first,
        int last,
        bool order = false){

 int pivotIndex = 0;

 if (!order){

  if (first < last){

   // Create partition
   partition(array, first, last, pivotIndex, order);

   // Sort regions

   quicksort(array, first, pivotIndex - 1, order);

   quicksort(array, pivotIndex + 1, last, order);
  }

 }else{

 }

}

template <typename T>
void Sort<T>::selection(T array[],
      int first,
      int last,
      bool order){

 int i, j, min;

 i = first;

 for (; i < last; i++){

  min = i;

  for (j = i + 1; j < last; j++){

   if (array[j] < array[min]){

    min = j;

   }// end if

  }//end for

  swap(array[i], array[min]);

 }//end for

}
```

```
template <typename T>
void Sort<T>::insertion(T array[],
      int first,
      int last,
      bool order){

 int i, j, min;

 T tmp_to_insert;

 for (i = first + 1; i < last; i++){

  tmp_to_insert = array[i];

  j = i;

  for (; j > first && array[ j - 1] > tmp_to_insert; j--){

   array[j] = array[j - 1];

  }// end for

  array[j] = tmp_to_insert;

 }//end for

}

template <typename T>
void Sort<T>::bubble(T array[],
      int first,
      int last,
      bool order){

 int i,j,min;

 i = last - 1;

 j = first + 1;

 for (; i>= first; i--){

  for (; j <= i; i++){

   if (array[j - 1] > array[j]){

    swap(array[j - 1], array[j]);

   }//end if

  }// end for

 }//end for

}

//**** Destructor ****/

template <typename T>
Sort<T>::~Sort(void){
 cout << '[~Sort(X)]' << endl;
}

#endif
```

input.txt:

```
words
```

The
taxi
took
a
sudden
right
turn
I
heard
gravel
under
the
tires
and
saw
a
brightly
lit
house
silhouetting
dark
trees
and
so
surmised
than
the
driver
had
turned
into
someones
drive
I
knew
that
Derek
had
done
well
for
himself
since
moving
down
south
but
a
tree
lined
drive
was
impressive
It
was
even
more
impressive
that
it
took
nearly
five
minutes
to
actually
wind
our
way
to
the
top
We

pulled
up
in
front
of
a
three
storied
granite
house
It
looked
to
be
several
hundred
years
old
with
pillars
either
side
of
the
tall
hall
door
and
large
Georgian
windows
It
was
probably
built
as
the
country
seat
of
some
lord
or
wealthy
industrialist
I
began
to
think
that
the
driver
had
brought
me
to
the
wrong
place
Are
you
sure
this
is
Morcome
Hall
I
asked
This
is
the
place
guv

came
the
gruff
reply
I
like
to
travel
light
So
all
I
only
had
a
small
hold
all
with
a
couple
of
changes
of
clothes
and
toiletries
in
it
It
seemed
pretty
meagre
compared
to
the
granite
might
of
this
imposing
building
Reaching
to
the
back
seat
behind
me
I
pulled
my
bag
into
the
front
of
the
car
I
looked
out
the
window
again
as
I
slipped
the
strap
over
my
shoulder

This
was
not
what
Id
been
expecting
when
Derek
an
old
school
friend
of
my
big
brothers
had
suggested
Id
come
down
south
and
do
some
work
for
him
At
the
time
the
offer
had
seemed
infinitely
preferable
to
staying
unemployed
at
home
But
now
when
it
came
to
taking
the
last
few
steps
Id
suddenly
gotten
very
cold
feet
Derrick
might
have
done
well
for
himself
recently
But
surely
he
couldnt
have

done
so
well
that
he
could
be
living
in
a
millionaires
mansion
I
asked
the
driver
again
if
this
was
the
correct
address
He
was
a
bit
short
in
his
answer
but
I
had
no
option
except
to
believe
that
he
knew
where
he
was
Id
been
lost
ever
since
wed
left
the
train
station
I
paid
the
fare
got
out
and
looked
around
It
was
a
couple
of
hours
after
sunset
so

I
couldnt
make
out
much
of
the
grounds
But
the
neighbours
either
didnt
have
any
lights
on
or
lived
a
fair
bit
away
I
looked
up
again
at
the
house
as
the
taxi
crunched
gravel
and
pulled
away
The
person
who
lives
here
must
have
a
fortune
So
either
Derrick
had
struck
gold
or
hed
robbed
a
bank
because
there
was
no
way
he
could
have
made
that
much
money
out
of
landscape

gardening
Even
with
the
inflated
prices
they
charge!
Feeling
very
unsure
of
myself
I
climbed
the
steps
to
the
door
There
seemed
to
be
a
party
going
on
as
I
could
hear
loud
music
thumping
away
in
the
depths
of
the
house
Ringing
the
doorbell
seemed
a
waste
of
time
with
all
that
racket
going
on
but
I
did
so
anyway
I
waited
a
couple
of
minutes
There
was
no
answer
I
waited

some
more
Still
nobody
came
So
I
rang
the
bell
again
And
waited
and
thought
that
I
should
go
away
But
I
had
no
place
else
to
go
to
except
back
home
and
I
didnt
want
to
admit
defeat
and
go
running
back
to
my
mother
with
my
tail
between
my
legs
Then
I
noticed
that
the
door
was
slightly
ajar
I
pushed
at
it
gently
and
it
swung
open
a
few
inches

letting
a
wedge
of
yellow
light
spring
out
I
cleared
my
throat
loudly
then
felt
foolish
I
pushed
the
door
again
and
looked
past
it
into
the
hall
All
I
could
see
was
the
black
and
white
tiles
of
the
floor
and
a
small
mahogany
hall
table
against
the
wall
with
a
gold
framed
mirror
hanging
over
it
There
was
nobody
in
sight
Looking
around
one
last
time
to
make
sure
I
was

alone
I
gingerly
put
one
hand
on
to
the
door
and
pushed
again
It
swung
open
fully
and
I
stepped
into
the
large
hall
The
tiles
on
the
floor
were
brightly
polished
reflecting
the
crystal
chandelier
hanging
from
the
high
ceiling
The
hall
was
two
stories
high
and
a
wide
marble
staircase
climbed
up
the
middle
to
a
balcony
that
ran
the
width
of
the
hall
Several
dark
wooden
doors
could
be
seen

on
the
balcony
from
where
I
stood
but
they
were
all
closed
Now
that
Id
entered
I
didnt
know
whether
to
wait
here
to
be
discovered
or
to
do
some
exploring
There
were
two
double
doors
on
either
side
of
the
hall
and
another
smaller
single
door
at
the
back
underneath
the
balcony
All
made
from
the
same
dark
stained
wood
and
all
firmly
closed
It
seemed
especially
empty
with
all
the
party

noises
that
emanated
from
further
back
in
the
house
I
decided
to
play
for
time
by
closing
the
front
door
behind
me
though
I
left
the
lock
on
the
latch
as
Id
found
it
When
I
turned
back
around
again
I
discovered
a
young
lady
dressed
a
long
sleeved
black
dress
over
which
she
wore
a
white
frilly
apron
With
a
frilly
maids
cap
pinned
to
her
black
hair
I
didnt
know
if

she
was
a
real
maid
or
someone
attending
a
fancy
dress
party
dressed
up
as
one
The
skirt
of
her
dress
was
rather
short
and
I
thought
I
could
just
make
out
the
tops
of
her
stockings
peeping
from
under
the
hem
Her
legs
were
long
and
slender
and
her
petite
feet
were
bound
by
thin
leather
straps
and
her
heels
rested
on
four
inch
stilettos
She
was
holding
the
handle
of
the

```
left
hand
door
which
was
now
half
open
and
looking
directly
at
me
Hhello
I
said
Who
are
you
she
asked
Im
David
I
replied
Im
a
friend
of
Dereks
He
invited
me
to
come
Well
youre
late
the
partys
already
started
she
said
I
was
about
to
explain
that
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.