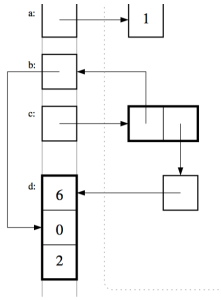
Write code to produce the memory structure shown in the diagram to the right.



Solution:

```
int* a = new int;
*a = 1;
int* b;
int*** c = new int**[2];
c[0] = &b;
c[1] = new int*;
int d[3];
d[0] = 6;
d[1] = 0;
d[2] = 2;
*c[1] = d;
b = &d[1];
```

Write code to print the current year to `std::cout` using ***ALL*** of the variables (`a`, `b`, `c`, and `d`).

Solution:

```
std::cout << d[2] << *b << *a << **c[1] << std::endl;
```

Finally, write code to clean up the dynamically-allocated memory so we don't have any leaks.

Solution:

```
delete a;
delete c[1];
delete [] c;
```

Solution:

```
std::vector<std::string> compound_detector(const std::vector<std::string> &words) {
  std::vector<std::string> answer;
  // loop over each word, testing to see if it is a compound word
  for (int w = 0; w < words.size(); w++) {
    bool found = false;
    for (int x = 0; !found && x < words.size(); x++) {
      for (int y = 0; !found && y < words.size(); y++) {
        // 2 word combinations
        if (words[w] == words[x]+words[y]) {
          answer.push_back(words[w]);
          found = true;
        }
        for (int z = 0; !found && z < words.size(); z++) {
          // 3 word combinations
          if (words[w] == words[x]+words[y]+words[z]) {
            answer.push_back(words[w]);
            found = true;
          }
        }
      }
    }
  }
  return answer;
}
```

```
       1        0        0        0        0        0        0
       1        1        1        1        1        1        1
       1        2        4        8       16       32       64
```



```
       1        3        9       27       81      243      729
       1        4       16       64      256     1024     4096
```

Try to write this function *without using* the `pow` function.

Solution:

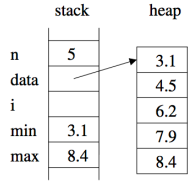
```
std::vector<std::vector<int> > make_power_matrix(int rows, int cols) {
  std::vector<std::vector<int> > answer;
  for (int r = 0; r < rows; r++) {
    std::vector<int> helper;
    int val = 1;
    for (int c = 0; c < cols; c++) {
      helper.push_back(val);
      val *= r;
    }
    answer.push_back(helper);
  }
  return answer;
}
```

Finish the program below to read in a positive integer `n` followed by n floating point numbers from the keyboard (`std::cin`). The code prints out the absolute value of each of the input numbers as well as the minimum and maximum absolute values. If the user types:

```
5   3.1  -4.5  6.2  7.9  -8.4
```

The program should produce the memory diagram on the right, and output this to the console (`std::cout`):

```
absolute values: 3.1 4.5 6.2 7.9 8.4
min: 3.1
max: 8.4
```



The main function is responsible for input and output. A helper function will edit and process the data. *Note:* Make sure the finished program does not have any memory leaks.

```
int main() {
```

Solution:

```
  int n;
  std::cin >> n;
  float *data = new float[n];
  int i;
  for (i = 0; i < n; i++) {
     std::cin >> data[i];
  }
  float min;
  float max;

  find_min_and_max(data,n,min,max);
  std::cout << "absolute values: ";
  for (i = 0; i < n; i++) { std::cout << data[i] << " "; }
  std::cout << std::endl;
  std::cout << "min: " << min << std::endl;
  std::cout << "max: " << max << std::endl;
```

Solution:

```
  delete [] data;

  return 0;
}
```

Now implement the helper function `find_min_and_max`:

Solution:

```
void find_min_and_max(float data[], int n, float &min, float &max) {
  for (int i = 0; i < n; i++) {
    if (data[i] < 0)
       data[i] = -data[i];
    if (i == 0 || data[i] < min)
       min = data[i];
    if (i == 0 || data[i] > max)
       max = data[i];
  }
}
```

2 Text Justification Redux [/18]

Write a function named `print_square` that takes in a single argument, an STL `string`, and reformats that text to fit in the *smallest square box*, surrounded by a border of stars. Unlike Homework 1, we won't worry about fitting complete words or hyphenation. Just break the words when you get to the end of the row. A few sample calls to the function are shown below, and the output to `std::cout` of each call is shown on the right.

```
*********
*the qui*
*ck brow*
*n fox j*
*umped o*
*ver the*
* lazy d*
*ogs    *
*********
```

```
**********
*Twinkle, tw*
*inkle, litt*
*le star, ho*
*w I wonder *
*what you ar*
*e. Up above*
* the world *
*so high, li*
*ke a diamon*
*d in the sk*
*y.         *
**********
```

```
print_square("Here is an example.");
print_square("the quick brown fox jumped over the lazy dogs");
print_square("Twinkle, twinkle, little star, how I wonder what you are. Up above the " +
```

Solution:

```
void print_square(const std::string& sentence) {
  // calculate dimensions of smallest square
  int dim = ceil(sqrt(sentence.size()));
  std::cout << std::string(dim+2,'*') << std::endl;
  // helper variable to select next character of the sentence
  int k = 0;
  for (int i = 0; i < dim; i++) {
    std::cout << "*";
    for (int j = 0; j < dim; j++) {
      // make sure we don't attempt to access characters beyond the end of the string
      if (k < sentence.size()) {
        std::cout << sentence[k];
        k++;
      } else {
        std::cout << " ";
      }
    }
    std::cout << "*" << std::endl;
  }
  std::cout << std::string(dim+2,'*') << std::endl;
}
```

Solution:

```
void HasLetter(const std::vector<std::string> &words, char letter, std::vector<std::string> &selected) {
  // clear out the vector of any previous answer
  selected.clear();
  for (int i = 0; i < words.size(); i++) {
    // use a boolean to check for the letter
    // (in case there are repeated letters)
    bool flag = false;
    for (int j = 0; j < words[i].size(); j++) {
      if (words[i][j] == letter) {
        flag = true;
        break;
      }
    }
    if (flag)
      selected.push_back(words[i]);
  }
```

Iclicker

1.Which of the following statements is false about keyword const and the symbol &
Compiler error message related to mismactches in const &reference are long and sometimes confusing to decipher, so its ok to ignore them.

2.What is not a reason for making the member varabiles of c++ class private?
If you don't specify, everything about c++ class will be public. In consrast, by default everything is private for a struct.

3.Which of the following is a correct use of the hash or pound symbol#
Simple if/else logic can be performed with the preprocessor before compliation & linking begins.

4.Which of the following is not a use of &
It means follow the pointer.

5.10 15/n 150 15 /n 150 25

6.Which statement is true about good software engineering class design?
None of the above.

7.Which of the following is false
If you have a big dataset and you care about performance, you should write your own sort. Because stl vector sort is inefficient.

8.Which of the following is not an important consideration when call push_back
Everything is important

9.Which of the following statements is true about heap vs stack
If your code doesn't explicity use the new your data will always be on the stack.

Solution:

```
class Customer {
public:
  // CONSTRUCTOR
  Customer(const std::string& name);
  // ACCESSORS
  const std::string& getName() const;
  const std::string& getStylist() const;
  const Date& lastAppointment() const;
  int numAppointments() const;
  // MODIFIERS
  void hairCut(const Date &d,const std::string &stylist);
private:
  // REPRESENTATION
  std::string customer_name;
  std::string preferred_stylist;
  std::vector<Date> appointments;
};

// helper function for sorting
bool stylist_then_last_appointment(const Customer &c1, const Custome
```

The first const reference, '**const** std::string**&** Customer::getName() const;', means that the returned value of the getName() member function is not a copy, and cannot be modified.

The second const, 'const std::string& Customer::getName() **const**;' at the end of the function name is to state that the function does not modify any class member variables. If you try to change a variable in a const function, it will error.

Solution:

```
class Dishwasher {
public:
  // CONSTRUCTOR
  Dishwasher(int max_plates);
  // PRINT & ACCESSORS
  void printContents() const;
  int completeUtensilSets() const;
  // MODIFIERS
  bool addPlate(const std::string& color);
  bool addCup();
  void addFork()  { forks++; }
  void addSpoon() { spoons++; }
  void addKnife() { knives++; }
private:
  // PRIVATE HELPER FUNCTION
  bool valid_counts(int p, int c) const;
  // REPRESENTATION
  int max_plates;
  std::vector<std::string> plates;
  int cups;
  int forks;
  int knives;
  int spoons;
};
```

```
// MODIFIERS
bool Dishwasher::addPlate(const std::string& color) {
  if (valid_counts(plates.size()+1,cups)) {
    plates.push_back(color);
    return true;
  }
  return false;
}
bool Dishwasher::addCup() {
  if (valid_counts(plates.size(),cups+1)) {
    cups++;
    return true;
  }
  return false;
}
```

1.2 Dishwasher Class Implementation [/18]

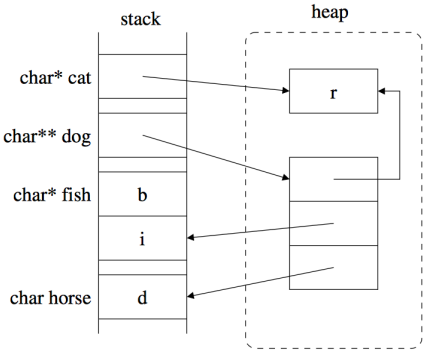
Now implement the member functions, as they would appear in the corresponding `dishwasher.cpp` file.

Solution:

```
// CONSTRUCTOR
Dishwasher::Dishwasher(int max_p) {
  cups = forks = knives = spoons = 0;
  max_plates = max_p;
}

// PRINT & ACCESSORS
void Dishwasher::printContents() const {
  std::cout << plates.size() << " plate(s):";
  for (int i = 0; i < plates.size(); i++) { std::cout << " " << plates[i]; }
  std::cout << std::endl;
  std::cout << cups << " cup(s)" << std::endl;
  std::cout << forks+knives+spoons << " utensil(s)" << std::endl;
}
int Dishwasher::completeUtensilSets() const {
  return std::min(std::min(forks,knives),spoons);
}
```

```
char* cat;
char** dog;
char fish[2];
char horse;
dog = new char*[3];
dog[0] = new char;
fish[0] = 'b';
fish[1] = 'i';
dog[1] = &fish[1];
dog[2] = &horse;
cat = dog[0];
*cat = 'r';
horse = 'd';
```

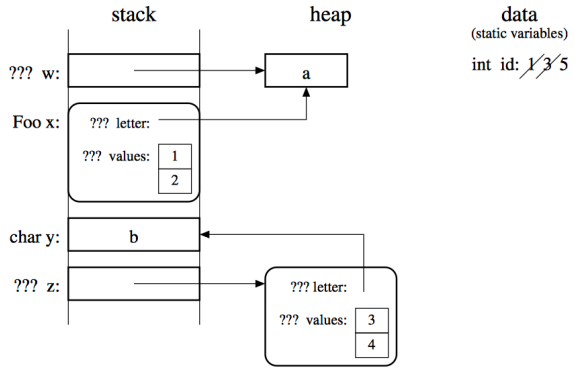


Solution:

Solution:

```
class Foo {
public:
  Foo(char* l);
private:
  char* letter;
  int values[2];
};

Foo::Foo(char *l) {
  static int id = 1;
  letter = l;
  values[0] = id;
  values[1] = id+1;
  id += 2;
}
```




```
int main() {
  char* w = new char;
  *w = 'a';
  Foo x(w);
  char y = 'b';
  Foo* z = new Foo(&y);
}
```

Solution:

```
// CONSTRUCTOR
Customer::Customer(const std::string &name) {
  customer_name = name;
}

// ACCESSORS
const std::string& Customer::getName() const {
  return customer_name;
}
const std::string& Customer::getStylist() const {
  return preferred_stylist;
}
const Date& Customer::lastAppointment() const {
  return appointments.back();
}
int Customer::numAppointments() const {
  return appointments.size();
}

// MODIFIER
void Customer::hairCut(const Date &d,const std::string &stylist) {
  if (stylist != preferred_stylist) {
    std::cout << "Setting " << stylist << " as " << customer_name << "'s preferred stylist.
    preferred_stylist = stylist;
  }
  appointments.push_back(d);
}

// COMPARISON FUNCTION FOR SORTING
bool stylist_then_last_appointment(const Customer &c1, const Customer &c2) {
  return (c1.getStylist() < c2.getStylist() ||
          (c1.getStylist() == c2.getStylist() && c1.lastAppointment() < c2.lastAppointment
}
```