University of Calgary
Department of Electrical and Computer Engineering
ENCM 339: Programming Fundamentals
Lecture Instructors: Steve Norman, Mahmood Moussavi

Fall 2014 FINAL EXAMINATION SOLUTIONS
Saturday, December 13 — 12:00 noon to 3:00 pm
Locations: ICT 102, ICT 121, ICT 122

NAME (printed):

U of C ID NUMBER:

LECTURE SECTION
(01 was MWF at 1:00pm with Dr. Norman,
02 was MWF at noon with Dr. Moussavi):

SIGNATURE:

Please don't write anything within this box.

1	/ 10
2	/ 10
3	/ 10
4	/ 10
5	/ 20
6	/ 14
7	/ 20
8	/ 6
9	/ 10
10	/ 6
11	/ 4
TOTAL	/ 120

Instructions

- Please note that the official University of Calgary examination regulations are printed on page 1 of the *Examination Regulations and Reference Material* booklet that accompanies this examination paper. All of those regulations are in effect for this examination, except that you must write your answers on the question paper, not in the examination booklet.
- You may **not** use electronic calculators or computers during the examination.
- The examination is **closed-book**. You may not refer to books or notes during the examination, with one exception: you may refer to the *Examination Regulations and Reference Material* booklet that accompanies this examination paper.
- Some problems are relatively **easy** and some are relatively **difficult**. Go after the easy marks first.
- Write all answers on the question paper and hand in the question paper when you are done. Please do *not* hand in the *Examination Regulations and Reference Material* booklet.
- Please print or write your answers **legibly**. What cannot be read cannot be marked.
- If you write anything you do not want marked, put a large X through it and write "rough work" beside it.
- You may use the backs of pages for rough work.

PROBLEM 1 (*total of 10 marks*) For all of Questions 1–10 within this problem, you may assume that any needed library header files are included.

1. What is the output of the program listed to the right?
   (a) 1234
   [(b)] 2321
   (c) 2312
   (d) 2323

```
void fun(int n) {
  if(n < 3) {
    cout <<  n + 1;
    fun(n+1);
    cout <<  n ;
  }
  else
    return;
}

int main() {
    fun(1);
    cout << endl;
    return 0;
}
```

2. Consider the following statements, and assume that the constructor call for `fp` succeeds:

```
fstream  fp ("data.bin",  ios::out|ios::binary);
const char* s = "ENCM-339";
fp.write(s, 4);
fp.close();
```

   [(a)] The size of the file created by the above code fragment is 4 bytes.
   (b) The size of the file created by the above code fragment is 8 bytes.
   (c) The size of the file created by the above code fragment is 9 bytes.
   (d) None of the above statements is correct.

3. Which one of the following statements is **incorrect**:
   (a) A constructor can have a return value.
   (b) A destructor can be overloaded.
   (c) A destructor can have a return value.
   [(d)] All of the above.

Use the following class definition to answer questions 4 and 5:

```
class Point {
public:
  Point(double x=0, double y=0) { this->x = x; this->y = y; }
  double getx() { return x; }
  double gety() { return y; }
  void setx(double x) { this->x = x; }
  void sety(double y) { this->y = y; }
private:
  double x, y;
};
```

4. What is output of the following code snippet?

```
Point a(20, 30);
a.setx(a.getx()+a.getx());
cout << a.getx() << endl;
```

   (a) 20
   (b) 30
   [(c)] 40
   (d) 50
   (e) none of the above

5. Which of the following statements are correct to create an object of class `Point`?
   (a) `Point a(23);`
   (b) `Point b;`
   (c) `Point c(5, 5);`
   [(d)] all of (a), (b), and (c)
   (e) none of (a), (b), or (c)

Use the following program to answer questions 6 and 7:

```
1  #include <iostream>
2  using namespace std;
3  int main(void){
4      double b = 9, c =10, d = 23;
5      double a[] = {1, 2, 33, 4};
6      double* x[4] = {a, &b, &c, &d};
7      cout << x[0][2] << endl;
8      cout << "and " << **(x+2) << endl;
9      return 0;
10 }
```

6. Which one of the following statements is correct?
   - (a) The output printed by line 7 is: `1`
   - (b) The output printed by line 7 is: `10`
   - (c) The output printed by line 7 is: `23`
   - [(d)] The output printed by line 7 is: `33`
   - (e) none of the above

7. Which one of the following statements is correct?
   - (a) The output printed by line 8 is: `and 1`
   - (b) The output printed by line 8 is: `and 2`
   - [(c)] The output printed by line 8 is: `and 10`
   - (d) The output printed by line 8 is: `and 33`
   - (e) none of the above

8. Which of the following answers are correct with regards to this code snippet? Assume that `func` is part of a program in which `func` is called at least once. (Circle *all* answers that are correct.)

```
void func() {
  int f = 20;
  if ( f ) {
    int *temp = new int[25 -f];
    temp[20-f] = 20;
    cout << temp[0] << endl;
  }
}
```

   - (a) The code gives a compilation error.
   - (b) The statement `temp[20-f] = 20;` will cause a segmentation fault when the program runs.
   - [(c)] The code causes a memory leak.
   - [(d)] The code creates an array of 5 integers dynamically and puts a value of 20 in element 0.
   - (e) none of the above

9. Which of the following statements about *null pointers* are true? (Circle *all* statements that are correct.)
   - (a) A null pointer can point to a `char` with value `'\0'`, in statically-allocated memory.
   - (b) A null pointer can point to a `char` with value `'\0'`, at the end of any C-style string.
   - [(c)] If `p` is a pointer variable with a null pointer value, then the expression `p == 0` has a value of 1 in C and a value of `true` in C++.
   - [(d)] A null pointer definitely does not point to any data object.
   - (e) none of the above

10. Which of the following is true about a variable of type `void*`?
    - (a) It definitely does not point to any data object.
    - (b) It must point to a block of heap memory allocated by one of the C `malloc`, `calloc`, or `realloc` functions.
    - [(c)] It can hold the address of any object (`int`, `double`, structure object, etc.).
    - (d) none of the above

PROBLEM 2 (*total of 10 marks*) For all parts of this problem, you may assume that necessary library header files are included.

Part a (*4 marks*)**.** Consider a program made from these two functions ...

```
void create_array(int* p, int size)
{
  p = malloc(sizeof (int) * size);
  if (p == NULL) {
    printf("Error: out of memory...");
    exit(1);
  }
}
```

```
int main(void) {
  int* a, size = 3;
  create_array(a, size);
  a[0] = 99;
  a[1] = 991;
  a[2] = 9;
  return 0;
}
```

It is possible to build an executable from the program, but the program is very likely to crash when it runs. Briefly but clearly explain what the cause of the crash is.

The pointer a is not initialized in main, and the call to create_array does nothing to change that. The write to a[0] will likely cause a crash.

Briefly explain how to fix the problem.

create_array should send the address it gets from new back to main as a return value of type int*.
(Also, create_array really needs only one parameter—the size of the array.)

Part b (*3 marks*)**.** Consider the following program. Assume that all of the constructor and destructor functions of class Text are properly defined. Then answer the questions below the program listing.

```
class Text {
public:
    Text();
    Text(const char *s);
    ~Text();
private:
    int lengthM;
    char *storageM;
};
```

```
int main(void) {
  Text text1 ("Blue");
  {
    Text* text2 = new Text("Red");
    Text text3 ("AB");
  }
  Text *text4 = new Text [2];
  delete [ ] text4;
  return 0;
}
```

When the program runs, how many times does the default constructor of `Text` get called?

2 times. (Once for each element in the array allocated with new.)

How many times does the other constructor of `Text` get called?

3 times. (Arguments are "Blue", "Red", and "AB").

How many times does the destructor of `Text` get called?

4 times. (Once for text1, once for text3, and once for each of the array elements.)

Part c (*3 marks*)**.** Consider the following program.

```
int func (int n)
{
  int m;
  if (n == 0 || n == 1 || n == 2)
    m = 1;
  else
    m = func(n - 3) + func(n - 1);
  return m;
}
```

```
int main(void)
{
  int result = func(6);

  printf("result is: %d\n", result);
  return 0;
}
```

When the program runs, `m` will be assigned a value of 3 exactly once. What is the value of `n` when that happens, and at that moment, how many activation records for `func` are present on the stack?

Reading the code for func generates the table at right. m is the return value of func. m = 3 implies that n = 4.

main calls func with an argument of 6, and that call generates a call to func with an argument of 4. So there are 2 ARs for func when m = 3.

n	0	1	2	3	4	5	6
func(n)	1	1	1	2	3	4	6

What is the program output?

The table says func(6) is 6, so the output is

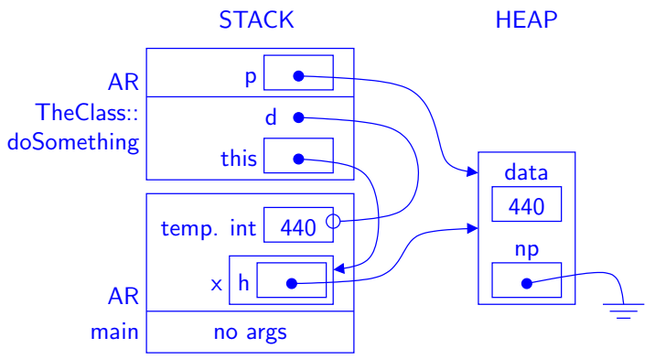
result is: 6

Hint: Tracing program execution down to every instance of a base case is not an efficient way to solve the problem. There is a much faster method!

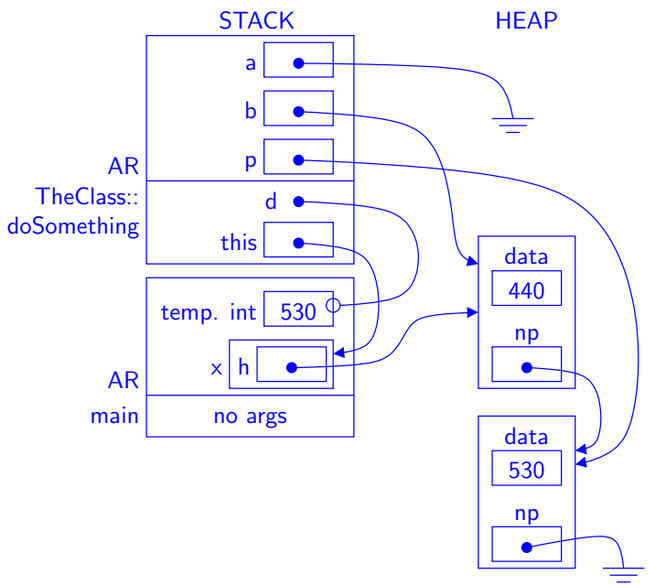
PROBLEM 3 *(10 marks)* Draw memory diagrams for `POINT ONE` and `POINT TWO` in the box to the space beside the listing.

```
struct TheStruct {
  int data;
  TheStruct *np;
};

class TheClass {
public:
  TheClass() : h(0) { }

  void doSomething(const int& d);

private:
  TheStruct *h;
};

void TheClass::doSomething(const int& d)
{
  TheStruct *p = new TheStruct;
  p->data = d;

  if (h == 0 || d <= h->data) {
    p->np = h;
    h = p;
    // POINT ONE
  }
  else {
    TheStruct *b = h;
    TheStruct *a = h->np;
    while(a != 0 && d > a->data) {
      b = a;
      a = a->np;
    }
    p->np = a;
    b->np = p;
    // POINT TWO

  }
}

int main()
{
  TheClass x;

  x.doSomething(440);
  x.doSomething(530);
  return 0;
}
```

POINT ONE:



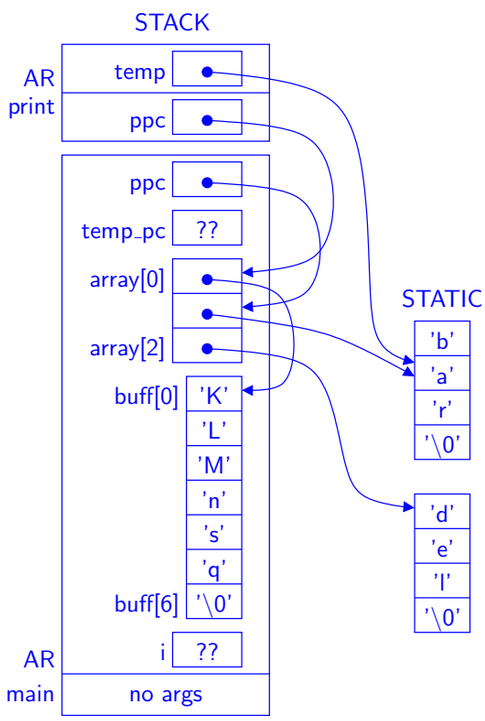
POINT TWO:



PROBLEM 4 (*total of 10 marks*)

Part a (*7 marks*)**.** Draw a memory diagram for `POINT ONE` in the following program. Include string constants used in `main`, such as `"bar"`, but do *not* include string constants used as first arguments to `printf`, such as `"<ONE>%s\n"`.

```
#include <stdio.h>

void print(char** ppc)
{
    char *temp;
    ppc--;
    (*ppc)++;
    temp = ppc[0];
    ppc[0] = ppc[1];
    ppc[1] = temp;

    /* POINT ONE */

    printf("<ONE>%s\n", *(ppc + 1));
    printf("<TWO>%s\n", *ppc);
    printf("<THREE>%c\n", *(*ppc + 1));
    printf("<FOUR>%c\n", ppc[0][1]);
    printf("<FIVE>%c\n", **ppc);
    printf("<SIX>%c\n", *ppc[2]);
}

int main(void)
{
    int i;
    char buff[ ] = "KLMnsq";
    char *array[3];
    char *temp_pc;
    char **ppc;

    array[0] = "bar";
    array[1] = buff;
    array[2] = "del";
    ppc = array;
    ppc = array + 1;
    print(ppc);

    return 0;
}
```



Part b (*3 marks*)**.** What is the output of the program of **part a**?

```
<ONE>ar
<TWO>KLMnsq
<THREE>L
<FOUR>L
<FIVE>K
<SIX>d
```

PROBLEM 5 (*total of 20 marks*) You can assume all necessary header files are included. Consider the following listings and answer the questions in **parts a**, **b**, **c**, and **d**.

```
struct Point {
  int id;
  double x, y;
};

typedef Point ListItem;

class PointList {
private:
  class Node {
  public:
    ListItem item;
    Node *next;
  };
  Node *headM;
public:
  PointList(): headM(0) { }
  PointList(const PointList& source);
  PointList& operator =(const PointList& rhs);
  ~PointList() ;

  void push_back(const ListItem& new_item);
  // PROMISES: Adds a node with an item equal to new_item
  //   at the end of the list.

  void pop_front();
  //  PROMISES: If list is empty, does nothing.
  //    Otherwise, removes the first node from the list.

  void print() const;
  //  PROMISES: If list is empty, does nothing.  Otherwise, prints
  //    the values of each point item on cout, one item per line,
  //    starting from the head.  A typical line of output might be
  //        id: 20  x: 9.8  y: 7.4
};
```

Example `main` that uses the above types ...

```
int main() {
  PointList list;
  ListItem x = {10, 5.5, 6.6};
  ListItem y = {20, 9.8, 7.4};
  ListItem z = {30, 4.5, 2.4};
  list.push_back(x);
  list.push_back(y);
  list.push_back(z);
  list.pop_front();
  list.print();
  return 0;
}
```

Part a (*4 marks*)**.** Write a definition for the member function `pop_front`.

```
void PointList::pop_front()
{
  if (headM != 0) {
    Node * old_head = headM;
    headM = old_head->next;
    delete old_head;
  }
}
```

Part b (*4 marks*)**.** Write a definition for the member function `print`. Do not worry about making the number of spaces or number of decimal places printed exactly match the example on the previous page.

```
void PointList::print() const
{
  for (Node *p = headM; p != 0; p = p->next)
    cout << "id: " << p->item.id
         << " x: "  << p->item.x
         << " y: "  << p->item.y << endl;
}
```

Part c (*6 marks*)**.** Write a definition for the member function `push_back`.

```
void PointList::push_back(const ListItem& item)
{
  Node *new_node = new Node;
  new_node->item = item;
  new_node->next = 0;
  if (headM == 0)
    headM = new_node;
  else {
    // Traverse to make pre point to the last node in the list.
    Node *pre = headM;
    while(pre->next != 0)
      pre = pre->next;

    // Make new_node become the new last node.
    pre->next = new_node;
  }
}
```

Part d (*6 marks*)**.** Write a definition for the copy assignment operator. (Note: "copy assignment operator" is another name for the function called the "assignment operator" in Section 02 lectures.)

```
PointList& PointList::operator =(const PointList& rhs)
{
  if (this == &rhs)
    return *this;

  // Get rid of nodes owned by *this.
  Node *p = headM;
  while (p != 0) {
    Node *delete_me = p;
    p = p->next;
    delete delete_me;
  }

  if (rhs.headM == 0) {
    headM = 0;  // Set up empty list.
    return *this;
  }

  Node *this_node = new Node;
  headM = this_node;
  const Node *rhs_node = rhs.headM;
  while (true) {
    this_node->item = rhs_node->item;
    rhs_node = rhs_node->next;
    if (rhs_node == 0)
      break;
    this_node->next = new Node;
    this_node = this_node->next;
  }
  this_node->next = 0;
  return *this;
}
```

PROBLEM 6 (*total of 14 marks*) Write C function definitions to match the given function prototypes and function interface comments. In both parts, your code *must not* call any library functions.

Part a (*7 marks*).

```
int count_matches(const char *s, int c);
// REQUIRES: s points to the beginning of a string.
//           c contains a character code that is not '\0'.
// PROMISES: Return value is the number of times c is in the string.
// EXAMPLES: count_matches("synchronous sequential circuits", 's') == 4
//           count_matches("boring string", '!') == 0
```

```
int count_matches(const char *s, int c)
{
  int result = 0;
  while (*s != '\0') {
    if (*s == c)
      result++;
    s++;
  }
  return result;
}

// Of course, there are other easy solutions, such as using an index
// variable i and checking s[i] against '\0 and c.
```

Part b (*7 marks*).

```
int maxmindiff(const int *a, int n);
// REQUIRES: n > 0, elements a[0] ... a[n-1] exist.
// PROMISES: Return value is difference between largest and smallest elements.
// EXAMPLE:  If foo is set up with  int foo[ ] = {-10, 2, 15, -3}; ,
//           then maxmindiff(foo, 4) == 25.
```

```
int maxmindiff(const int *a, int n)
{
  int min = a[0], max = a[0], i;
  for (i = 1; i < n; i++) {
    if (a[i] < min)
      min = a[i];
    else if (a[i] > max)
      max = a[i];
  }
  return max - min;
}
```

PROBLEM 7 (*total of 20 marks*) This problem concerns a vector-of-`doubles` class. Each class object uses three pointers to keep track of the storage and also the number of elements belonging to a vector.

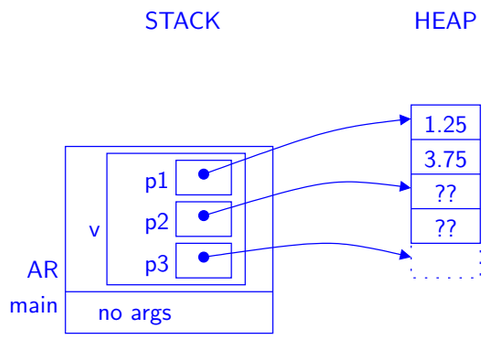
Reminder: When one pointer is subtracted from another, the result is a number of array elements, not a number of bytes. *C++ Fact:* If a null pointer is subtracted from another null pointer, the result is zero.

Part a (*5 marks*)**.** In the space beside the program listing, make a diagram for `point one`, showing the stack and the heap, but not static storage.

```
#include <iostream>
using std::size_t;

class VecD {
public:
  VecD() : p1(0), p2(0), p3(0) { }
  VecD(const VecD& src);
  VecD& operator=(const VecD& rhs);
  ~VecD() { delete [ ] p1; }

  void push_back(double d);
  double& at(size_t i) { return p1[i]; }

  const double& at(size_t i) const {
    return p1[i];
  }
  size_t size() const { return p2 - p1; };
  size_t capacity() const { return p3 - p1; };

private:
  void grow_cap(size_t new_cap);
  double *p1;
  double *p2;
  double *p3;
};

void VecD::grow_cap(size_t new_cap)
{
  double *larger = new double[new_cap];
  for (size_t i = 0; i < size(); i++)
    larger[i] = p1[i];
  p2 = larger + size();
  p3 = larger + new_cap;
  delete [ ] p1;
  p1 = larger;
}

void VecD::push_back(double d)
{
  if (size() == 0)
    grow_cap(4);
  else if (size() == capacity())
    grow_cap(2 * capacity());
  *p2 = d;
  p2++;
}

int main()
{
  VecD v;
  v.push_back(1.25);
  v.push_back(3.75);

  // point one

  for (size_t i = 0; i < v.size(); i++)
    std::cout << v.at(i) << std::endl;
  std::cout << "capacity: "
            << v.capacity() << std::endl;

  return 0;
}
```

PROGRAM OUTPUT:

```
1.25
3.75
capacity: 4
```



(v.p3 points to v.p1[4], which is just past the end of the array.)

PROBLEM 7, continued

For convenience, the class definition from the previous page is printed to the right.

The memory management policy is as follows:

- empty vector: no storage is allocated, and `p1 == 0`
- non-empty vector: `p1` points to the beginning of an array in the heap, large enough to hold all elements, and possibly larger
- at all times, addresses in `p2` and `p3` must be correct for the `size` and `capacity` functions

```
class VecD {
public:
  VecD() : p1(0), p2(0), p3(0) { }
  VecD(const VecD& src);
  VecD& operator=(const VecD& rhs);
  ~VecD() { delete [ ] p1; }

  void push_back(double d);
  double& at(size_t i) { return p1[i]; }

  const double& at(size_t i) const {
    return p1[i];
  }
  size_t size() const { return p2 - p1; };
  size_t capacity() const { return p3 - p1; };

private:
  void grow_cap(size_t new_cap);
  double *p1;
  double *p2;
  double *p3;
};
```

Part b (*6 marks*)**.** Write a definition for the copy constructor of `VecD`.

```
VecD::VecD(const VecD& src)
{
  if (src.p1 == 0)
    p3 = p2 = p1 = 0;
  else {
    size_t s = src.size();
    p1 = new double[s];
    p2 = p1 + s;
    p3 = p1 + s;
    for (size_t i = 0; i < s; i++)
      p1[i] = src.p1[i];
  }
}
```

Part c (*9 marks*)**.** Write a definition for the copy assignment operator of `VecD`. (Again, note: "copy assignment operator" is another name for the function called the "assignment operator" in Section 02 lectures.)

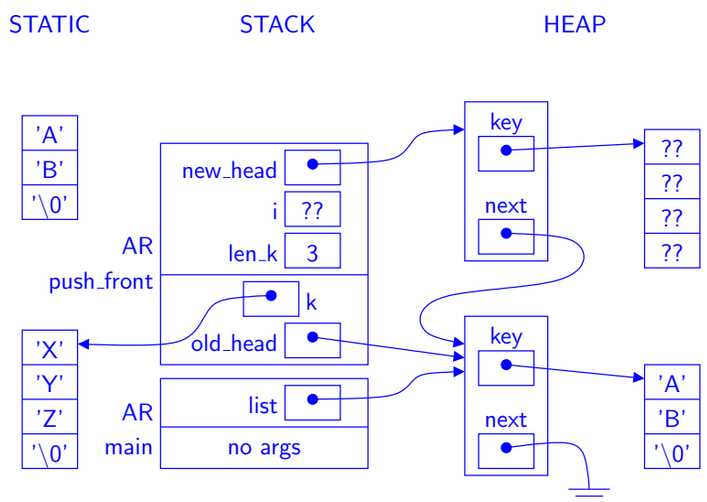
```
VecD& VecD::operator=(const VecD& rhs)
{
  if (this != &rhs) {
    if (rhs.p1 == 0) {
      delete [ ] p1;
      p3 = p2 = p1 = 0;
    }
    else {
      size_t s = rhs.size();
      if (s > capacity()) {
        delete [ ] p1;
        p1 = new double[s];
        p3 = p1 + s;
      }
      for (size_t i = 0; i < s; i++)
        p1[i] = rhs.p1[i];
      p2 = p1 + s;
    }
  }
  return *this;
}
```

PROBLEM 8 *(6 marks)*

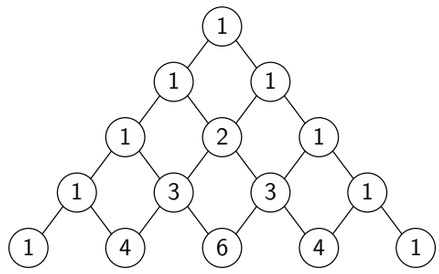
The program to the right demonstrates the use of `malloc` to build a linked list of strings in C.

In the space below the program listing, make a diagram for the *second time* the program reaches `point one`.

```
#include <stdlib.h>

struct node {
  char *key;
  struct node *next;
};
typedef struct node node_t;

node_t *push_front(node_t *old_head, char *k);
void print_all(const node_t *head);

int main(void) {
  node_t *list = NULL;
  list = push_front(list, "AB");
  list = push_front(list, "XYZ");
  return 0;
}

node_t *push_front(node_t *old_head, char *k) {
  int len_k = 0, i;
  node_t *new_head = malloc(sizeof(node_t));
  new_head->next = old_head;
  while (k[len_k] != '\0')
    len_k++;
  new_head->key = malloc(len_k + 1);

  // point one

  for (i = 0; i <= len_k; i++)
    new_head->key[i] = k[i];
  return new_head;
}
```



PROBLEM 9 *(10 marks) Pascal's triangle* is a famous arrangement of numbers in mathematics. The diagram to the right shows rows 0 to 4 of Pascal's triangle.



Let $P_{i,j}$ be a number in row i of the triangle, and let the index j go from 0 up to i. Then

$$P_{i,j} = \begin{cases} 1 & \text{if } j = 0 \text{ or } j = i \\ P_{i-1,j-1} + P_{i-1,j} & \text{otherwise} \end{cases}$$

For example, $P_{4,2} = P_{3,1} + P_{3,2} = 3 + 3 = 6$.

Your job is to complete the C program below, so that it will write the first N rows of Pascal's triangle into a text file, for values of N with $N > 0$ and $N < 16$. The number of rows and name of the output file should be command-line arguments. For example, if the name of the executable is `tri`, the command `./tri 6 foo.txt` should create the file `foo.txt` shown to the right of this paragraph.

```
row 0: 1
row 1: 1 1
row 2: 1 2 1
row 3: 1 3 3 1
row 4: 1 4 6 4 1
row 5: 1 5 10 10 5 1
```

The program should print an error message on `stderr` and call `exit` if it fails to open the output file. Substantial amounts of *partial credit* will be awarded to programs that write the correct number of lines of text to an output file, even if the lines don't contain numbers from Pascal's triangle!

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
  // These are ALL of the variables you need!
  int a[16], b[16], *current, *previous, *temp, i, j, nrow;
  FILE *fp;
  if (argc != 3) {
    fprintf(stderr, "Syntax is: %s NROW FILENAME\n,", argv[0]);
    exit(1);
  }
  nrow = atoi(argv[1]);
  if (nrow < 1 || nrow > 15) {
    fprintf(stderr, "Can't use %s as a number of rows.\n", argv[1]);
    exit(1);
  }
  // You can assume here that nrow > 0 and nrow < 16.

  // Try to open the file.
  fp = fopen(argv[2], "w");
  if (fp == NULL)  {
    fprintf(stderr, "Couldn't open file %s for output.\n", argv[2]);
    exit(1);
  }

  // Current row and previous row of integers.  We'll swap pointers
  // after every pass through the outer loop.
  current = a;
  previous = b;
  for (i = 0; i < nrow; i++) {
    fprintf(fp, "row %d:", i);

    // Compute and print numbers in row.
    for (j = 0; j <= i; j++) {
      current[j] = (j == 0 || j == i) ? 1 : (previous[j-1] + previous[j]);
      fprintf(fp, " %d", current[j]);
    }
    fputc('\n', fp);

    // Get ready for next pass.
    temp = previous;
    previous = current;
    current = temp;
  }
  fclose(fp);

  return 0;
}
```

PROBLEM 10 *(6 marks)* Write a recursive, divide-and-conquer C++ function definition for `is_sorted` ...

```
bool is_sorted(const double *a, int lo, int hi);
// REQUIRES: lo < hi; array elements a[lo] ... a[hi-1] exist.
// PROMISES: Returns true if lo + 1 == hi or if a[lo] ... a[hi-1]
//   are sorted from smallest to largest; otherwise returns false.
```

```
bool is_sorted(const double *a, int lo, int hi)
{
  // Base case only one element.
  if (lo == hi - 1)
    return true;

  // Split into two sub-arrays, check a[mid-1] <= a[mid],
  // then check sub-arrays with recursive calls.
  int mid = (lo + hi) / 2;
  return (a[mid-1] <= a[mid])
    && is_sorted(a, lo, mid)
    && is_sorted(a, mid, hi);
}
```

PROBLEM 11 *(4 marks)* Assume that all of the memory allocation, file open, and file write operations succeed in the following program. Also assume that the size of an `int` is 4 bytes.

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

int write_read(int *a, int n) {
  ofstream foo("quux.dat", ios::binary);
  int data[ ] = { 100, -200, 300 };
  foo.write(reinterpret_cast<const char*>(data), sizeof(data));
  foo.close();

  ifstream bar("quux.dat", ios::binary);
  bar.read(reinterpret_cast<char*>(a), n * sizeof(int));
  return bar.gcount();
}

int main() {
  int gc;
  int *p = new int[4];
  gc = write_read(p, 4);

  // point one

  delete [ ] p;
  return 0;
}
```

Draw a memory diagram for `point one`, showing only the stack and the heap.