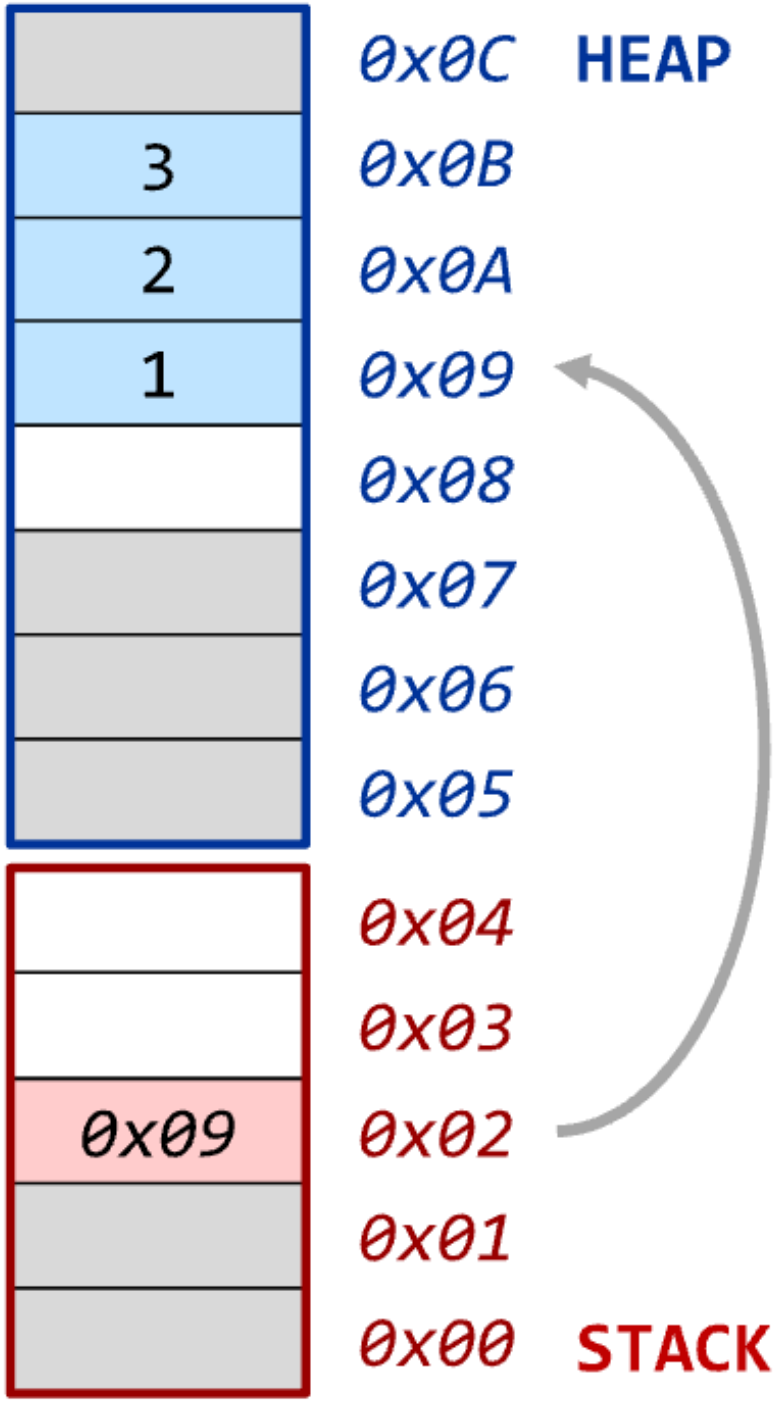
CS 101
Computer Programming and Utilization

Variable-Length Data Structures

Suyash P. Awate

Variable-Size Entities

- Motivation: in many programs:
  1. Size of array is unknown at compile time.
     e.g., involving arrays of floating-point numbers;
     e.g., involving strings storing names
     (Jet Li vs. Arnold Schwarzenegger)
  2. Size of the data structure changes with time.
     e.g., queue, library database
- We want to:
  1. Allocate only as much memory as is necessary
  2. De-allocate memory when not needed
- C++ program can allocate, and de-allocate, memory outside activation frame (call stack)
  - This memory region is called "heap"



Variable-Size Entities

- "Request" memory from heap using "new" keyword/operator
  - Format: For data type T, "new T" requests allocation of memory from heap
  - If request is granted (by operating system), then expression "new T" evaluates to address of newly allocated memory, else expression evaluates to NULL (indicating request failed)
  - In general: "new call-to-a-constructor-for-T" (allocate memory + call constructor)
- Example
  - "T * q = new T[n];" allocates heap memory to store an array of n elements of type T ('n' determined at run time)
  - "Book * p = new Book;" allocates heap memory for struct, calls constructor, allocates pointer variable p, stores address of struct in p

Variable-Size Entities

- Example

AF of main()
intptr : 24000
cptr : 24004

```
int main(){
   int* intptr = new int;
   char* cptr  = new char[3];
   *intptr     = 279;
   cptr[0]     = 'a';
   cptr[1]     = 'b';
   cptr[2]     = '\0';
}
```

Heap memory	
Address	Content
24000	279
24001	
24002	
24003	
24004	'a'
24005	'b'
24006	0
24007	
24008	

Variable-Size Entities

- Memory Leak

```
void function_witout_memory_leak ()
{
    int ptr[5];
    return;
}

void function_with_memory_leak ()
{
    int* ptr = new int(5);
    return;
}
```

Variable-Size Entities

- Static binding and dynamic binding
- If we do: "float a[20];",
  then definition of 'a' → static (compile-time) binding
  - Recall: array name is a constant pointer (cannot be reassigned)
- If we do "float * p = new float[20];",
  then definition of 'p' → dynamic (run-time) binding,
  but where 'p' can be reassigned
- If we do: "float * const q = new float[20];"
  then definition of 'q' → dynamic (run-time) binding,
  but, because of "const" keyword, 'q' cannot be reassigned

Variable-Size Entities

- Memory de-allocation
  - Explicitly free memory when no longer needed, using "delete" operator:
    "delete p;" for non-array data structures
    "delete[] q;" for array of data structures
  - Good practice:
    As soon as you execute "delete[] q;",
    let the next statement either set q to NULL ("q = NULL")
    or set q to point to another valid object (e.g., q = new … , or q = &obj)
  - Until then 'q' is dangling/wild pointer
    = pointer variable that doesn't point to valid object of appropriate type,
    e.g., unallocated memory
    - "Dangling" = unconnected
    - "Wild" = unpredictable

```
char *dp = NULL;
/* ... */
{
    char c;
    dp = &c;
}
  /* c falls out of scope */
  /* dp is now a dangling pointer */
```

```
int *func(void)
{
    int num = 1234;
    /* ... */
    return &num;
}
```

Variable

- Using dynamic arrays

```
void get(double*& a, int& n)
{ cout << "Enter number of items: ";  cin >> n;
  a = new double[n];
  cout << "Enter " << n << " items, one per line:\n";
  for (int i = 0; i < n; i++)
  { cout << "\t" << i+1 << ": ";
    cin >> a[i];
  }
}
void print(double* a, int n)
{ for (int i = 0; i < n; i++)
  cout << a[i] << " ";
  cout << endl;
}
int main()
{ double* a;     // a is simply an unallocated pointer
  int n;
  get(a,n);      // now a is an array of n doubles
  print(a,n);
  delete [] a;   // now a is simply an unallocated pointer again
  get(a,n);      // now a is an array of n doubles
  print(a,n);
}
```

```
Enter number of items: 4
Enter 4 items, one per line:
        1: 44.4
        2: 77.7
        3: 22.2
        4: 88.8
44.4 77.7 22.2 88.8
Enter number of items: 2
Enter 2 items, one per line:
        1: 3.33
        2: 9.99
3.33 9.99
```

Standard Library

- Programming languages provide a lot of functionality relating to algorithms that are used very very often
  - Carefully written code with respect to
    minimizing bugs,
    efficiency of computation,
    efficiency of memory usage,
    modularity,
    etc.
  - Shared through header files
- C standard library
  - en.wikipedia.org/wiki/C_standard_library
- C++ standard library
  - en.wikipedia.org/wiki/C++_Standard_Library

Standard Library

- String class
  - #include <string>
  - Use this to handle text data instead of char arrays
  - Handles memory management in a way that is transparent to programmer

```
string p = "abc", q ="defg", r;
r = p;

cout << p << "," << q << "," << r <<endl;  //prints ‘‘abc,defg,abc’’

getline(cin, p);

r = p + q;
string s = p + "123";          set r,s to "abcdefg" and "abc123"
```

Standard Library

set r,s to "abcdefg" and "abc123"

- String class
  - Comparing strings (based on lexicographic ordering)
    - Operators ==, <, > are defined
    - Can use expressions like: p == q, p < q, p >= q, etc.
  - Strings can be passed to functions
  - String elements. Sub-strings.

```
s[2] = s[3];              // indexing allowed. s will become ab1123.
cout << r.substr(2)       // substring starting at 2 going to end
     << s.substr(1,3)     // starting at 1 and of length 3
     << endl;             // will print out ‘‘cdefgb11’’, assuming r, s as above
```

Standard Library

- String class
  - Searching within a string

```
int i = p.find("ab");          // find from the beginning
int j = p.find("ab",1);        // find from position 1.
cout << i << ", " << j << endl; // will print out 0, 3
```

  - If string not found, then find() returns a constant "string::npos"
    - npos is a constant static member value with greatest possible value for an element of type size_t (npos = "no position")

```
string t;  getline(cin, t);
int i = p.find(t);
if(i == string::npos)
  cout << "String: "<< p << " does not contain "<< t << endl;
else
  cout << "String: "<< p << " contains "<< t << " from position "<< i << endl;
```

Standard Library

- Template class “vector”
  - #include <vector>
  - Creation: empty
  - Creation: with elements
  - Accessing elements
  - Append elements to end of vector (changes size)
  - Remove elements from end of vector (changes size)
  - We can insert and remove elements at any position from a vector (later)
  - Vector automatically adjusts and tracks its size: v3.size() will return 11

```
vector<int> v1;
vector<float> v2;
```

```
vector<short> v3(10);    // vector of 10 elements, each of type short.
vector<char>  v4(5,'a'); // vector of 5 elements, each set to 'a'.
vector<short> v5(v3);    // copy of v3.
```

```
v3[6] = 34;
v4[0] = v4[0] + 1;
```

```
v1.push_back(37);
v3.push_back(22);
v.pop_back()
```

Standard Library

- Template class “vector”
  - Can also change size

```
v.resize(newSize);
w.resize(newSize,newValue);
```

    - Call with newValue changes size, and
      if newSize is greater, then new elements replaced with newValue
  - Unlike arrays, you can assign one vector to another, e.g., “v = w;”
  - Instead of using [] operator to access array elements,
    vector class provides .at() function to access elements with bounds checking

```
vector<int> v;
for(int i=0; i<10; i++) v.push_back(i*10);

v.at(0) = v.at(1);
```

    - If index is outside bounds, program gives error message and halts

Standard Library

- Template class “vector”
  - Vector variables can be passed as arguments to functions, either by value or by reference

```
void print(vector<int> v){
  for(unsigned int i=0; i<v.size(); i++) cout << v[i] <<' ';
  cout << endl;
}
void read(vector<int> &v){
  for(unsigned int i=0; i<v.size(); i++) cin >> v[i];
}

int main(){vector<int> v(5); read(v); print(v);}
```

Standard Library

- Template class “vector”
  - Vectors of user-defined data types (e.g., structs)
  - Vectors of pointers
  - Multidimensional vectors
    - A vector of vectors
    - Can modify entire rows at once
    - Such a datatype allows “rows” if varying lengths

```
vector<V3> v3vec;
vector<Circle> circles;
vector<Circle*> circlevec;
```

```
vector<vector<int> > v;
vector<vector<int> > w(10, vector<int>(20));
```

```
w[0] = vector<int>(5);
```

Standard Library

- Template class “vector”
  - A matrix class created as a multidimensional vector; Enforcing all rows to be of same length

```
int main(){
  matrix D(10,10);          // 10 x 10 matrix

  for(int i=0; i<D.nrows(); i++){
    for(int j=0; j<D.ncols(); j++)
      D(i,j) = (i==j);      // access i,j th element
  }

  for(int i=0; i<D.nrows(); i++){
    for(int j=0; j<D.ncols(); j++)
      cout << D(i,j) <<' ';
    cout << endl;
  }
}
```

```
class matrix{
  vector<vector<double> > elements;
public:
  matrix(int m, int n) : elements(m, vector<double>(n)){}
  double &operator()(int i, int j){return elements[i][j];}
  int nrows(){return elements.size();}
  int ncols(){return elements[0].size();}
};
```

Standard Library

- Template class “vector”
  - Sorting a vector
  - #include <algorithm>

```
sort(v.begin(), v.end());
```

  - v.begin() = first element of vector
  - v.end() = last element of vector

  - If v is non-standard user-defined struct (e.g., student record, book record), we must specify a way to compare elements of v that can be used by sort()
    1. Can define an overloaded operator<, which sort will use implicitly
    2. Can define a non-member function, which can be passed to sort explicitly

```
struct student{
  int rollno;
  float marks;
  bool operator<(const student& rhs) const{ // used by the sort function
    return rollno < rhs.rollno;             // note the two const keywords
  }   int main(){
};      vector<student> svec;

        student s;
        while(cin >> s.rollno){
          cin >> s.marks;
          svec.push_back(s);
        }

        sort(svec.begin(), svec.end());         // will use operator< internally

        for(int i=0; i<svec.size(); i++)
          cout << svec[i].rollno << " " << svec[i].marks << endl;
      }
```

Standard Library

- Template class “vector”
  - Sorting students by marks

```
bool compareMarksFunction(const student &a, const student &b){
  return a.marks < b.marks;
}

sort(svec.begin(), svec.end(), compareMarksFunction);
```

  - Can have another function to sort by roll number, if desired

Standard Library

- Iterators
  - Vector class is a "container", because it is designed to hold multiple elements
    - In some ways, also the string class
  - Standard library allows a common way to iterate over elements of a container, using the "iterator"; its coding is also made highly computationally efficient
  - Iterator is like a generalized pointer to an element of container
  - Just as we could have the following for an array …

```
int A[10]
int* Aptr
for(Aptr = A; Aptr<A+10; Aptr++)   f(*Aptr);
```

  - … we have the following for a container:

```
for(vector<float>::iterator mi = marks.begin();
        mi != marks.end();
        ++mi)
    cout << *mi << endl;
```

a = b++

a = b++

Standard Library

- Iterators
  - Template class “vector”
  - We can add and remove elements at any position from a vector

```
vector<int> v;
for(int i=0; i<10; i++) v.push_back(i*10);

vector<int>::iterator vi = v.begin()+7;
v.insert(vi,100);                          // inserting into a vector

vi = v.begin() + 5;
v.erase(vi);                               // deleting an element
```

Practice Examples for Lab: Set 16

- 1

Consider the following code. Identify all errors in it.

```
int *ptr1, *ptr2, *ptr3, *ptr4;
ptr1 = new int;
ptr3 = new int;
ptr4 = new int;
ptr2 = ptr1;
ptr3 = ptr1;
*ptr2 = 5;
cout << *ptr2 << *ptr1 << endl;
delete ptr1;
cout << *ptr3 << *ptr4 << endl;
```

The possible errors are: memory leaks, dangling pointers (accessing memory that was allocated to us earlier but has since been deallocated), and referring to uninitialized variables.

Practice Examples for Lab: Set 16

- 2

Suppose you have a file that contains some unknown number of numbers. You want to read it into memory and print it out in the sorted order. Develop an extensible array data type into which you can read in values. Basically, the real array should be on the heap, pointed to by a member of your structure. If the array becomes full, you should allocate a bigger array. Be sure to return the old unused portion back to the heap. Write copy constructors etc. so that the array will not have leaks etc.

- 3

Write a code fragment that creates a 10×10 matrix stored as vector of vectors of doubles and initializes it to the identity matrix.

- 4

Write a program to multiply two matrices of arbitrary sizes represented as vector of vectors.

- 5

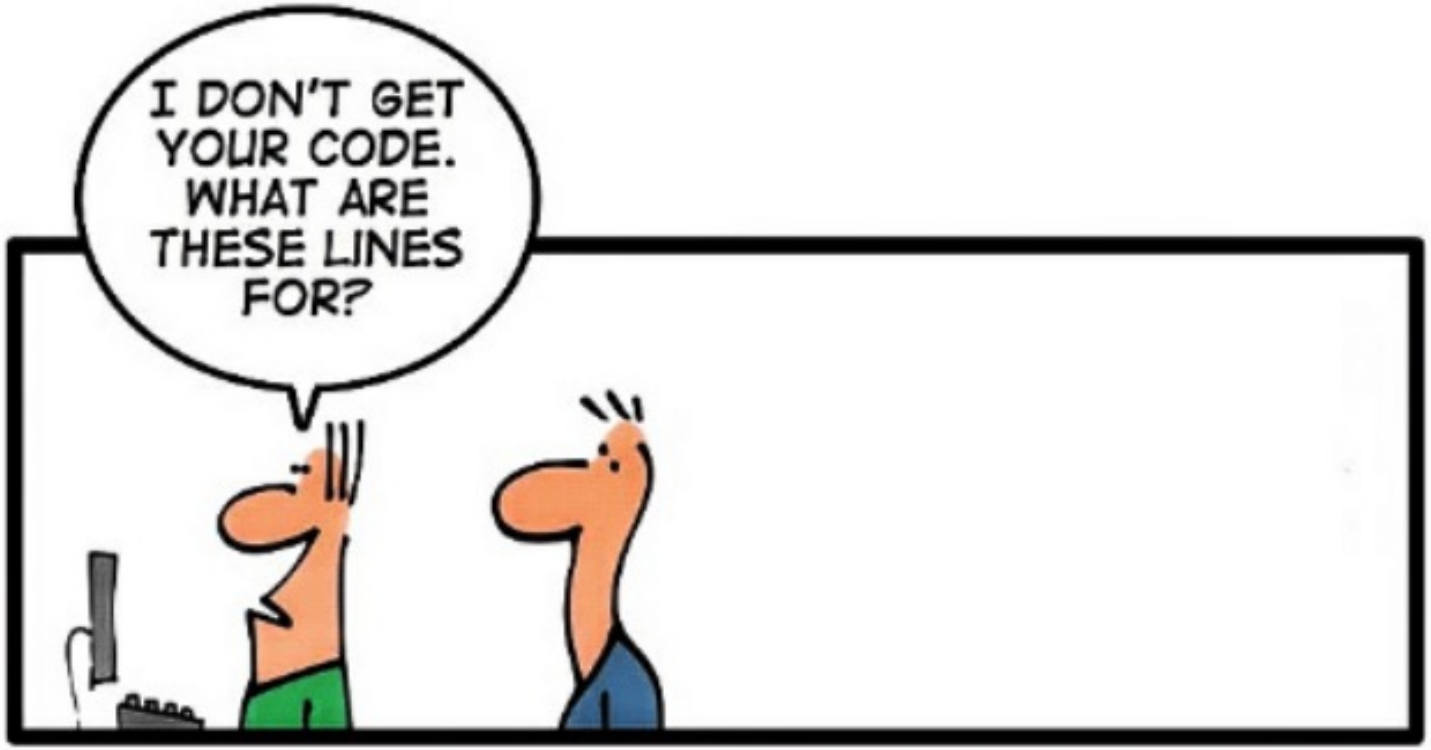
Write a program that prints out all positions of the occurrences of one string `pattern` inside another string `text`. Use appropriate functions from the `string` class.

Practice Examples for Lab: Set 16

- 6 Define a class LTM for storing lower triangular matrices, with signature as follows.

```
class LTM{
  vector<vector<double> > data;
public:
  LTM(int n);
  double getElem(int i, int j);
  void setElem(int i, int j, double v);
}
```

As you might guess the constructor constructs an LTM matrix with the given number of rows and columns. The member functions return the element at index `i,j` and assign the value `v` to the element at index `i,j` respectively. Note that if `j>i` then `getElem` must return 0. If `j>i` the `setElem` must do nothing and print a message. Give implementations of all the member functions. Of course, it will be much nicer to use array indices rather than `getElem` and `setElem`. The natural indexing operator is `[]` – but that can take only one index. So instead overload the `()` operator, so that the function arguments can be indices. Return a reference so that you can use assign to array elements as well as read array elements.


I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?


I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM


TOYOTA
COMMON SENSE
Just because you can, doesn't mean you should.

"With Great Power Comes Great Responsibility"

- Voltaire

The only way to learn a new programming language is by writing programs in it.

— Dennis Ritchie —

AZ QUOTES