

CSCI124

Applied Programming

Class Examples

Edition: Year 2017

Dr Heng Aik Koan

All examples developed in class should be read in conjunction with the lecture notes provided by UOW. Please do not pass these class examples and all your works to your juniors so that they can concentrate and can learn in class. Their futures depend on you. Guide them in the right way of learning



All rights *not* reserved



Lecture Schedule

Module 1	More on C++	
	Topic 1.1	C++ Revision and Programming Case study Examples on - bitwise manipulations - structures - union - text file processings.
	Topic 1.2	Advanced C++ File I / O, Random I/O and Binary I/O Examples on - binary file processings. - How to create? - How to access? - How to append? - How to update? - How to create a random access file (file sort)?
	Topic 1.3	Program Design Approaches and Structures, Compilation Processes Multi Files C++ Projects and Make Example on - how to create a make file in the Linux environment.
	Topic 1.4	Testing (White box and Black box). Testing Techniques and Debuggers
Module 2	Pointers and Dynamic Memory	
	Topic 2.1	Pointers Examples on - pointer concepts. - pointers and arrays (1D and 2D). - applications to cstrings. - applications using structure pointers.
	Topic 2.2	Pointers, Dynamic Types and Generic Code Examples on - void* arrays - the use of “new” keyword. - C-string versus void**
	Topic 2.3	Case study - Simple linked list

		Examples on linked list - data of int types - data of structure typrs - data of void* type
Module 3	Classes	
	Topic 3.1	Classes Examples on - How to create a class and declare objects? - A linked list class with void* - A class with multiple construcors - Communications between more than one classes
	Topic 3.2	Namespaces Examples on - How to design namespaces? - How to interact information in a namespace? - How to interact more than one namespaces in a file? - The use of “using”
	Topic 3.3	Exceptions Handling Examples on - the use of try / throw / catch - more then one catches - nested try block - user defined exception classes
	Topic 3.4	Static members Examples on - use of staitic variables - use of static functions
Module 4	Data Structures	
	Topic 4.1	Lists - Linked List (Single and Double) - Stacks, Queues, Priority Queues - Circularly Linked List (Single and Double) Examples on - linear linked list with head anf tail pointers - implementation of stack using linear linked list

		- implementation of queue using linear linked list - implementation of deque using doubly linked list - implementation of stack using circularly linked list - Using stack to do various conversions (binary, octal, hexa) - Using stack and queue to check if an item is symmetric
	Topic 4.2	Sorting & Searching (I) - Selection Sort, Bubble Sort, Insertion Sort
	Topic 4.3	Sorting & Searching (II) - Binary Search, Quick Sort, Hashing - Know how to use the algorithms - Comparison of algorithms - Some simple implementation of algorithms - An example on the use of hash function
	Topic 4.4	Binary Trees Heap and Heap Sorts Examples on - implementation of BST - traverse a BST - update a BST

Some sample examples on applications

Bitwise shift operators (applications)

1. To test if a positive integer is odd or even.

```
int m, n,

cin >> m;

n = m;
n = (n >> 1);           // n / 2;
n = (n << 1);           // n = n * 2;

if (m == n)
    cout << m << " is even" << endl;
else
    cout << m << " is odd" << endl;
```

2. To convert a positive integer to its binary equivalent.

For example: n = 193 to binary

```
193 / 2 = 96 R 1  <==> (n >> 0) & 1 = 1
96 / 2 = 48  R 0  <==> (n >> 1) & 1 = 0
48 / 2 = 24  R 0  <==> (n >> 2) & 1 = 0
24 / 2 = 12  R 0  <==> (n >> 3) & 1 = 0
12 / 2 = 6   R 0  <==> (n >> 4) & 1 = 0
6 / 2 = 3    R 0  <==> (n >> 5) & 1 = 0
3 / 2 = 1    R 1  <==> (n >> 6) & 1 = 1
1 / 2 = 0    R 1  <==> (n >> 7) & 1 = 1
```

Binary representation is
11000001

Display a positive integer n to its k bits binary representation

```
void displayBinary (int n, int k)
{
    for (int i = k - 1; i >= 0; i--)
```

```

        cout << ((n >> i) & 1);
    cout << endl;
}

```

Enhancement, what about octal & hexa representation?

```

// k stands for k octal symbols

void displayOctal (int n, int k)
{
    for (int i = k - 1; i >= 0; i--)
        cout << ((n >> i * 3) & 7);
    cout << endl;
}

```

3. Which of the followings swap two positive integers?

```

int x, y;
cin >> x >> y;

```

(a) $x = x \& y;$
 $y = x \& y;$
 $x = x \& y;$

(b) $x = x | y;$
 $y = x | y;$
 $x = x | y;$

(c) $x = x \wedge y;$
 $y = x \wedge y;$
 $x = x \wedge y;$

Answer: C

**// A structure can be passed to a function by
// value, by rerefence, and the return type can
// also be a structure**

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
struct RationalNo
{
    int numer;
    int denom;
    float value;
};
```

```
RationalNo getARationalNo ();
void updateARationalNo (RationalNo&);
void printARationalNo (RationalNo);
```

```
int main ()
{
    srand (time (NULL));

    for (int i = 1; i <= 10; i++)
    {
        RationalNo r = getARationalNo ();
        updateARationalNo (r);
        printARationalNo (r);
    }
}
```

```
RationalNo getARationalNo ()
{
    RationalNo r;

    r.numer = rand ();
    r.denom = rand () + 1;

    return r;
}
```

```
void updateARationalNo (RationalNo& r)
{
    r.value = 1.0 * r.numer / r.denom;
}
```

```
void printARationalNo (RationalNo r)
{
    cout << r.numer << " / "
        << r.denom << " = "
        << r.value
        << endl;
}
```

```
// A structure can be passed to a function by  
// value, by rerefence, and the return type can  
// also be a structure  
//  
// Text file processing
```

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

```
const int MAX = 80;
```

```
struct RationalNo  
{  
    char label [MAX];  
    int numer;  
    int denom;  
    float value;  
};
```

```
int infileToArray (fstream&, char [], RationalNo []);  
void displayArray (RationalNo [], int);
```

```
int main ()  
{  
    fstream afile;  
  
    RationalNo r [MAX];  
  
    int size = infileToArray (afile, "infile.txt", r);  
  
    displayArray (r, size);  
  
}
```

```
int infileToArray (fstream& afile, char fileName [], RationalNo r [])  
{  
    afile.open (fileName, ios::in);  
  
    if (!afile)
```

```

    {
        cout << fileName << " opened for reading failed" << endl;
        exit (-1);
    }

    cout << "Begin file to array" << endl;

    int i = 0;

    while (afile.getline (r [i].label, MAX))
    {
        afile >> r [i].numer >> r [i].denom;
        r [i].value = 1.0 * r [i].numer / r [i].denom;

        afile.clear ();
        afile.ignore (MAX, '\n');

        ++i;
    }

    afile.close ();
    cout << "File to array OK" << endl;

    return i;
}

void displayArray (RationalNo r [], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << r[i].label << ": ";

        cout << r [i].numer << " / "
        << r [i].denom << " = "
        << r [i].value
        << endl;
    }
}

```

Using one short integer word (16 bits), try to pack a binary symbol, an octal symbol and an hexa symbol in it, with some holes between the fields. Suggest a structure to denote this definition.

Draw a diagram and show to pack

- a binary symbol 1
- an octal symbol 6
- an hexa symbol A

according to your definition

```
struct Number
{
    short int bin :1;
    short int :4;
    short int oct :3;
    short int :4;
    short int hex :4;
};
```

```

          1 1 1 1 1 1 1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
1 h h h h 1 1 0 h h h h 1 0 1 0
```

```

// various conversions
// using bitwise operations
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

// no, no of symbols to be displayed
void displayBinary (int, int);
void displayOctal (int, int);
void displayHexa (int, int);

// general function - no to be converted
// no of bits to store a symbol
// the base
// the no of symbols to be displayed
void displayConversion (int, int, int, int);

int main ()
{
    int n;

    srand (time (NULL));

    n = rand ();

    cout << "Given no " << n << endl;

    cout << "Binary representation" << endl;
    displayBinary (n, 32);
    displayConversion (n, 1, 2, 32);

    cout << endl;
    cout << "Octal representation" << endl;
    displayOctal (n, 10);
    displayConversion (n, 3, 8, 10);

    cout << endl;
    cout << "hexa representation" << endl;
    displayHexa (n, 8);
}

```

```

        displayConversion (n, 4, 16, 8);
    }

void displayBinary (int n, int k)
{
    for (int i = k - 1; i >= 0; i--)
        cout << ((n >> i) & 1);
    cout << endl;
}

void displayOctal (int n, int k)
{
    for (int i = k - 1; i >= 0; i--)
        cout << ((n >> i * 3) & 7);
    cout << endl;
}

void displayHexa (int n, int k)
{
    char hexaDigit [] = "0123456789ABCDEF";

    for (int i = k - 1; i >= 0; i--)
        cout << hexaDigit [(n >> i * 4) & 15];
    cout << endl;
}

// general function - no to be converted
// no of bits to store a symbol
// the base
// the no of symbols to be displayed
void displayConversion (int n, int noOfBits, int base, int k)
{
    char charDigit [] = "0123456789ABCDEF";

    for (int i = k - 1; i >= 0; i--)
        cout << charDigit [(n >> i * (noOfBits)) & (base - 1)];
    cout << endl;
}

```

```
// Two shapes: circle and rectangle  
// Two types of boxes: circular box, rectangular box
```

```
#include <iostream>  
#include <fstream>  
#include <cstdlib>  
#include <ctime>  
using namespace std;
```

```
const double PI = 3.14159;
```

```
enum ShapeType {CShape, RShape};
```

```
const int MAX = 100;
```

```
struct Circle  
{  
    int radius;  
};
```

```
struct Rectangle  
{  
    int length;  
    int width;  
};
```

```
union Shape  
{  
    Circle c;  
    Rectangle r;  
};
```

```
struct Box  
{  
    ShapeType st;  
    Shape s;  
    int height;  
    double volume;  
};
```

```
void constructBoxFile (fstream&, char*);
```

```

int fileToArray (fstream&, char *, Box *);
void processArray (Box *, int);
void arrayToOutfile (Box*, int, ofstream&, char *);
void getLabel (char *, ShapeType);

int main ()
{
    fstream afile;

    srand (time (NULL));

    constructBoxFile (afile, "bFile.txt");

    cout << "-----" << endl;

    Box b [MAX];

    int size = fileToArray (afile, "bFile.txt", b);

    cout << "-----" << endl;

    processArray (b, size);

    cout << "-----" << endl;

    ofstream outfile;

    arrayToOutfile (b, size, outfile, "outfile.txt");

    cout << "-----" << endl;

    cout << "Last time compiled: " << __TIME__ << endl;
    cout << "Last date compiled: " << __DATE__ << endl;
    cout << "Current File: " << __FILE__ << endl;
    cout << "The line number: " << __LINE__;

}

void constructBoxFile (fstream& afile, char *fileName)
{

```

```

afile.open (fileName, ios::out);

if (!afile)
{
    cout << fileName << " opened for creation failed" << endl;
    exit (-1);
}

cout << "Begin the creation of " << fileName << endl;

int size = rand () % 51 + 50;

int type;

for (int i = 1; i <= size; i++)
{
    type = rand () % 2;

    afile << type << "\t";

    switch (type)
    {
        case 0: afile << rand () % 100 + 1;
                break;
        case 1: afile << rand () % 100 + 1 << "\t"
                << rand () % 100 + 1;
                break;
    }

    afile << "\t" << rand () % 100 + 1 << endl;
}

afile.close ();
cout << fileName << " successfully created" << endl;
}

int fileToArray (fstream& afile, char *fileName, Box *b)
{
    afile.open (fileName, ios::in);

```

```

    if (!afile)
    {
        cout << fileName << " opened for reading failed" << endl;
        exit (-1);
    }

    cout << "Begin from " << fileName << " to array" << endl;

    int i = 0;
    int type;

    while (afile >> type)
    {
        switch (type)
        {
            case 0: b [i].st = CShape;
                    afile >> b [i].s.c.radius;
                    break;
            case 1: b [i].st = RShape;
                    afile >> b [i].s.r.length;
                    afile >> b [i].s.r.width;
        }

        afile >> b [i].height;
        ++i;
    }

    afile.close ();
    cout << fileName << " to array OK" << endl;

    return i;
}

void processArray (Box *b, int size)
{
    cout << "Begin the process of box array" << endl;

    for (int i = 0; i < size; i++)
    {
        switch (b [i].st)

```

```

        {
            case CShape: b [i].volume = PI * b [i].s.c.radius
                                * b [i].s.c.radius
                                * b [i].height;
                                break;
            case RShape: b [i].volume = b [i].s.r.length *
                                b [i].s.r.width *
                                b [i].height;
        }
    }

    cout << "Array was processed" << endl;
}

void arrayToOutfile (Box *b, int size, ofstream& outfile, char *fileName)
{
    outfile.open (fileName);

    if (!outfile)
    {
        cout << "Array to outfile failed" << endl;
        exit (-1);
    }

    cout << "Begin from array to outfile" << endl;

    char label [MAX];

    for (int i = 0; i < size; i++)
    {
        getLabel (label, b [i].st);

        outfile << label;

        switch (b [i].st)
        {
            case CShape: outfile << "("
                                << b [i].s.c.radius << ", "
                                << b [i].height << ", "
                                << b [i].volume

```

```

        << ")"
        << endl;
        break;
    case RShape: outfile << "("
        << b [i].s.r.length << ", "
        << b [i].s.r.width << ", "
        << b [i].height << ", "
        << b [i].volume
            << ")"
            << endl;
    }
}

outfile.close ();
cout << "Array to outfile OK" << endl;
}

void getLabel (char *label, ShapeType st)
{
    switch (st)
    {
        case CShape: strcpy (label, "CircularBox ");
            break;
        case RShape: strcpy (label, "RectagularBox ");
    }
}

```

// create a binary file of integers

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
void createBinaryFile (fstream&, char *);
void processBinaryFile (fstream&, char *);
```

```
int main ()
{
    fstream afile;

    srand (time (NULL));

    createBinaryFile (afile, "bFile.dat");
    processBinaryFile (afile, "bFile.dat");
}
```

```
void createBinaryFile (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::out | ios::binary);

    // testing as usual

    int n;

    int size = rand () % 10 + 20;

    for (int i = 1; i <= size; i++)
    {
        n = rand ();

        afile.write (reinterpret_cast <const char *>(&n), sizeof (n));
    }

    afile.close ();
}
```

```
void processBinaryFile (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::in | ios::binary);

    // test as usual

    int n;

    while (afile.read (reinterpret_cast <char *>(&n), sizeof (n)))
    {
        cout << n << endl;
    }

    afile.close ();
}
```

// create a binary file of real fumbbers

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
void createBinaryFile (fstream&, char *);
void processBinaryFile (fstream&, char *);
```

```
int main ()
{
    fstream afile;

    srand (time (NULL));

    createBinaryFile (afile, "bFile.dat");
    processBinaryFile (afile, "bFile.dat");
}
```

```
void createBinaryFile (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::out | ios::binary);

    // testing as usual

    float n;

    int size = rand () % 10 + 20;

    for (int i = 1; i <= size; i++)
    {
        n = 1.0 * rand () / (rand () + 1);

        afile.write (reinterpret_cast <const char *>(&n), sizeof (n));
    }

    afile.close ();
}
```

```
void processBinaryFile (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::in | ios::binary);

    // test as usual

    float n;

    while (afile.read (reinterpret_cast <char *>(&n), sizeof (n)))
    {
        cout << n << endl;
    }

    afile.close ();
}
```

// create a binary file of rectangles

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
struct Rectangle
{
    int length;
    int width;
};
```

```
void createBinaryFile (fstream&, char *);
void processBinaryFile (fstream&, char *);
```

```
int main ()
{
    fstream afile;

    srand (time (NULL));

    createBinaryFile (afile, "bFile.dat");
    processBinaryFile (afile, "bFile.dat");
}
```

```
void createBinaryFile (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::out | ios::binary);

    // testing as usual

    Rectangle n;

    int size = rand () % 10 + 20;

    for (int i = 1; i <= size; i++)
    {
        n.length = rand () + 1;
```

```

        n.width = rand () + 1;

        afile.write (reinterpret_cast <const char *>(&n), sizeof (n));
    }

    afile.close ();
}

void processBinaryFile (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::in | ios::binary);

    // test as usual

    Rectangle n;

    while (afile.read (reinterpret_cast <char *>(&n), sizeof (n)))
    {
        cout << "R (" << n.length << ", "
                << n.width << ")"
                << endl;
    }

    afile.close ();
}

```

// A complete binary file processing

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

struct Rectangle
{
    int length;
    int width;
    int area;
    int perimeter;
};

void createBinary (fstream&, char *);
void binaryToText (fstream&, char *, ofstream&, char *);

int getNoOfRecords (fstream&, char*);
Rectangle getARecord (fstream&, char *, int);
void appendARecord (fstream&, char *, Rectangle);
void updateARecord (fstream&, char *, int);
void updateWholeFile (fstream&, char*);

int main ()
{
    fstream afile;

    srand (time (NULL));

    createBinary (afile, "bFile.dat");

    cout << "-----" << endl;

    cout << "There are "
        << getNoOfRecords (afile, "bFile.dat")
        << " records in " << " bFile.dat"
        << endl;

    cout << "-----" << endl;
```

```

int k;

cout << "Wish to see record: ";
cin >> k;

Rectangle r = getARecord (afile, "bFile.dat", k);

cout << "The rectangle is "
      << "R (" << r.length << ", " << r.width
      << ")"
      << endl;

cout << "-----" << endl;

cout << "Update record " << k << endl;

updateARecord (afile, "bFile.dat", k);

cout << "Record " << k << " updated" << endl;

cout << "-----" << endl;

r.length = 888;
r.width = 999;

appendARecord (afile, "bFile.dat", r);

cout << "-----" << endl;

updateWholeFile (afile, "bFile.dat");

cout << "-----" << endl;

ofstream outfile;
binaryToText (afile, "bFile.dat", outfile, "bFile.txt");
}

```

```

void createBinary (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::out | ios::binary);

    if (!afile)
    {
        cout << fileName << " opened for creation failed" << endl;
        exit (-1);
    }

    cout << "Begin the creation of binary file " << fileName << endl;

    Rectangle r;

    int size = rand () % 50 + 50;

    for (int i = 1; i <= size; i++)
    {
        r.length = rand () % 100 + 1;
        r.width = rand () % 100 + 1;

        afile.write (reinterpret_cast <const char *>(&r), sizeof (r));
    }

    afile.close ();
    cout << "Binary file " << fileName
        << " successfully created"
        << endl;
}

```

```

void binaryToText (fstream& afile, char *fileName1,
                  ofstream& outfile, char *fileName2)
{
    afile.open (fileName1, ios::in | ios::binary);

    if (!afile)
    {
        cout << "Binary file opened for reading failed" << endl;
        exit (-1);
    }
}

```

```

    }

    cout << "Binary file " << fileName1
          << " opened for reading OK"
          << endl;

    outfile.open (fileName2);

    if (!outfile)
    {
        cout << "Text file opened for creation failed" << endl;
        exit (-1);
    }

    cout << "Begin the creation of text file" << endl;

    Rectangle r;

    int i = 0;

    while (infile.read (reinterpret_cast <char *>(&r), sizeof (r)))
    {
        outfile << ++i << "\t"
                << r.length << "\t"
                << r.width << "\t"
                << r.area << "\t"
                << r.perimeter
                << endl;
    }

    infile.close ();
    cout << "Binary file " << fileName1
          << " closed for reading"
          << endl;

    outfile.close ();
    cout << "Binary to text file OK" << endl;
}

```

```

int getNoOfRecords (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::in | ios::binary);

    if (!afile)
    {
        afile.close ();
        return 0;
    }

    afile.seekg (0, ios::end);
    int totalNoBytes = afile.tellg ();
    int noOfRecords = totalNoBytes / sizeof (Rectangle);

    afile.close ();
    return noOfRecords;
}

Rectangle getARecord (fstream& afile, char *fileName, int k)
{
    afile.open (fileName, ios::in | ios::binary);

    // perform testing

    afile.seekg ((k - 1) * sizeof (Rectangle), ios::beg);

    Rectangle r;

    afile.read (reinterpret_cast <char *>(&r), sizeof (r));

    afile.close ();

    return r;
}

```

```

void appendARecord (fstream& afile, char *fileName, Rectangle r)
{
    afile.open (fileName, ios::out | ios::app | ios::binary);

    if (!afile)
    {
        cout << "File opened for appending failed" << endl;
        exit (-1);
    }

    cout << fileName << " opened for appending" << endl;

    afile.write (reinterpret_cast <const char *>(&r), sizeof (r));

    afile.close ();

    cout << "Record appended and file closed for appending" << endl;
}

```

```

void updateARecord (fstream& afile, char *fileName, int k)
{
    afile.open (fileName, ios::in | ios::out | ios::binary);

    Rectangle r;

    afile.seekg ((k - 1) * sizeof (Rectangle), ios::beg);
    afile.read (reinterpret_cast <char *>(&r), sizeof (r));

    r.area = r.length * r.width;
    r.perimeter = 2 * (r.length + r.width);

    afile.seekp ((k - 1) * sizeof (Rectangle), ios::beg);
    afile.write (reinterpret_cast <const char *>(&r), sizeof (r));

    afile.close ();
}

```

```

void updateWholeFile (fstream& afile, char *fileName)
{
    cout << "Update of whole file" << endl;
}

```

```
int totalRecords = getNoOfRecords (afile,fileName);

for (int k = 1; k <= totalRecords; k++)
    updateARecord (afile, fileName, k);

cout << "Whole file updated" << endl;
}
```

// File sort – random access file

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

```
const int MAX = 80;
```

```
struct Student
```

```
{
    int seatNo;
    char name [MAX];
    // can add in more info
};
```

```
void createRAF (fstream&, char *, ifstream&, char *);
```

```
void printRAF (fstream&, char *);
```

```
int main ()
```

```
{
    fstream afile;
    ifstream infile;

    createRAF (afile, "exam.dat", infile, "exam.txt");
    printRAF (afile, "exam.dat");
}
```

```
void createRAF (fstream& afile, char *fileName1,  
                ifstream& infile, char *fileName2)
```

```
{
    afile.open (fileName1, ios::out | ios::binary);
    infile.open (fileName2);
```

```
    Student s;
    char space;
```

```
    while (infile >> s.seatNo)
    {
        infile.get (space);
        infile.getline (s.name, MAX);
    }
```

```

        afile.seekp ((s.seatNo - 1) * sizeof (Student), ios::beg);
        afile.write (reinterpret_cast <const char *>(&s), sizeof (s));
    }

    afile.close ();
    infile.close ();
}

void printRAF (fstream& afile, char *fileName)
{
    afile.open (fileName, ios::in | ios::binary);

    Student s;

    while (afile.read (reinterpret_cast <char *>(&s), sizeof (s)))
    {
        // if (s.seatNo != 0)
        {
            cout << s.seatNo << "\t"
                 << s.name
                 << endl;
        }
    }

    afile.close ();
}

```



```

infile - Notepad
File Edit Format View Help
34 Tan Ah Kao
3 Lim Ah Beng
22 Lily nHo
18 Nancy Tan
5 Mohamed
2 Ali Abang
14 Tan Ah Kao 1
33 Lim Ah Beng 2
12 Lily nHo 1
28 Nancy Tan 3
15 Mohamedm 5
12 Ali Abang 6

```



```

2 Ali Abang
3 Lim Ah Beng
5 Mohamed
12 Ali Abang 6
14 Tan Ah Kao 1
15 Mohamedm 5
18 Nancy Tan
22 Lily nHo
28 Nancy Tan 3
33 Lim Ah Beng 2
34 Tan Ah Kao

```

Important to note - Binary file processing

Given

```
fstream afile,  
char fileName [MAX];
```

1. No array to be used for processing

2. Open binary file for reading

```
afile.open (fileName, ios::in | ios::binary);  
==> use the read function
```

3. Open binary file for writing

```
afile.open (fileName, ios::out | ios::binary);  
==> use the write function
```

4. Open binary file for appending

```
afile.open (fileName, ios::out | ios::app | ios::binary);  
==> use the write function
```

5. Open binary file for updating

```
afile.open (fileName, ios::in | ios::out | ios::binary);  
==> use the read function  
    use the seekg function  
    use the tellg function
```

```
    use the seekp function  
    use the write function
```

6. Always remember to close the file when coming out.

Make file utility

lab0 has three files

- lab0.h header file
- lab0.cpp file and we have included lab0.h in it
- mainLab0.cpp and we have included lab0.h in it

The following shows how to create a make file called Lab0.make, upon execution, lab0.out is generated

```
# The file name is called lab0.make
# Upon execution, lab0.out is generated
lab0.out: lab0.o mainLab0.o
    g++ -o lab0.out lab0.o mainLab0.o
lab0.o: lab0.cpp lab0.h
    g++ -c lab0.cpp
mainLab0.o: mainLab0.cpp lab0.h
    g++ -c mainLab0.cpp
clean:  rm *.o lab0.out

# The above is the content of lab0.make# Some remarks:
# To generate the lab0.out, type the following command
# make -f lab0.make
# To clean the object files and lab0.out,
# type the following command
# make -f lab0.make clean
```

```
# Three files, exam.cpp (included exam.h)
# supp.cpp (included supp.h) and mainExam.cpp (included
# exam.h and supp.h). Design a makefile called
# exam.make, upon execution, an executable called
# exam.out is generated
exam.out: exam.o supp.o mainExam.o
    g++ -o exam.out exam.o supp.o mainExam.o
exam.o: exam.cpp exam.h
    g++ -c exam.cpp
supp.o: supp.cpp supp.h
    g++ -c supp.cpp
mainExam.o: mainExam.cpp supp.h exam.h
    g++ -c mainExam.cpp
clean:
    rm *.o, exam.out
# To generate exam.out, type the following at command line
#    make -f exam.make
# To clear old object file and exam.out
#    make -f exam.make clean
```

// address of a variable

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

```
int main ()
```

```
{
```

```
    int *ptr = NULL;
```

```
    // Compile OK
```

```
    // runtime error
```

```
    // NULL point cannot be dereferenced
```

```
    // to get the value
```

```
    // cout << *ptr << endl;
```

```
    int n = 100;
```

```
    cout << "The address of n = "
         << &n
         << endl;
```

```
    ptr = &n;
```

```
    cout << "ptr = " << ptr << endl;
```

```
    cout << "n = " << n << endl;
```

```
    cout << "Dereferencing the pointer ptr = "
         << *ptr
         << endl;
```

```
    *ptr = 200;
```

```
    cout << "n = " << n << endl;
```

```
}
```

// array versus pointers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
const int MAX = 10;
```

```
void constructArray_1 (int [], int);
void constructArray_2 (int *, int);
void constructArray_3 (int *, int);
```

```
void printArray (int *, int);
```

```
int main ()
{
    int a [MAX];

    srand (time (NULL));

    constructArray_1 (a, MAX);
    printArray (a, MAX);

    constructArray_2 (a, MAX);
    printArray (a, MAX);

    constructArray_3 (a, MAX);
    printArray (a, MAX);
}

void constructArray_1 (int a [], int size)
{
    for (int i = 0; i < size; i++)
        a [i] = rand () % 100;
}

void constructArray_2 (int *a, int size)
{
    int *p = &a [0];
```

```

        for (int i = 0; i < size; i++)
        {
            *p = rand () % 100;
            ++p;
        }
    }

void printArray (int *a, int size)
{
    for (int i = 0; i < size; i++)
        cout << *(a + i) << "\t";
    cout << endl;
}

void constructArray_3 (int *a, int size)
{
    int *p = &a [0];
    int *q = &a [size - 1];

    while (q >= p)
    {
        *q = rand () % 100;
        q--;
    }
}

```

// Structure pointers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

struct Rectangle
{
    int length;
    int width;
    int area;
};

int main ()
{
    Rectangle r = {12, 34};
    Rectangle* rPtr = &r;

    (*rPtr).area = (*rPtr).length * (*rPtr).width;

    cout << "R ("
        << (*rPtr).length << ", "
        << (*rPtr).width << ", "
        << (*rPtr).area << ")"
        << endl;
}
```

// Structure pointers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

struct Rectangle

```
{
    int length;
    int width;
    int area;
};
```

```
int main ()
```

```
{
    Rectangle r = {12, 34};
    Rectangle* rPtr = &r;

    rPtr -> area = (rPtr -> length) * (rPtr -> width);

    cout << "R ("
        << rPtr -> length << ", "
        << rPtr -> width << ", "
        << rPtr -> area << ")"
        << endl;
}
```

// array of Structure pointers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
const int MAX = 10;
```

```
struct Rectangle
{
    int length;
    int width;
};
```

```
void constructRectangleArray (Rectangle*, int);
void printArray (Rectangle*, int);
```

```
int main ()
{
    Rectangle r [MAX];

    srand (time (NULL));

    constructRectangleArray (r, MAX);
    printArray (r, MAX);

}
```

```
void constructRectangleArray (Rectangle* r, int size)
{
    Rectangle *p = &r [0];

    for (int i = 0; i < size; i++)
    {
        p -> length = rand () % 100 + 1;
        p -> width = rand () % 100 + 1;

        ++p;
    }
}
```

```

}

void printArray (Rectangle *r, int size)
{
    Rectangle *p = &r [0];

    for (int i = 0; i < size; i++)
    {
        cout << "R ("
            << p -> length << ", "
            << p -> width << ")"
            << endl;

        ++p;
    }
}

```

// Using array via pointers, how to check if an integer is symmetric

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

```
const int MAX = 20;
```

```
bool isSymmetric (int);
int intToArray (int *, int);
```

```
int main ()
{
    int n;

    cout << "Enter a positive integer: ";
    cin >> n;

    if (isSymmetric (n))
        cout << n << " is symmetric" << endl;
    else
        cout << n << " is not symmetric" << endl;
}
```

```
bool isSymmetric (int n)
{
    int a [MAX];

    int size = intToArray (a, n);

    int *left = &a [0];
    int *right = &a [size - 1];

    while (left < right)
    {
        if (*left != *right)
            return false;

        ++left;
        --right;
    }
```

```

        return true;
    }

int intToArray (int *a, int n)
{
    int *p = &a [0];

    int i = 0;

    while (n > 0)
    {
        *p = n % 10;

        ++p;
        ++i;

        n /= 10;
    }

    return i;
}

```

// character pointer versus c-string

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

```
const int MAX = 20;
```

```
void change1 (char *);
void change2 (char *);
void change3 (char *&);
```

```
int main ()
```

```
{
    char cstring [MAX] = "abc";
    char *charPtr = "abc";

    change1 (cstring);
    change1 (charPtr);
    cout << cstring << endl; // abc
    cout << charPtr << endl; // abc

    cout << "----" << endl;

    change2 (cstring);
    cout << cstring << endl; // xyz
```

```
// compile OK, runtime failed
// because no actual storage
// to store the string value
// other than the 4 bytes memory
// for address
// change2 (charPtr);
```

```
cout << "----" << endl;
```

```
change3 (charPtr);
cout << charPtr << endl; // xyz
```

```
// Not able to compile with the following statement
// as array is already passed by reference
// change3 (cstring);
```

```
}
```

```
void change1 (char *str)
{
    str = "xyz";
}
```

```
void change2 (char *str)
{
    strcpy (str, "xyz");
}
```

```
void change3 (char *&str)
{
    str = "xyz";
}
```

// String tokenizer function strtok

```
#include <iostream>
#include <cstring>
using namespace std;

const int MAX = 80;

int main ()
{
    char str [MAX];

    cout << "Enter a string: ";
    cin.getline (str, MAX);

    char *p = strtok (str, ".,? !");

    while (p != NULL)
    {
        cout << p << endl;

        p = strtok (NULL, ".,? !");
    }
}
```

```
// String tokenizer function strtok
// we store the tokens in an array of tokens
```

```
#include <iostream>
#include <cstring>
using namespace std;

const int MAX = 80;
const int NUMTOKENS = 20;

int main ()
{
    char str [MAX];
    char token [NUMTOKENS][MAX];

    cout << "Enter a string: ";
    cin.getline (str, MAX);

    char *p = strtok (str, ".,? !");

    int i = 0;

    while (p != NULL)
    {
        strcpy (token [i], p);
        p = strtok (NULL, ".,? !");

        ++i;
    }

    for (int k = 0; k < i; k++)
        cout << token [k] << endl;
}
```

```
// String tokenizer function strtok  
// we store the tokens in an array of tokens  
// and each token is a char*
```

```
#include <iostream>  
#include <cstring>  
using namespace std;
```

```
const int MAX = 80;  
const int NUMTOKENS = 20;
```

```
typedef char* token;
```

```
int main ()  
{  
    char str [MAX];  
    token t [NUMTOKENS];  
  
    cout << "Enter a string: ";  
    cin.getline (str, MAX);  
  
    char *p = strtok (str, ".,? !");  
  
    int i = 0;  
  
    while (p != NULL)  
    {  
        t [i] = p; // Note we use = sign  
        p = strtok (NULL, ".,? !");  
  
        ++i;  
    }  
  
    for (int k = 0; k < i; k++)  
        cout << t [k] << endl;  
}
```

// String tokenizer function strtok
// an array of words

```
#include <iostream>
#include <cstring>
using namespace std;
```

```
const int MAX = 80;
const int MAXWORDS = 20;
```

```
typedef char* Word;
```

```
int main ()
{
```

```
    char str [MAX];
    Word *w;
```

```
    // To initialize w
    w = new char * [MAXWORDS];
    for (int i = 0; i < MAXWORDS; i++)
        w [i] = new char [MAX];
```

```
    cout << "Enter a string: ";
    cin.getline (str, MAX);
```

```
    char *p = strtok (str, ".,? !");
```

```
    int i = 0;
```

```
    Word *wPtr = &w [0];
```

```
    while (p != NULL)
```

```
    {
        strcpy (*wPtr, p); // Note we use strcpy here
        p = strtok (NULL, ".,? !");
```

```
        ++i;
        ++wPtr;
    }
```

// We can do pointer airthmetic on word array

```
wPtr = &w [0];  
for (int k = 0; k < i; k++)  
{  
    cout << *wPtr << endl;  
    ++wPtr;  
}
```

// Garbage collection

```
for (int k = 0; k < MAXWORDS; k++)  
    delete [] w [k];  
delete [] w;
```

```
}
```

```

// Explore the use of structure pointers and
// its application to array of structures
//
// Question:
// Two words are anagrams if and only all the characters
// appeared in the 1st word also appeared in the 2nd word
// and vice versa, including the same number of duplications
// For example: abbba babab are anagrams
//               ababa babab are not anagrams
//
// Idea: How to check 2 words are anagrams?
//
//     abbba           ababa
//     *****        *****
//     babab           babab
//     *****        *****

```

```

#include <iostream>
using namespace std;

const int MAX = 20;

struct CharInfo
{
    char ch;
    char marker;
};

bool areAnagrams (char *, char *);
void transfer (CharInfo *, char *);
void printCI (CharInfo *, int);
bool charInWord (CharInfo *, char, int);

int main ()
{
    char *word1;
    char *word2;

    for (int i = 1; i <= 5; i++)
    {
        word1 = new char [MAX];

```

```

word2 = new char [MAX];

cout << "Enter two words: ";
cin >> word1 >> word2;

if (areAnagrams (word1, word2))
    cout << word1 << " and " << word2
        << " are anagrams" << endl;
else
    cout << word1 << " and " << word2
        << " are not anagrams" << endl;

// Garbage collection
delete [] word1;
delete [] word2;

cout << "-----" << endl;
}
}

```

```

bool areAnagrams (char *word1, char *word2)
{
    int len1 = strlen (word1);
    int len2 = strlen (word2);

    if (len1 != len2)
        return false;

    CharInfo* ci_1 = new CharInfo [len1];
    CharInfo* ci_2 = new CharInfo [len2];

    transfer (ci_1, word1);
    transfer (ci_2, word2);

    CharInfo *p = &ci_1 [0];

    for (int i = 0; i < len1; i++)
    {

```

```

        if (!charInWord (ci_2, p -> ch, len2))
        {
            cout << "Reasons" << endl;
            printCI (ci_1, len1);
            printCI (ci_2, len2);

            delete [] ci_1;
            delete [] ci_2;

            return false;
        }

        p-> marker = '*';
        ++p;
    }

    cout << "Reasons" << endl;
    printCI (ci_1, len1);
    printCI (ci_2, len2);

    delete [] ci_1;
    delete [] ci_2;

    return true;
}

void transfer (CharInfo *ci, char *word)
{
    CharInfo *p = &ci [0];
    char *w = &word [0];

    while (*w != '\0')
    {
        p -> ch = *w;
        p -> marker = '#';

        ++w;
        ++p;
    }
}

```

```

void printCI (CharInfo *ci, int len)
{
    CharInfo *p = &ci [0];

    cout << "\t";
    for (int i = 0; i < len; i++)
    {
        cout << p -> ch;
        ++p;
    }

    cout << endl;

    cout << "\t";

    p = &ci [0];
    for (int i = 0; i < len; i++)
    {
        cout << p -> marker;
        ++p;
    }

    cout << endl;
}

bool charInWord (CharInfo *ci, char aChar, int len)
{
    CharInfo *p = &ci [0];

    for (int i = 0; i < len; i++)
    {
        if ((p -> ch == aChar) && (p -> marker == '#'))
        {
            p -> marker = '*';
            return true;
        }

        ++p;
    }
}

```

```
        return false;  
    }
```

```

// Explore the use of structure pointers and
// its application to array of structures
//
// Question:
// Two words are anagrams if and only all the characters
// appeared in the 1st word also appeared in the 2nd word
// and vice versa, including the same number of duplications
// For example: abbba babab are anagrams
//               ababa babab are not anagrams
//
// Idea: How to check 2 words are anagrams?
// Non-case sensitive
//
//      abBBa          ababa
//      *****       *****
//      babab          babab
//      *****       *****
#include <iostream>
#include <cctype>
using namespace std;

const int MAX = 20;

struct CharInfo
{
    char ch;
    char marker;
};

bool areAnagrams (char *, char *);
void transfer (CharInfo *, char *);
void printCI (CharInfo *, int);
bool charInWord (CharInfo *, char, int);

int main ()
{
    char *word1;
    char *word2;

    for (int i = 1; i <= 5; i++)

```

```

{
    word1 = new char [MAX];
    word2 = new char [MAX];

    cout << "Enter two words: ";
    cin >> word1 >> word2;

    if (areAnagrams (word1, word2))
        cout << word1 << " and " << word2
            << " are anagrams" << endl;
    else
        cout << word1 << " and " << word2
            << " are not anagrams" << endl;

    // Garbage collection
    delete [] word1;
    delete [] word2;

    cout << "-----" << endl;
}
}

bool areAnagrams (char *word1, char *word2)
{
    int len1 = strlen (word1);
    int len2 = strlen (word2);

    if (len1 != len2)
        return false;

    CharInfo* ci_1 = new CharInfo [len1];
    CharInfo* ci_2 = new CharInfo [len2];

    transfer (ci_1, word1);
    transfer (ci_2, word2);

    CharInfo *p = &ci_1 [0];

```

```

for (int i = 0; i < len1; i++)
{
    if (!charInWord (ci_2, p -> ch, len2))
    {
        cout << "Reasons" << endl;
        printCI (ci_1, len1);
        printCI (ci_2, len2);

        delete [] ci_1;
        delete [] ci_2;

        return false;
    }

    p-> marker = '*';
    ++p;
}

cout << "Reasons" << endl;
printCI (ci_1, len1);
printCI (ci_2, len2);

delete [] ci_1;
delete [] ci_2;

return true;
}

void transfer (CharInfo *ci, char *word)
{
    CharInfo *p = &ci [0];
    char *w = &word [0];

    while (*w != '\0')
    {
        p -> ch = *w;
        p -> marker = '#';

        ++w;
        ++p;
    }
}

```

```

    }
}

void printCI (CharInfo *ci, int len)
{
    CharInfo *p = &ci [0];

    cout << "\t";
    for (int i = 0; i < len; i++)
    {
        cout << p -> ch;
        ++p;
    }

    cout << endl;

    cout << "\t";

    p = &ci [0];
    for (int i = 0; i < len; i++)
    {
        cout << p -> marker;
        ++p;
    }

    cout << endl;
}

```

```

bool charInWord (CharInfo *ci, char aChar, int len)
{
    CharInfo *p = &ci [0];

    for (int i = 0; i < len; i++)
    {
        if ((tolower (p -> ch) == tolower (aChar))
            && (p -> marker == '#'))
        {
            p -> marker = '*';
            return true;
        }
    }
}

```

```
        ++p;  
    }  
    return false;  
}
```

```

// an array of void*
//
// implicitly it can be an array of characters or an array of integers
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

typedef void* VoidPtr;

int main ()
{
    VoidPtr *vpArray;
    int size;

    srand (time (NULL));
    size = rand () % 10 + 10;

    vpArray = new VoidPtr [size];

    int *temp;

    for (int i = 0; i < size; i++)
    {
        temp = new int;
        *temp = rand () % 26 + 65;

        vpArray [i] = temp;
    }

    for (int i = 0; i < size; i++)
    {
        int n = *(static_cast <int *>(vpArray [i]));
        char ch = *(static_cast <char *>(vpArray [i]));

        cout << n << " - " << ch << endl;
    }
}

```

// An array of void*
// Design a generic function to find the largest
// and the smallest elements stored in the array

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

typedef void* VoidPtr;

void findMinMax (VoidPtr*, int, VoidPtr&, VoidPtr&);
int compareVP (VoidPtr, VoidPtr);
int getVP (VoidPtr);

int main ()
{
    VoidPtr *vpArray;
    int size;

    srand (time (NULL));
    size = rand () % 10 + 10;

    vpArray = new VoidPtr [size];

    int *temp;

    for (int i = 0; i < size; i++)
    {
        temp = new int;
        *temp = rand ();

        vpArray [i] = temp;
    }

    for (int i = 0; i < size; i++)
    {
        cout << getVP (vpArray [i]) << endl;
    }
}
```

```

    cout << endl;

    VoidPtr min, max;
    findMinMax (vpArray, size, min, max);

    cout << "Minimum = " << getVP (min) << endl;
    cout << "Maximum = " << getVP (max) << endl;

}

void findMinMax (VoidPtr *vpArray, int size,
                VoidPtr& min, VoidPtr& max)
{
    min = vpArray [0];
    max = vpArray [0];

    for (int i = 1; i < size; i++)
    {
        if (compareVP (min, vpArray [i]) > 0)
            min = vpArray [i];

        if (compareVP (max, vpArray [i]) < 0)
            max = vpArray [i];
    }
}

int compareVP (VoidPtr vp1, VoidPtr vp2)
{
    int item1 = getVP (vp1);
    int item2 = getVP (vp2);

    if (item1 > item2)
        return 1;
    else if (item1 == item2)
        return 0;
    else
        return -1;
}

```

```
int getVP (VoidPtr vp)
{
    return *(static_cast <int *>(vp));
}
```

// An array of void*

// Design a generic function to test if a word is symmetric

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
```

```
const int MAX = 20;
```

```
typedef void* VoidPtr;
```

```
char compareVP (VoidPtr, VoidPtr);
```

```
char getVP (VoidPtr);
```

```
bool isSymmetric (char *);
```

```
int wordToVPArray (char *, VoidPtr*);
```

```
int main ()
```

```
{
```

```
    char *word;
```

```
    word = new char [MAX];
```

```
    cout << "Enter a word: ";
```

```
    cin >> word;
```

```
    if (isSymmetric (word))
```

```
        cout << word << " is symmetric" << endl;
```

```
    else
```

```
        cout << word << " is not symmetric" << endl;
```

```
}
```

```
bool isSymmetric (char *word)
```

```
{
```

```
    VoidPtr *vpArray;
```

```
    vpArray = new VoidPtr [strlen (word)];
```

```
    int size = wordToVPArray (word, vpArray);
```

```

    int left = 0;
    int right = size - 1;

    while (left < right)
    {
        if (compareVP (vpArray [left], vpArray [right]) != 0)
            return false;

        ++left;
        --right;
    }

    return true;
}

```

```

char compareVP (VoidPtr vp1, VoidPtr vp2)
{
    char item1 = getVP (vp1);
    char item2 = getVP (vp2);

    if (item1 > item2)
        return 1;
    else if (item1 == item2)
        return 0;
    else
        return -1;
}

```

```

char getVP (VoidPtr vp)
{
    return *(static_cast <char *>(vp));
}

```

```

int wordToVPArray (char *word, VoidPtr *vpArray)
{
    char *p = &word [0];
    char *temp;

    int i = 0;

```

```
while (*p != '\0')
{
    temp = new char;
    *temp = *p;

    vpArray [i] = temp;
    ++i;
    ++p;
}

return i;
}
```

// A general swap function
// Indirectly swap two integers

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

typedef void* VoidPtr;

void generalSwap (VoidPtr&, VoidPtr&);

int main ()
{
    int *item1;
    int *item2;
    VoidPtr vp1, vp2;

    item1 = new int;
    item2 = new int;

    *item1 = 111;
    *item2 = 222;

    cout << "Before swap" << endl;
    cout << endl;

    cout << "Item 1 = "
         << *item1
         << endl;

    cout << "Item 2 = "
         << *item2
         << endl;

    vp1 = item1;
    vp2 = item2;
    generalSwap (vp1, vp2);

    cout << "\nAfter swap" << endl;
    cout << endl;
```

```

    cout << "Item 1 = "
          << *(static_cast<int*>(vp1))
          << endl;

    cout << "Item 2 = "
          << *(static_cast<int*>(vp2))
          << endl;

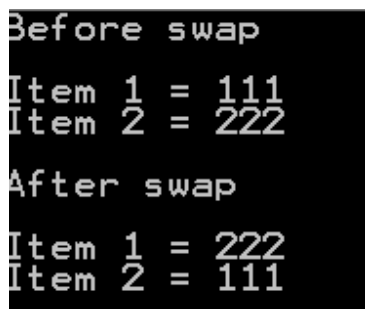
    cout << endl;

}

void generalSwap (VoidPtr& vp1, VoidPtr& vp2)
{
    VoidPtr temp;

    temp = vp1;
    vp1 = vp2;
    vp2 = temp;
}

```



```

Before swap
Item 1 = 111
Item 2 = 222

After swap
Item 1 = 222
Item 2 = 111

```

```

// A general swap function
// Indirectly swap two characters
#include <iostream>
using namespace std;

typedef void* VoidPtr;

void generalSwap (VoidPtr&, VoidPtr&);

int main ()
{
    char *item1;
    char *item2;
    VoidPtr vp1, vp2;

    item1 = new char;
    item2 = new char;

    *item1 = 'A';
    *item2 = 'B';

    cout << "Before swap" << endl;
    cout << endl;

    cout << "Item 1 = "
         << *item1
         << endl;

    cout << "Item 2 = "
         << *item2
         << endl;

    vp1 = item1;
    vp2 = item2;
    generalSwap (vp1, vp2);

    cout << "\nAfter swap" << endl;
    cout << endl;

    cout << "Item 1 = "

```

```

        << *(static_cast<char*>(vp1))
        << endl;

    cout << "Item 2 = "
        << *(static_cast<char*>(vp2))
        << endl;

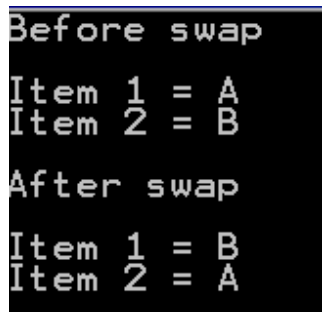
    cout << endl;

}

void generalSwap (VoidPtr& vp1, VoidPtr& vp2)
{
    VoidPtr temp;

    temp = vp1;
    vp1 = vp2;
    vp2 = temp;
}

```



```

Before swap
Item 1 = A
Item 2 = B
After swap
Item 1 = B
Item 2 = A

```

```

// void ***
// - Implicitly it is a 2D array of integers
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

typedef void* VoidPtr;

int main ()
{
    VoidPtr **vp2D;

    srand (time (NULL));

    int row = rand () % 6 + 5;
    int col = rand () % 6 + 5;

    vp2D = new VoidPtr* [row];
    for (int i = 0; i < row; i++)
        vp2D [i] = new VoidPtr [col];

    int *temp;

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            temp = new int;
            *temp = rand () % 100;

            vp2D [i][j] = temp;
        }
    }

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cout << *(static_cast <char *> (vp2D [i][j])) << "\t";

```

```

    }
    cout << endl;
}
}

```

◀	\	V	=	A
😊	)	&	U	@
]	]	?	Z	7
(	#	Z	7	7
▲	S	W	]	S

// Transfer from array of C-string to array of void*

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
const int MAX = 80;
const int MAXWORDS = 5;
```

```
const char SomeWords [[MAX] = {"aaa", "bbb", "ccc", "ddd", "eee"};
```

```
typedef void* VoidPtr;
typedef char* Word;
```

// get a word from the wordArray

```
Word getAWord (Word *wordArray, int k);
```

// get a word indirectly

```
VoidPtr getAVP (VoidPtr *, int);
```

// From word array to vpArray

```
void wordArrayToVPArray (Word *wordArray, VoidPtr* vpArray,
int);
```

// Print the void* array

```
void printVPArray (VoidPtr *, int);
```

// get VP function

```
char * getVP (VoidPtr);
```

```
int main ()
{
```

```
    Word *wordArray;
```

```
    // Implicit wordArray [MAXWORDS][MAX]
```

```
    wordArray = new Word [MAXWORDS];
```

```
    for (int i = 0; i < MAXWORDS; i++)
```

```
        wordArray [i] = new char [MAX];
```

```

// Transfer the constant SomeWords array to wordArray
for (int i = 0; i < MAXWORDS; i++)
    strcpy (wordArray [i], SomeWords [i]);

// Some tests on getting of words
for (int i = 1; i <= 5; i++)
{
    int k = rand () % MAXWORDS;

    cout << "Word " << k << " is "
         << getAWord (wordArray, k)
         << endl;
}

// Next feature, on array of void*
VoidPtr *vpArray = new VoidPtr [MAXWORDS];
wordArrayToVPArray (wordArray, vpArray, MAXWORDS);

cout << endl;
printVPArray (vpArray, MAXWORDS);

// Some tests on getting of words indirectly
for (int i = 1; i <= 5; i++)
{
    int k = rand () % MAXWORDS;

    cout << "Word " << k << " is "
         << getVP (getAVP (vpArray, k))
         << endl;
}

}

Word getAWord (Word *wordArray, int k)
{
    Word *p = &wordArray [0];

    for (int i = 0; i < k; i++)
        ++p;
}

```

```

        return *p;
    }

VoidPtr getAVP (VoidPtr *vpArray, int k)
{
    VoidPtr *p = &vpArray [0];

    for (int i = 0; i < k; i++)
        ++p;

    return *p;
}

void wordArrayToVPArray (Word *wordArray, VoidPtr* vpArray, int size)
{
    Word *p = &wordArray [0];
    VoidPtr *q = &vpArray [0];

    for (int i = 0; i < size; i++)
    {
        *q = *p;
        ++p;
        ++q;
    }
}

void printVPArray (VoidPtr *vpArray, int size)
{
    VoidPtr *p = &vpArray [0];

    for (int i = 0; i < size; i++)
    {
        cout << getVP (*p) << " ";
        ++p;
    }

    cout << endl;
}

char * getVP (VoidPtr vp)

```

```
{  
    return static_cast<char*>(vp);  
}
```

```
Word 1 is bbb  
Word 2 is ccc  
Word 4 is eee  
Word 0 is aaa  
Word 4 is eee  
  
aaa bbb ccc ddd eee  
Word 4 is eee  
Word 3 is ddd  
Word 3 is ddd  
Word 2 is ccc  
Word 4 is eee
```

// A linked list of integers
// - a linked list is pointed by a head pointer

#include <iostream>
using namespace std;

struct Node;
typedef Node* NodePtr;

struct Node
{
 int data;
 NodePtr next;
};

void addHead (NodePtr&, int);
void printList (NodePtr);

void garbageCollection (NodePtr&);

int main ()
{
 NodePtr head = NULL;

 for (int i = 1; i <= 10; i++)
 addHead (head, i);

 printList (head);
 garbageCollection (head);
}

void addHead (NodePtr& head, int item)
{
 NodePtr temp = new Node;
 temp -> data = item;
 temp -> next = head;

 head = temp;

}

```

void printList (NodePtr head)
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << temp -> data << "\t";
        temp = temp -> next;
    }

    cout << endl;
}

void garbageCollection (NodePtr& head)
{
    NodePtr temp;

    cout << endl;
    cout << "Garbage collection" << endl;

    while (head != NULL)
    {
        temp = head;

        head = head -> next;

        cout << "Node with value "
             << temp -> data;

        delete temp;
        cout << " deleted" << endl;
    }
}

```

```

// A linked list of characters
// - a linked list is pointed by a head pointer
//
// No difference to to the previous linked list of integers
// except the data item is a character
#include <iostream>
using namespace std;

```

```

struct Node;
typedef Node* NodePtr;

```

```

struct Node
{
    char data;
    NodePtr next;
};

```

```

void addHead (NodePtr&, char);
void printList (NodePtr);

```

```

int main ()
{
    NodePtr head = NULL;

    for (char i = 'A'; i <= 'Z'; i++)
        addHead (head, i);

    printList (head);
}

```

```

void addHead (NodePtr& head, char item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;
}

```

```
void printList (NodePtr head)
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << temp -> data << " ";
        temp = temp -> next;
    }

    cout << endl;
}
```

```

// A linked list of Rectangles
// - a linked list is pointed by a head pointer
//
// No difference to the two previous linked lists except the data
// items are now some structures
#include <iostream>
using namespace std;

struct Node;
typedef Node* NodePtr;

struct Rectangle
{
    int length;
    int width;
};

struct Node
{
    Rectangle data;
    NodePtr next;
};

void addHead (NodePtr&, Rectangle);
void printList (NodePtr);

int main ()
{
    NodePtr head = NULL;

    Rectangle r;

    for (int i = 1; i <= 10; i++)
    {
        r.length = i;
        r.width = i;

        addHead (head, r);
    }
}

```

```

        printList (head);
    }

void addHead (NodePtr& head, Rectangle item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;

}

void printList (NodePtr head)
{
    NodePtr temp = head;

    Rectangle r;

    while (temp != NULL)
    {
        r = temp -> data;

        cout << "R (" << r.length
            << ", " << r.width << ")"
            << endl;

        temp = temp -> next;
    }

    cout << endl;

}

```

// A linked list of English words

```
#include <iostream>
#include <cstring>
using namespace std;

const int MAX = 80;

struct Node;
typedef Node* NodePtr;

struct Node
{
    char* data;
    NodePtr next;
};

void addHead (NodePtr&, char *);
void printList (NodePtr);

int main ()
{
    NodePtr head = NULL;
    char str [MAX];
    char *word;

    cout << "Enter a string: ";
    cin.getline (str, MAX);

    char *p = strtok (str, ",. ; ?");

    while (p != NULL)
    {
        word = new char [MAX];

        strcpy (word, p);

        addHead (head, word);

        p = strtok (NULL, ",. ; ?");
    }
}
```

```

        printList (head);
    }

void addHead (NodePtr& head, char* item)
{
    NodePtr temp = new Node;

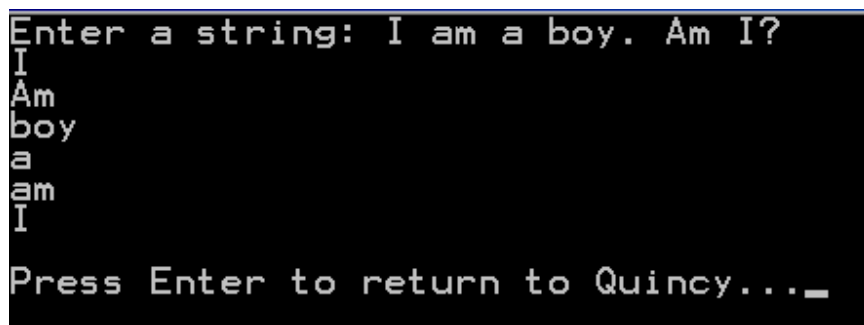
    temp -> data = new char [MAX];
    strcpy (temp -> data, item);

    temp -> next = head;
    head = temp;
}

void printList (NodePtr head)
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << temp -> data << endl;
        temp = temp -> next;
    }
}

```



```

Enter a string: I am a boy. Am I?
I
Am
boy
a
am
I
Press Enter to return to Quincy..._

```

```
// A linked list of integers  
// - a linked list is pointed by a head pointer  
//
```

```
// More functions added to the linked list
```

```
#include <iostream>  
#include <ctime>  
#include <cstdlib>  
using namespace std;
```

```
struct Node;  
typedef Node* NodePtr;
```

```
struct Node  
{  
    int data;  
    NodePtr next;  
};
```

```
void addHead (NodePtr&, int);  
void addTail (NodePtr&, int);
```

```
void printList (NodePtr);
```

```
int main ()  
{  
    NodePtr head = NULL;  
  
    srand (time (NULL));  
  
    int item;  
  
    for (int i = 1; i <= 10; i++)  
    {  
        item = rand ();  
  
        if (item % 2)  
            addHead (head, item);  
        else  
            addTail (head, item);  
    }  
}
```

```

        printList (head);
    }

void addHead (NodePtr& head, int item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;
}

void addTail (NodePtr& head, int item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;

    if (head == NULL)
        head = temp;
    else
    {
        NodePtr curr = head;

        while (curr -> next != NULL)
            curr = curr -> next;

        curr -> next = temp;
    }
}

void printList (NodePtr head)
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << temp -> data << endl;
    }
}

```

```
        temp = temp -> next;
    }

    cout << endl;

}

/
```

**// Application – using linear linked list, we can check
// if a positive integer is symmetric**

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

struct Node;
typedef Node* NodePtr;

struct Node
{
    int data;
    NodePtr next;
};

void addHead (NodePtr&, int);
void addTail (NodePtr&, int);

void printList (NodePtr);
bool isSymmetric (int);

int main ()
{
    int n;

    cout << "Enter an integer: ";
    cin >> n;

    if (isSymmetric (n))
        cout << n << " is symmetric" << endl;
    else
        cout << n << " is not symmetric" << endl;
}

void addHead (NodePtr& head, int item)
{
    NodePtr temp = new Node;
```

```

    temp -> data = item;
    temp -> next = head;

    head = temp;

}

void addTail (NodePtr& head, int item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;

    if (head == NULL)
        head = temp;
    else
    {
        NodePtr curr = head;

        while (curr -> next != NULL)
            curr = curr -> next;

        curr -> next = temp;
    }
}

void printList (NodePtr head)
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << temp -> data << endl;
        temp = temp -> next;
    }

    cout << endl;
}

```

```

bool isSymmetric (int n)
{
    NodePtr head1 = NULL;
    NodePtr head2 = NULL;

    while (n > 0)
    {
        addHead (head1, n % 10);
        addTail (head2, n % 10);

        n /= 10;
    }

    NodePtr temp1 = head1;
    NodePtr temp2 = head2;

    while (temp1 != NULL && temp2 != NULL)
    {
        if (temp1 -> data != temp2 -> data)
            return false;

        temp1 = temp1 -> next;
        temp2 = temp2 -> next;
    }

    return true;
}

```

```
// A linked list of english words  
//  
// access to the list and analyze each word  
// stored in the list, for example, length,  
// change all vowels to '*' ... etc
```

```
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
#include <cstring>  
using namespace std;
```

```
typedef void* VoidPtr;  
const int MAX = 80;
```

```
struct Node;  
typedef Node* NodePtr;
```

```
struct Node  
{  
    VoidPtr data;  
    NodePtr next;  
};
```

```
void addHead (NodePtr&, VoidPtr);  
void addTail (NodePtr&, VoidPtr);
```

```
char * getVP (VoidPtr);
```

```
void printList (NodePtr);  
void garbageCollection (NodePtr&);
```

```
// given a word, we wish to know  
// the length and the number of vowels  
void wordInfo (char *, int&, int&);
```

```
int main ()  
{  
    NodePtr head = NULL;
```

```

char str [MAX];
char * word;
VoidPtr item;

srand (time (NULL));

cout << "Enter a string: ";
cin.getline (str, MAX);

char *p = strtok (str, ",. ");

while (p != NULL)
{
    word = new char [MAX];
    strcpy (word, p);

    item = word;
    addTail (head, item);

    p = strtok (NULL, ",. ");
}

printList (head);

cout << "-----" << endl;

garbageCollection (head);
}

void addHead (NodePtr& head, VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;
}

```

```
void addTail (NodePtr& head, VoidPtr item)
```

```
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;

    if (head == NULL)
        head = temp;
    else
    {
        NodePtr curr = head;

        while (curr -> next != NULL)
            curr = curr -> next;

        curr -> next = temp;
    }
}
```

```
void printList (NodePtr head)
```

```
{
    NodePtr temp = head;

    char *word;

    int length, vowels;

    while (temp != NULL)
    {
        word = new char [MAX];
        strcpy (word, getVP (temp -> data));
        wordInfo (word, length, vowels);

        cout << word << " - " << length
              << " - " << vowels << endl;

        temp = temp -> next;
    }
}
```

```

void garbageCollection (NodePtr& head)
{
    NodePtr temp = head;

    while (head != NULL)
    {
        temp = head;

        head = head -> next;

        cout << "Node with value "
              << getVP (temp -> data);

        delete temp;
        cout << " was deleted" << endl;
    }
}

char * getVP (VoidPtr vp)
{
    return static_cast <char *>(vp);
}

void wordInfo (char *word, int& length, int& vowels)
{
    length = strlen (word);
    vowels = 0;

    char *p = &word [0];

    while (*p != '\0')
    {
        char temp = tolower (*p);

        if (temp == 'a' || temp == 'e' ||
            temp == 'o' || temp == 'u' || temp == 'i')
        {
            ++vowels;
            *p = '*';
        }
    }
}

```

```

        ++p;
    }
}

```

```

Enter a string: The life is hard in SIM. But I enjoy.
Th* - 3 - 1
l*f* - 4 - 2
*s - 2 - 1
n*rd - 4 - 1
*n - 2 - 1
S*M - 3 - 1
B*t - 3 - 1
* - 1 - 1
*nj*y - 5 - 2
-----
Node with value The was deleted
Node with value life was deleted
Node with value is was deleted
Node with value hard was deleted
Node with value in was deleted
Node with value SIM was deleted
Node with value But was deleted
Node with value I was deleted
Node with value enjoy was deleted

```

Revision Examples

What we have done ...

Topic 1

1. Bitwise manipulation

(a) to test if a positive integer is odd or even

(b) to convert a positive integer to its
binary / octal / hexa equivalent

```
void printConversion (int n, int base, int shift, int noOfBits)
{
    char symbol [] = "0123456789ABCDEF";

    for (int i = noOfBits - 1; i >= 0; i--)
        cout << symbol [((n >> shift * i) & (base - 1))];
    cout << endl;
}
```

```
printConversion (n, 2, 1, 24);
printConversion (n, 8, 3, 8);
printHexa (n, 16, 4, 6);
```

(c) To swap two positive integers

```
int m, n;
m = rand (); // 3
n = rand (); // 5
```

```
case 1:
m = m | n; //binary 011 | 101 = 111
n = m | n; //binary 111 | 101 = 111
m = m | n;
```

```
case 2:
m = m & n; //binary 011 & 101 = 001
n = m & n; //binary 001 & 101 = 001
```

$m = m \& n;$

case 3:

$m = m \wedge n;$ //binary $011 \wedge 101 = 110$

$n = m \wedge n;$ //binary $110 \wedge 101 = 011 = 3$

$m = m \wedge n;$ //binary $110 \wedge 011 = 101 = 5$

case 4:

None of the above

2. Enum / struct / union

- representation using bits and holes
- : lecture note examples - calendar date
- : class example, symbols

3. Binary file processing

- (a) How to access?
 - : read, tellg, seekg
- (b) How to create?
 - : write, tellp, seekp
- (c) Important features:
 - : How to append?
 - : How to update?
 - : count the number records
 - : file sorting!!!!

Some examples on binary

Given the following structure definition;

```
enum Gender { Male, Female};
```

```
struct Student
{
    Gender g;
    char name [MAX];
    int mark;
    // etc
};
```

Example 1: Your lecturer stored all students' info in a binary file. Design a function to get the no of students in this binary file. The function has the following prototype:

```
int getNoOfStudents (const char fileName [])
{
    fstream afile;
    afile.open(fileName, ios::in | ios::binary);

    afile.seekg(0, ios::end);
    int numRecords = afile.tellg() / sizeof(Student);

    afile.close();

    return numRecords;
}
```

Example 2. One of the students sitting at kth records in the binary file requesting the lecturer to add 1 mark to his submission. Design a function to entertain this student. The function must have the following prototype:

```
void updateKStudent (const char fileName [], int k)
{
    Student info;
    fstream afile;
    afile.open(fileName, ios::in|ios::out|ios::binary);

    afile.seekg((k-1)*sizeof(Student), ios::beg);
    afile.read(reinterpret_cast <char*> (&info), sizeof (Student));
    info.marks++;
    afile.seekp((k-1)*sizeof(Student), ios::beg);
    afile.write(reinterpret_cast <const char*> (&info), sizeof (Student));

    afile.close();
}
```

Example 3: Your lecturer decided to add 10 marks to all the male

students and 20 marks to all the female students. Design a function to help your lecturer to update all records. The function must have the following prototype:

```
void updateFile (const char fileName [])
{
    fstream afile;
    Student s;

    afile.open (fileName, ios::in | ios:: out | ios:: binary);

    int k = 1;

    while (afile.read (reinterpret_cast <char *> (&s), sizeof (s)));
    {
        if (s.gender == Male)
            s.mark += 10;
        else
            s.mark += 20;

        afile.seekp ((k-1)*sizeof(Student), ios::beg);
        afile.write (reinterpret_cast <const char *> (&s), sizeof (s));

        k++;
    }

    afile.close ();
}
```

4. make file utility

Topic 2

- Pointers

dataType *ptr;

- Pointers and arrays
- char * versus char []
- Applications on pointers and arithmetic
 - : find minimum and the maximum
 - : symmetric
- void *
 - : symmetric
 - : find min max of an array of void*
- the use of new and delete
 - : primitive data type
 - : 1D array
 - : 2D array
 - : etc

```

// void*
// void**
// void*** - Intend to define a 2D array of void*, and store some "enum"
// pointers
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cstdlib>
using namespace std;

typedef void* VoidPtr;

enum Fruit {Orange, Apple, Durian, Peach, Pear};

const int MAXROW = 10;
const int MAXCOL = 8;

VoidPtr getAFruit ();
Fruit getEnumFruit (VoidPtr);

void findMinMax (VoidPtr *[MAXCOL], int, int, VoidPtr&, VoidPtr&);

int compare (VoidPtr, VoidPtr);

int main ()
{
    VoidPtr** vp2D;

    // Define a 2D array of void* that has
    // MAXROW and MAXCOL
    // i.e. VoidPtr vp2D [MAXROW][MAXCOL];
    //

    vp2D = new VoidPtr* [MAXROW];
    for (int i = 0; i < MAXROW; i++)
        vp2D [i] = new VoidPtr [MAXCOL];

    for (int i = 0; i < MAXROW; i++)
    {
        for (int j = 0; j < MAXCOL; j++)

```

```

        {
            vp2D [i][j] = getAFruit ();
        }
    }

    for (int i = 0; i < MAXROW; i++)
    {
        for (int j = 0; j < MAXCOL; j++)
        {
            cout << getEnumFruit (vp2D [i][j]) << "\t";

        }

        cout << endl;
    }

    VoidPtr min;
    VoidPtr max;

    findMinMax (vp2D, MAXROW, MAXCOL, min, max);

    cout << "Mimimum = " << getEnumFruit (min) << endl;
    cout << "Maximum = " << getEnumFruit (max) << endl;
}

VoidPtr getAFruit ()
{
    Fruit *fPtr = new Fruit;

    int k = rand () % 5;

    *fPtr = static_cast <Fruit> (k);

    VoidPtr vp = fPtr;

    return vp;
}

Fruit getEnumFruit (VoidPtr vp)
{
    Fruit *fPtr = static_cast <Fruit *>(vp);

```

```

        return *fPtr;
    }

void findMinMax (VoidPtr *vp2D[MAXCOL], int rows, int columns,
                 VoidPtr& min, VoidPtr& max)
{
    min = vp2D [0][0];
    max = vp2D [0][0];

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            if (compare (vp2D [i][j], min) < 0)
                min = vp2D [i][j];

            if (compare (vp2D [i][j], max) > 0)
                max = vp2D [i][j];
        }
    }
}

int compare (VoidPtr vp1, VoidPtr vp2)
{
    if (getEnumFruit (vp1) > getEnumFruit (vp2))
        return 1;
    else if (getEnumFruit (vp1) < getEnumFruit (vp2))
        return -1;
    else
        return 0;
}

```

```
1      2      4      0      4      4      3      3
2      4      0      0      1      2      1      1
0      2      2      1      1      4      2      3
2      2      1      1      3      0      2      1
1      3      4      2      2      4      0      4
3      1      2      3      3      4      1      1
3      3      2      4      2      2      2      4
3      1      4      3      1      0      0      2
3      1      0      2      4      3      1      0
0      4      0      0      1      1      3      3
Minimum = 0
Maximum = 4
```

// A linked list of void* - implicitly a linked list of integers
// Classes and objects

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

typedef void* VoidPtr;

struct Node;
typedef Node* NodePtr;

struct Node
{
    VoidPtr data;
    NodePtr next;
};

class LinkedList
{
public:
    // constructor (or default constructor)
    LinkedList ();

    // destructor - garbage collection
    ~LinkedList ();

    void addHead (VoidPtr);

    void printList ();

private:
    NodePtr head;

    int getVP (VoidPtr);
};

#include "LinkedList.h"
```

```

// constructor (or default constructor)
LinkedList::LinkedList ()
{
    head = NULL;
}

```

```

// destructor - garbage collection
LinkedList::~~LinkedList ()
{
    NodePtr temp;

    while (head != NULL)
    {
        temp = head;
        head = head -> next;

        cout << "Node with value "
              << getVP (temp -> data);

        delete temp;
        cout << " deleted" << endl;
    }
}

```

```

void LinkedList::addHead (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;
}

```

```

void LinkedList::printList ()
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << getVP (temp -> data) << "\t";
    }
}

```

```

        temp = temp -> next;
    }

    cout << endl;
}

int LinkedList::getVP (VoidPtr item)
{
    return *(static_cast <int *>(item));
}

#include "LinkedList.h"

int main ()
{
    LinkedList alist;

    srand (time (NULL));

    VoidPtr item;
    int *temp;

    for (int i = 1; i <= 10; i++)
    {
        temp = new int;
        *temp = rand () % 100;
        item = temp;

        alist.addHead (item);
    }

    alist.printList ();

    cout << "Garbage collection begins" << endl;

}

```

```
// A linked list of void*  
// - Two linked lists  
// - Interesting remarks, how the compiler does the garbage collection  
// via the destructor
```

```
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
using namespace std;
```

```
typedef void* VoidPtr;
```

```
struct Node;  
typedef Node* NodePtr;
```

```
struct Node  
{  
    VoidPtr data;  
    NodePtr next;  
};
```

```
class LinkedList  
{  
    public:  
        // constructor (or default constructor)  
        LinkedList ();  
  
        // destructor - garbage collection  
        ~LinkedList ();  
  
        void addHead (VoidPtr);  
  
        void printList ();  
  
    private:  
        NodePtr head;  
  
        int getVP (VoidPtr);  
};
```

```

#include "LinkedList.h"

// constructor (or default constructor)
LinkedList::LinkedList ()
{
    head = NULL;
}

// destructor - garbage collection
LinkedList::~~LinkedList ()
{
    NodePtr temp;

    while (head != NULL)
    {
        temp = head;
        head = head -> next;

        cout << "Node with value "
              << getVP (temp -> data);

        delete temp;
        cout << " deleted" << endl;
    }

    cout << "-----" << endl;
}

void LinkedList::addHead (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;
}

void LinkedList::printList ()
{
    NodePtr temp = head;

```

```

        while (temp != NULL)
        {
            cout << getVP (temp -> data) << "\t";
            temp = temp -> next;
        }

        cout << endl;
    }

int LinkedList::getVP (VoidPtr item)
{
    return *(static_cast <int *>(item));
}

#include "LinkedList.h"

int main ()
{
    LinkedList alist1;
    LinkedList alist2;

    srand (time (NULL));

    VoidPtr item;
    int *temp;

    for (int i = 1; i <= 20; i++)
    {
        temp = new int;
        *temp = rand () % 100;
        item = temp;

        if (*temp % 2)
            alist1.addHead (item);
        else
            alist2.addHead (item);
    }

    alist1.printList ();

```

```

alist2.printList ();

cout << "Garbage collection begins" << endl;

}

```

```

63      19      65      37      47      13      81      73      67      76
88      72      30      14      54      72      14      74      50
54
Garbage collection begins
Node with value 88 deleted
Node with value 72 deleted
Node with value 30 deleted
Node with value 14 deleted
Node with value 54 deleted
Node with value 72 deleted
Node with value 14 deleted
Node with value 74 deleted
Node with value 50 deleted
Node with value 76 deleted
Node with value 54 deleted
-----
Node with value 63 deleted
Node with value 19 deleted
Node with value 65 deleted
Node with value 37 deleted
Node with value 47 deleted
Node with value 13 deleted
Node with value 81 deleted
Node with value 73 deleted
Node with value 67 deleted
-----

```

```

// A linked list of void*
// - store some C-string
//
// Node structure is private
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cstring>
using namespace std;

typedef void* VoidPtr;

class LinkedList
{
    public:
        // constructor (or default constructor)
        LinkedList ();

        // destructor - garbage collection
        ~LinkedList ();

        void addHead (VoidPtr);
        void printList ();

    private:
        struct Node;
        typedef Node* NodePtr;

        struct Node
        {
            VoidPtr data;
            NodePtr next;
        };

        NodePtr head;

        char * getVP (VoidPtr);
};

```

```

#include "LinkedList.h"

// constructor (or default constructor)
LinkedList::LinkedList ()
{
    head = NULL;
}

// destructor - garbage collection
LinkedList::~~LinkedList ()
{
    NodePtr temp;

    while (head != NULL)
    {
        temp = head;
        head = head -> next;

        cout << "Node with value "
              << getVP (temp -> data);

        delete temp;
        cout << " deleted" << endl;
    }
}

void LinkedList::addHead (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    head = temp;
}

void LinkedList::printList ()
{
    NodePtr temp = head;

    while (temp != NULL)

```

```

        {
            cout << getVP (temp -> data) << "\t";
            temp = temp -> next;
        }

        cout << endl;
    }

char * LinkedList::getVP (VoidPtr item)
{
    return static_cast <char *>(item);
}

#include "LinkedList.h"

int main ()
{
    LinkedList alist;

    srand (time (NULL));

    char str [80];
    VoidPtr item;

    cout << "Enter a string: ";
    cin.getline (str, 80);

    char *p = strtok (str, "., ?");
    while (p != NULL)
    {
        item = p;
        alist.addHead (item);
        p = strtok (NULL, "., ?");
    }

    alist.printList ();
    cout << "Garbage collection begins" << endl;
}

```

// A class to describe a rectangle

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

class Rectangle
{
    public:
        // default constructor
        Rectangle ();

        // Other constructor
        Rectangle (int, int);

        // destructor
        ~Rectangle ();

        // What can we do?
        int getArea () const;
        int getPerimeter () const;
        void print () const;

        // accessor functions
        int getLength () const;
        int getWidth () const;

        // mutator functions
        void set (int, int);
        void setLength (int);
        void setWidth (int);

        // Other operations
        bool isEqual (const Rectangle&) const;
        bool isGreater (const Rectangle&) const;

    private:
        int length, width;

        int area () const;
        int perimeter () const;
};
```

```

#include "Rectangle.h"

// default constructor
Rectangle::Rectangle ()
{
    length = 1;
    width = 1;
}

// Other constructor
Rectangle::Rectangle (int length, int width)
{
    this -> length = length;
    this -> width = width;
}

// destructor
Rectangle::~Rectangle ()
{
    // let compiler do it
}

// What can we do?
int Rectangle::area () const
{
    return length * width;
}

int Rectangle::perimeter () const
{
    return 2 * (length + width);
}

int Rectangle::getArea () const
{
    return area ();
}

int Rectangle::getPerimeter () const
{

```

```

        return perimeter ();
    }

void Rectangle::print () const
{
    cout << "R (" << length
        << ", " << width << ")";
}

        // accessor functions
int Rectangle::getLength () const
{
    return length;
}

int Rectangle::getWidth () const
{
    return width;
}

        // mutator functions
void Rectangle::set (int length, int width)
{
    (*this).length = length;
    (*this).width = width;
}

void Rectangle::setLength (int length)
{
    this -> length = length;
}

void Rectangle::setWidth (int width)
{
    this -> width = width;
}

        // Other operations
bool Rectangle::isEqual (const Rectangle& r) const
{

```

```

        return ((this -> length == r.length) &&
                (this -> width == r.width)) ||
                ((this -> length == r.width) &&
                (this -> width == r.length));
    }

bool Rectangle::isGreater (const Rectangle& r) const
{
    return this -> area () > r.area ();
}

#include "Rectangle.h"

int main ()
{
    Rectangle r1;
    Rectangle r2 (3, 4);

    cout << "r1 = ";
    r1.print ();
    cout << endl;

    cout << "r2 = ";
    r2.print ();
    cout << endl;

    if (r1.isEqual (r2))
        cout << "r1 is equal to r2" << endl;
    else
        cout << "r1 is not equal to r2" << endl;

    if (r2.isGreater (r1))
        cout << "r2 is greater than r1" << endl;
    else
        cout << "r2 is not greater than r1" << endl;

    cout << "Area of r1 = " << r1.getArea () << endl;
    cout << "Area of r2 = " << r2.getArea () << endl;
}

```

// A class to describe a rectangle

// Note the declarations of overload operators

#include <iostream>

using namespace std;

class Rectangle

{

friend ostream& operator<< (ostream&, const Rectangle&);

friend istream& operator>> (istream&, Rectangle&);

public:

// default constructor

Rectangle ();

// Other constructor

Rectangle (int, int);

// destructor

~Rectangle ();

// copy constructor

Rectangle (const Rectangle&);

// What can we do?

int area () const;

int perimeter () const;

void print () const;

// accessor functions

int getLength () const;

int getWidth () const;

// mutator functions

void set (int, int);

void setLength (int);

void setWidth (int);

// Other operations

bool isEqual (const Rectangle&) const;

bool isGreater (const Rectangle&) const;

```

        // Operators overload
        bool operator== (const Rectangle&) const;
        bool operator> (const Rectangle&) const;

        Rectangle& operator>= (Rectangle&);

    private:
        int length, width;
};

#include "Rectangle.h"

ostream& operator<< (ostream& os, const Rectangle& r)
{
    os << "R ("
        << r.length
        << ", "
        << r.width
        << ")";

    return os;
}

istream& operator>> (istream& is, Rectangle& r)
{
    // input format for rectangle is
    // *** - ***

    char hyphen;

    is >> r.length;
    is >> hyphen;
    is >> r.width;

    return is;
}

// default constructor

```

```

Rectangle::Rectangle ()
{
    length = 1;
    width = 1;
}

// Other constructor
Rectangle::Rectangle (int length, int width)
{
    this -> length = length;
    this -> width = width;
}

// destructor
Rectangle::~Rectangle ()
{
    // let compiler do it
}

Rectangle::Rectangle (const Rectangle& r)
{
    this -> length = r.length;
    this -> width = r.width;
}

// What can we do?
int Rectangle::area () const
{
    return length * width;
}

int Rectangle::perimeter () const
{
    return 2 * (length + width);
}

void Rectangle::print () const
{
    cout << "R (" << length
        << ", " << width << ")";
}

```

```

}

// accessor functions
int Rectangle::getLength () const
{
    return length;
}

int Rectangle::getWidth () const
{
    return width;
}

// mutator functions
void Rectangle::set (int length, int width)
{
    (*this).length = length;
    (*this).width = width;
}

void Rectangle::setLength (int length)
{
    this -> length = length;
}

void Rectangle::setWidth (int width)
{
    this -> width = width;
}

// Other operations
bool Rectangle::isEqual (const Rectangle& r) const
{
    return ((this -> length == r.length) &&
            (this -> width == r.width)) ||
           ((this -> length == r.width) &&
            (this -> width == r.length));
}

bool Rectangle::isGreater (const Rectangle& r) const

```

```

{
    return this -> area () > r.area ();
}

bool Rectangle::operator==(const Rectangle& r) const
{
    return ((this -> length == r.length) &&
            (this -> width == r.width)) ||
            ((this -> length == r.width) &&
            (this -> width == r.length));
}

bool Rectangle::operator> (const Rectangle& r) const
{
    return this -> area () > r.area ();
}

Rectangle& Rectangle::operator>= (Rectangle& r)
{
    if (this -> area () >= r.area ())
        return *this;
    else
        return r;
}

#include "Rectangle.h"

int main ()
{
    Rectangle r1;
    Rectangle r2 (3, 4);

    cout << "r1 = ";
    r1.print ();
    cout << endl;

    cout << "r2 = ";
    r2.print ();
    cout << endl;
}

```

```

if (r1.isEqual (r2))
    cout << "r1 is equal to r2" << endl;
else
    cout << "r1 is not equal to r2" << endl;

if (r2.isGreater (r1))
    cout << "r2 is greater than r1" << endl;
else
    cout << "r2 is not greater than r1" << endl;

cout << "Area of r1 = " << r1.area () << endl;
cout << "Area of r2 = " << r2.area () << endl;

Rectangle r3 (r2);

cout << "r3 = ";
r3.print ();
cout << endl;

if (r3.isEqual (r2))
    cout << "r3 is equal to r2" << endl;
else
    cout << "r3 is not equal to r2" << endl;

cout << "-----" << endl;

if (r2 == r3)
    cout << "r2 == r3" << endl;
else
    cout << "r2 != r3" << endl;

if (r2 > r1)
    cout << "r2 > r1" << endl;
else
    cout << "r2 is not > r1" << endl;

cout << "-----" << endl;

Rectangle r4 = (r2 >= r3);

```

```

cout << "r4 = ";
r4.print ();
cout << endl;

cout << "-----" << endl;

cout << "The rectangles are "
    << r1 << ", " << r2 << " and " << r4
    << endl;

cout << "-----" << endl;

Rectangle r;

cout << "Enter a rectangle: ";
cin >> r;

cout << "The reactangle is " << r << endl;
}

```

```

// A triangle class
// Three types of triangles
// - equilateral
// - isosceles
// - scalene
//
// Multiple constructors
#include <iostream>
#include <cmath>
using namespace std;

class Triangle
{
    friend ostream& operator<< (ostream&, const Triangle&);
    friend istream& operator>> (istream&, Triangle&);

public:
    // default constructor
    Triangle ();

    // Other constructors
    // Equilateral triangle
    Triangle (int);

    // Isosceles triangle
    Triangle (int, int);

    // Scalene triangle
    Triangle (int, int, int);

    // What we can do?
    double area () const;

    // What operations?
    Triangle& operator>= (Triangle&);
    bool operator<= (const Triangle&) const;

    // add in assessor functions
    // add in mutator functions

```

```

        private:
            int a, b, c;
};

#include "Triangle.h"

ostream& operator<< (ostream& os, const Triangle& t)
{
    // Output format
    // E (a)
    // I (a, b)
    // S (a, b, c)

    if (t.a * t.b * t.c != 0)
    {
        os << "S (" << t.a << ", " << t.b << ", " << t.c << ")";
    }
    else if (t.b == 0 && t.c == 0)
    {
        os << "E (" << t.a << ")";
    }
    else
    {
        os << "I (" << t.a << ", " << t.c << ")";
    }

    return os;
}

istream& operator>> (istream& is, Triangle& t)
{
    // Input format
    // e ***
    // i *** ***
    // s *** *** ***
    char type;

    t.b = 0;

```

```

    t.c = 0;

    is >> type;

    switch (type)
    {
        case 'e': is >> t.a;
                    break;
        case 'i': is >> t.a >> t.c;
                    break;
        default : is >> t.a >> t.b >> t.c;
    }

    return is;
}

// default constructor
Triangle::Triangle ()
{
    // leave it empty
}

// Other constructors
// Equilateral triangle
Triangle::Triangle (int a)
{
    this -> a = a;
    this -> b = 0;
    this -> c = 0;
}

// Isosceles triangle
Triangle::Triangle (int a, int b)
{
    this -> a = a;
    this -> b = 0;
    this -> c = b;
}

```

```

        //Scalene triangle
Triangle::Triangle (int a, int b, int c)
{
    this -> a = a;
    this -> b = b;
    this -> c = c;
}

        // What we can do?
double Triangle::area () const
{
    double s;

    if (a * b * c != 0)
    {
        s = (a + b + c) / 2.0;
        return sqrt (s * (s - a) * (s - b) * (s - c));
    }
    else if (b == 0 && c == 0)
    {
        s = 3.0 / 2 * a;
        return sqrt (s * (s - a) * (s - a) * (s - a));
    }
    else
    {
        s = (a + a + c) / 2.0;
        return sqrt (s * (s - a) * (s - a) * (s - c));
    }
}

        // What operations?
Triangle& Triangle::operator>= (Triangle& t)
{
    if (this -> area () >= t.area ())
        return *this;
    else
        return t;
}

```

```

bool Triangle::operator<= (const Triangle& t) const
{
    return this -> area () <= t.area ();
}

#include "Triangle.h"

int main ()
{
    Triangle t1 (6);
    Triangle t2 (7, 8);
    Triangle t3 (4, 5, 6);

    cout << "t1 = " << t1 << endl;
    cout << "t2 = " << t2 << endl;
    cout << "t3 = " << t3 << endl;

    Triangle t = (t2 >= t3);
    cout << "t = " << t << endl;

    for (int i = 1; i <= 3; i++)
    {
        cout << "Enter a triangle: ";
        cin >> t;

        cout << "The triangle is "
             << t
             << endl;
    }
}

```

```

// A box has a shape and a height
// Two types of boxes
// - circular box
// - rectangular box
// Two types of shapes
// - circle shape
// - rectangle shape
//
// Communication between classes

```

```

#include <iostream>
using namespace std;

```

```

class Shape
{
    friend ostream& operator<< (ostream&, const Shape&);
    friend istream& operator>> (istream&, Shape&);

    public:
        Shape ();

        Shape (int);

        Shape (int, int);

        double area () const;

    private:
        int a, b;
};

```

```

#include "Shape.h"

```

```

ostream& operator<< (ostream& os, const Shape& s)
{
    // Output format
    // Circle (***)
    // Rectangle (***, ***)

    if (s.b == 0)

```

```

        os << "Circle (" << s.a << ")";
    else
        os << "Rectangle (" << s.a << ", " << s.b << ")";

    return os;
}

```

```

istream& operator>> (istream& is, Shape& s )
{
    // Input format
    // c ***
    // r *** ***

    char type;

    is >> type;

    switch (type)
    {
        case 'c': is >> s.a;
                    s.b = 0;
                    break;
        case 'r': is >> s.a >> s.b;
                    break;
    }

    return is;
}

```

```

Shape::Shape ()
{
}

```

```

Shape::Shape (int radius)
{
    a = radius;
    b = 0;
}

```

```
Shape::Shape (int length, int width)
{
    this -> a = length;
    this -> b = width;
}
```

```
double Shape::area () const
{
    if (a * b == 0)
        return 3.14159 * a * a;
    else
        return 1.0 * a * b;
}
```

```
#include "Shape.h"
```

```
class Box
{
    friend ostream& operator<< (ostream&, const Box&);
    friend istream& operator>> (istream&, Box&);

    public:
        Box ();

        Box (Shape, int);

        double volume () const;

    private:
        Shape s;
        int height;
};
```

```
#include "Box.h"
```

```
ostream& operator<< (ostream& os, const Box& b)
{
    // output format
```

```

        // Box (Circle (**), ***)
        // Box (Rectangle (**, **), ***)

        os << "Box (" << b.s << ", " << b.height << ")";

        return os;
    }

istream& operator>> (istream& is, Box& b)
{
    // format
    // c ** **
    // r ** ** **

    is >> b.s >> b.height;

    return is;
}

Box::Box ()
{
}

Box::Box (Shape s, int height)
{
    this -> s = s;
    this -> height = height;
}

double Box::volume () const
{
    return s.area () * height;
}

#include "Box.h"

int main ()
{
    Shape s1 (23);

```

```

Shape s2 (11, 22);

cout << "s1 = " << s1 << endl;
cout << "s2 = " << s2 << endl;
cout << endl;

Box b1 (s1, 123);
Box b2 (s2, 456);

cout << "b1 = " << b1 << endl;
cout << "b2 = " << b2 << endl;

/*
Shape s;

for (int i = 1; i <= 2; i++)
{
    cout << "Enter a shape: ";
    cin >> s;

    cout << "The shape is " << s << endl;
    cout << endl;
}
*/

Box b;

cout << endl;

for (int i = 1; i <= 2; i++)
{
    cout << "Enter a box: ";
    cin >> b;

    cout << "The box is " << b << endl;
    cout << endl;
}
}

```

// The use of namespaces

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

namespace CSCI114
{
    void message1 ()
    {
        cout << "Welcome to 114" << endl;
    }

    void message2 ()
    {
        cout << "I hope you enjoyed the course" << endl;
    }
}

namespace CSCI124
{
    void message1 ();
    void message2 ();
}

int main ()
{
    CSCI114::message1 ();
    CSCI114::message2 ();
    CSCI124::message1 ();
    CSCI124::message2 ();
}

void CSCI124::message1 ()
{
    cout << "Welcome to CSCI124" << endl;
}

void CSCI124::message2 ()
{
    cout << "I know 124 is difficult" << endl;
}
```

// The use of namespaces

// The use of “using”

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

namespace CSCI114
{
    void message1 ()
    {
        cout << "Welcome to 114" << endl;
    }

    void message2 ()
    {
        cout << "I hope you enjoyed the course" << endl;
    }
}

namespace CSCI124
{
    void message1 ();
    void message2 ();
}

int main ()
{
    {
        using namespace CSCI114;

        message1 ();
        message2 ();
    }

    {
        using namespace CSCI124;

        message1 ();
        message2 ();
    }
}
```

```
}

void CSCI124::message1 ()
{
    cout << "Welcome to CSCI124" << endl;
}

void CSCI124::message2 ()
{
    cout << "I know 124 is difficult" << endl;
}
```

// The use of namespaces

// - A class definition in a namespace

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

namespace CSCI114
{
    void message1 ()
    {
        cout << "Welcome to 114" << endl;
    }

    void message2 ()
    {
        cout << "I hope you enjoyed the course" << endl;
    }
}

namespace CSCI124
{
    class Message
    {
    public:
        Message ();

        void message1 ();
        void message2 ();
    };
}

int main ()
{
    {
        using namespace CSCI114;

        message1 ();
        message2 ();
    }

    {
```

```

        using namespace CSCI124;

        Message m;

        m.message1 ();
        m.message2 ();

    }
}

CSCI124::Message::Message ()
{
    // do nothing
}

void CSCI124::Message::message1 ()
{
    cout << "Welcome to CSCI124" << endl;
}

void CSCI124::Message::message2 ()
{
    cout << "I know 124 is difficult" << endl;
}

```

// static members in a class

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

```
class Triangle
{
```

```
    public:
```

```
        Triangle ();
        Triangle (int);
        Triangle (int, int);
        Triangle (int, int, int);
```

```
        static int no;
```

```
    private:
```

```
        int a, b, c;
```

```
};
```

```
Triangle::Triangle ()
```

```
{
    ++no;
}
```

```
Triangle::Triangle (int a)
```

```
{
    ++no;
}
```

```
Triangle::Triangle (int a, int b)
```

```
{
    ++no;
}
```

```
Triangle::Triangle (int a, int b, int c)
```

```
{
    ++no;
}
```

```
int Triangle::no = 0;
```

```

int main ()
{
    Triangle t1;
    Triangle t2 (2);
    Triangle t3 (3, 2);
    Triangle t4 (3, 4, 5);

    cout << Triangle::no << endl; // 4
    cout << t4.no << endl;        // 4
    cout << t1.no << endl;

    Triangle *tPtr;
    tPtr = &t4;

    cout << tPtr -> no << endl;

    // This is bad if no is public static no
    Triangle::no += 100;
    cout << Triangle::no << endl;

}

```

// static members in a class

// static variable is now private and access its value via static function

```
#include <iostream>
```

```
using namespace std;
```

```
class Triangle
```

```
{
```

```
    public:
```

```
        Triangle ();
```

```
        Triangle (int);
```

```
        Triangle (int, int);
```

```
        Triangle (int, int, int);
```

```
        static int getNo ();
```

```
        int printNo () const;
```

```
    private:
```

```
        int a, b, c;
```

```
        static int no;
```

```
};
```

```
Triangle::Triangle ()
```

```
{
```

```
    ++no;
```

```
}
```

```
Triangle::Triangle (int a)
```

```
{
```

```
    ++no;
```

```
}
```

```
Triangle::Triangle (int a, int b)
```

```
{
```

```
    ++no;
```

```
}
```

```
Triangle::Triangle (int a, int b, int c)
```

```
{
```

```
    ++no;
```

```

}

int Triangle::no = 0;

int Triangle::getNo ()
{
    return no;
}

int Triangle::printNo () const
{
    cout << "No of triangles created = "
         << no
         << endl;
}

int main ()
{
    Triangle t1;
    Triangle t2 (2);
    Triangle t3 (3, 2);
    Triangle t4 (3, 4, 5);

    cout << Triangle::getNo () << endl;

    t4.printNo ();
}

```

// try - throw - catch

#include <iostream>

#include <cmath>

using namespace std;

double getSqrt (double);

int main ()

{

double x;

try

{

cout << "Enter a real number: ";

cin >> x;

cout << "The sqrt of " << x << " is "

<< getSqrt (x)

<< endl;

}

catch (double x)

{

cout << "Exception caught: "

<< " because " << x

<< " is negative"

<< endl;

}

cout << "We still can continue to work" << endl;

}

double getSqrt (double x)

{

if (x < 0)

throw x;

return sqrt (x);

}

```

// try - throw - catch
//
// More than one catches
#include <iostream>
#include <cmath>
using namespace std;

double getSqrt (double);
float getValue (int, int);

int main ()
{
    double x;
    int m, n;

    int *temp;

    try
    {
        temp = new int [100];

        cout << "Enter a real number: ";
        cin >> x;

        cout << "The sqrt of " << x << " is "
              << getSqrt (x)
              << endl;

        cout << "\nEnter two integers: ";
        cin >> m >> n;

        cout << "The value is "
              << getValue (m, n)
              << endl;

        delete [] temp;
    }
    catch (double x)
    {
        cout << "Real number exception caught: "

```

```

        << " because " << x
        << " is negative"
        << endl;

        delete [] temp;
    }
    catch (int a)
    {
        cout << "Integer exception caught: "
              << " division by " << a
              << endl;

        delete [] temp;
    }

    cout << "We still can continue to work" << endl;
}

double getSqrt (double x)
{
    if (x < 0)
        throw x;

    return sqrt (x);
}

float getValue (int m, int n)
{
    float x;

    if (n == 0)
        throw n;

    x = 1.0 * m / n;
    return x;
}

```

```
// try - throw - catch  
//  
// try block can be nested
```

```
#include <iostream>  
#include <cmath>  
using namespace std;
```

```
double getSqrt (double);  
float getValue (int, int);
```

```
int main ()  
{  
    double x;  
    int m, n;  
  
    try  
    {  
        cout << "Enter a real number: ";  
        cin >> x;  
  
        cout << "The sqrt of " << x << " is "  
            << getSqrt (x)  
            << endl;  
  
        try  
        {  
            cout << "\nEnter two integers: ";  
            cin >> m >> n;  
  
            cout << "The value is "  
                << getValue (m, n)  
                << endl;  
        }  
        catch (int a)  
        {  
            cout << "Integer exception caught: "  
                << " division by " << a  
                << endl;  
        }  
    }  
}
```

```

        cout << "\nI continue to work inside the outer try block"
            << endl;

    }
    catch (double x)
    {
        cout << "Real number exception caught: "
            << " because " << x
            << " is negative"
            << endl;
    }

    cout << "\nWe still can continue to work" << endl;
}

double getSqrt (double x)
{
    if (x < 0)
        throw x;

    return sqrt (x);
}

float getValue (int m, int n)
{
    float x;

    if (n == 0)
        throw n;

    x = 1.0 * m / n;
    return x;
}

```

// Throw some objects

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

class NotHappyException
{
    public:
        NotHappyException ();

        void printMessage () const;

    private:
        const char *message;
};

class HappyException
{
    public:
        HappyException ();

        HappyException (char *);

        void printMessage () const;

    private:
        const char *message;
};

void processExam (int);

int main ()
{
    int exam;

    srand (time (NULL));
```

```

for (int i = 1; i <= 10; i++)
{
    exam = rand () % 101;
    cout << "Your exam = " << exam << endl;

    try
    {
        processExam (exam);
    }
    catch (NotHappyException e)
    {
        e.printMessage ();
    }
    catch (HappyException e)
    {
        e.printMessage ();
    }

    cout << endl;
}
}

```

```

NotHappyException::NotHappyException ()
{
    message = "I failed the exam";
}

```

```

void NotHappyException::printMessage () const
{
    cout << message << endl;
}

```

```

/* ----- */

```

```

HappyException::HappyException ()
{
    message = "I passed the exam";
}

```

```

HappyException::HappyException (char *message)

```

```
{
    this -> message = message;
}
```

```
void HappyException::printMessage () const
{
    cout << message << endl;
}
```

```
void processExam (int exam)
{
    if (exam >= 85)
        throw HappyException ("Hey!! I got HDist");
    else if (exam >= 75)
        throw HappyException ("Not bad! I got a Dist");
    else if (exam >= 65)
        throw HappyException ("I scored only a Credit");
    else if (exam >= 50)
        throw HappyException ();
    else
        throw NotHappyException ();
}
```

```
Your exam = 97
Hey!! I got HDist

Your exam = 10
I failed the exam

Your exam = 73
I scored only a Credit

Your exam = 29
I failed the exam

Your exam = 6
I failed the exam

Your exam = 6
I failed the exam

Your exam = 22
I failed the exam

Your exam = 100
Hey!! I got HDist

Your exam = 81
Not bad! I got a Dist

Your exam = 87
Hey!! I got HDist
```

// User defined exception classes

// A static function inside a class

```
#include <iostream>
```

```
#include <ctime>
```

```
#include <cstdlib>
```

```
using namespace std;
```

```
class NotHappyException
```

```
{
```

```
    public:
```

```
        NotHappyException ();
```

```
        void printMessage () const;
```

```
    private:
```

```
        const char *message;
```

```
};
```

```
NotHappyException::NotHappyException ()
```

```
{
```

```
    message = "I am sad";
```

```
}
```

```
void NotHappyException::printMessage () const
```

```
{
```

```
    cout << message << endl;
```

```
}
```

```
class HappyException
```

```
{
```

```
    public:
```

```
        HappyException ();
```

```
        HappyException (char *);
```

```
        void printMessage () const;
```

```
        static void otherMessage ();
```

```
    private:
```

```
        const char *message;
```

```
};
```

```

HappyException::HappyException ()
{
    message = "I pass the exam";
}

HappyException::HappyException (char *message)
{
    this -> message = message;
}

void HappyException::printMessage () const
{
    cout << message << endl;
}

void HappyException::otherMessage ()
{
    cout << "I score only distinction" << endl;
}

int main ()
{
    int exam;

    srand (time (NULL));

    try
    {
        exam = rand () % 101;

        cout << "Your exam mark = ";
        cout << exam << endl;

        if (exam < 50)
            throw NotHappyException ();
        else if (exam < 75)
            throw HappyException ();
        else

```

```

        throw HappyException ("Hey!!! I score HDist");
    }
    catch (NotHappyException e)
    {
        e.printMessage ();
    }
    catch (HappyException e)
    {
        if (exam >= 75 && exam <= 84)
            HappyException::otherMessage ();
        else
            e.printMessage ();
    }

    cout << "I continue to work hard" << endl;
}

```

// try - throw - catch

//

// another example

```
#include <iostream>
```

```
#include <cmath>
```

```
using namespace std;
```

```
// area of a triangle
```

```
double area (int, int, int);
```

```
int main ()
```

```
{
```

```
    int a, b, c;
```

```
    try
```

```
    {
```

```
        cout << "Enter the sides of a triangle: ";
```

```
        cin >> a >> b >> c;
```

```
        cout << "The area is "
```

```
            << area (a, b, c)
```

```
            << endl;
```

```
    }
```

```
    catch (int n)
```

```
    {
```

```
        cout << "Exception caught: ";
```

```
        switch (n)
```

```
        {
```

```
            case 0: cout << "One of the integers "
```

```
                    << " is invalid"
```

```
                    << endl;
```

```
                    break;
```

```
            case 1: cout << "Triangle cannot be formed"
```

```
                    << endl;
```

```
        }
```

```
    }
```

```
    cout << "\nWe still work outside" << endl;
```

```
}
```

```
double area (int a, int b, int c)
```

```
{
```

```
    if (a <= 0 || b <= 0 || c <= 0)
```

```
        throw 0;
```

```
    else if (!(a + b > c && b + c > a && a + c > b))
```

```
        throw 1;
```

```
    else
```

```
    {
```

```
        double s = (a + b + c) / 2.0;
```

```
        return sqrt (s * (s - a) * (s - b) * (s - c));
```

```
    }
```

```
}
```

// bad_alloc exception

#include <iostream>

using namespace std;

int main ()

{

int *temp;

try

{

while (true)

{

temp = new int [10000];

// when memory overflow

// it throws bad_alloc exception

}

}

catch (bad_alloc e)

{

cout << "Memory overflow" << endl;

}

cout << "And so what" << endl;

}

// A linear linked list with head and tail pointers

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

typedef void * VoidPtr;

class LinkedList
{
    public:
        // default constructor
        LinkedList ();

        // destructor
        ~LinkedList ();

        // copy constructor
        LinkedList (const LinkedList&);

        // Basic operations
        void addToHead (VoidPtr);
        void addToTail (VoidPtr);
        VoidPtr removeHead ();
        VoidPtr removeTail ();

        // Future development
        // - General insert
        // - General remove

        // Other functions
        void print () const;
        int getVP (VoidPtr) const;
        bool isEmpty () const;

        // Other static function
        static int getNo ();
```

```

private:
    struct Node;
    typedef Node* NodePtr;

    struct Node
    {
        VoidPtr data;
        NodePtr next;
    };

    NodePtr head, tail;

    // monitor the number of nodes created
    static int no;

    void testThrowException (const NodePtr&) const;
};

#include "LinkedList.h"

// default constructor
LinkedList::LinkedList ()
{
    // when a list is empty
    head = NULL;
    tail = NULL;
}

// destructor
LinkedList::~~LinkedList ()
{
    NodePtr temp;

    while (head != NULL)
    {
        temp = head;
        head = head -> next;

        cout << "Node with value "
             << getVP (temp -> data);
    }
}

```

```

        delete temp;
        --no;
        cout << " was deleted" << endl;
    }

    // very important
    tail = NULL;
}

// copy constructor
LinkedList::LinkedList (const LinkedList& alist)
{
    head = NULL;
    tail = NULL;

    NodePtr temp = alist.head;

    while (temp != NULL)
    {
        addToTail (temp -> data);
        temp = temp -> next;
    }
}

void LinkedList::addToHead (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    if (head == NULL)
        tail = temp;

    head = temp;
    ++no;
}

void LinkedList::addToTail (VoidPtr item)

```

```

{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;

    if (tail == NULL)
        head = temp;
    else
        tail -> next = temp;

    tail = temp;
    ++no;
}

VoidPtr LinkedList::removeHead ()
{
    VoidPtr item;

    try
    {
        // This function's call replaces the
        // the following if statement
        testThrowException (head);
        // if (head == NULL)
        //     throw exception ();

        NodePtr temp = head;

        head = head -> next;

        if (head == NULL)
            tail = NULL;

        item = temp -> data;
        delete temp;
        --no;
    }
    catch (exception e)
    {
        cout << "Exception caught: "

```

```

        << "Removed head failed"
        << endl;
    }

    return item;
}

VoidPtr LinkedList::removeTail ()
{
    VoidPtr item;

    try
    {
        // This function's call replaces the
        // the following if statement
        testThrowException (tail);
        // if (tail == NULL)
        //     throw exception ();

        NodePtr temp = tail;

        // only one node
        if (tail == head)
        {
            tail = NULL;
            head = NULL;
        }
        else
        {
            NodePtr curr = head;

            // to get the node before the tail node
            while (curr -> next != tail)
                curr = curr -> next;

            curr -> next = NULL;
            tail = curr;
        }

        item = temp -> data;
    }
}

```

```

        delete temp;
        --no;
    }
    catch (exception e)
    {
        cout << "Exception caught: "
              << "Removed tail failed"
              << endl;
    }

    return item;
}

void LinkedList::print () const
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << getVP (temp -> data) << endl;
        temp = temp -> next;
    }
}

int LinkedList::getVP (VoidPtr item) const
{
    int *temp = static_cast <int *> (item);
    return *temp;
}

bool LinkedList::isEmpty () const
{
    if (head == NULL && tail == NULL)
        return true;
    else
        return false;
}

int LinkedList::getNo ()

```

```

{
    return no;
}

int LinkedList::no = 0;

void LinkedList::testThrowException (const NodePtr& aPtr) const
{
    if (aPtr == NULL)
        throw exception ();
}

#include "LinkedList.h"

int main ()
{
    LinkedList alist;

    alist.removeHead ();
    alist.removeTail ();

    cout << "-----" << endl;

    srand (time (NULL));

    int *temp;
    VoidPtr item;

    for (int i = 1; i <= 10; i++)
    {
        temp = new int;
        *temp = rand ();
        item = temp;

        if (*temp % 2)
        {
            alist.addToHead (item);
            cout << "Added to head " << *temp << endl;
        }
        else

```

```

        {
            alist.addToTail (item);
            cout << "Added to tail " << *temp << endl;
        }
    }

    cout << "-----" << endl;

    cout << "The list is " << endl;
    alist.print ();

    cout << "-----" << endl;

    cout << "Invoke the copy constructor" << endl;

    LinkedList duplicateList (alist);

    cout << "The duplicate list is " << endl;

    duplicateList.print ();

    cout << "-----" << endl;

    cout << "Garbage collection begins" << endl;
}

```

// We modify the list class and define our stack class

```
// A stack is implemented using a linear linked
// list with head and the tail pointer (though
// is not necessary for this tail pointer)
//
```

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
typedef void * VoidPtr;
```

```
class Stack
```

```
{
```

```
    public:
```

```
        // default constructor
        Stack ();
```

```
        // destructor
        ~Stack ();
```

```
        // copy constructor
        Stack (const Stack&);
```

```
        // Basic stack operations
        void push (VoidPtr);
        VoidPtr pop ();
        VoidPtr top () const;
```

```
        // Future development
        // - General insert
        // - General remove
```

```
        // Other functions
        void print () const;
        int getVP (VoidPtr) const;
        bool isEmpty () const;
```

```

        // Other static function
        static int getNo ();

private:
    struct Node;
    typedef Node* NodePtr;

    struct Node
    {
        VoidPtr data;
        NodePtr next;
    };

    NodePtr head, tail;

    // monitor the number of nodes created
    static int no;

    void testThrowException (const NodePtr&) const;

    void addToHead (VoidPtr);
    VoidPtr removeHead ();
};

#include "Stack.h"

        // default constructor
Stack::Stack ()
{
    // when a list is empty
    head = NULL;
    tail = NULL;
}

        // destructor
Stack::~~Stack ()
{
    NodePtr temp;

```

```

while (head != NULL)
{
    temp = head;
    head = head -> next;

    cout << "Node with value "
         << getVP (temp -> data);

    delete temp;
    --no;
    cout << " was deleted" << endl;
}

// very important
tail = NULL;
}

// copy constructor
Stack::Stack (const Stack& alist)
{
    head = NULL;
    tail = NULL;

    NodePtr temp = alist.head;

    while (temp != NULL)
    {
        addToHead(temp -> data);
        temp = temp -> next;
    }
}

void Stack::addToHead (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = head;

    if (head == NULL)

```

```

        tail = temp;

    head = temp;
    ++no;
}

VoidPtr Stack::removeHead ()
{
    VoidPtr item;

    try
    {
        // This function's call replaces the
        // the following if statement
        testThrowException (head);
        // if (head == NULL)
        //     throw exception ();

        NodePtr temp = head;

        head = head -> next;

        if (head == NULL)
            tail = NULL;

        item = temp -> data;
        delete temp;
        --no;
    }
    catch (exception e)
    {
        cout << "Exception caught: "
              << "Removed head failed"
              << endl;
    }

    return item;
}

```

```

void Stack::print () const
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << getVP (temp -> data) << endl;
        temp = temp -> next;
    }
}

int Stack::getVP (VoidPtr item) const
{
    int *temp = static_cast <int *> (item);
    return *temp;
}

bool Stack::isEmpty () const
{
    if (head == NULL && tail == NULL)
        return true;
    else
        return false;
}

int Stack::getNo ()
{
    return no;
}

int Stack::no = 0;

void Stack::testThrowException (const NodePtr& aPtr) const
{
    if (aPtr == NULL)
        throw exception ();
}

```

```

void Stack::push (VoidPtr item)
{
    addToHead (item);
}

VoidPtr Stack::pop ()
{
    removeHead ();
}

VoidPtr Stack::top () const
{
    VoidPtr item;

    try
    {
        testThrowException (head);

        item = head -> data;
    }
    catch (exception e)
    {
        cout << "Exception caught: Empty stack, no top element" <<
endl;
    }

    return item;
}

#include "Stack.h"

int main ()
{
    Stack s;

    srand (time (NULL));

    int *temp;
    VoidPtr item;

```

```

for (int i = 1; i <= 10; i++)
{
    temp = new int;
    *temp = rand ();
    item = temp;

    s.push (item);
    cout << "Pushed item " << *temp << endl;
}

// Test on pop function
cout << endl;
cout << "Test on pop function" << endl;

while (!s.isEmpty ())
{
    item = s.pop ();

    cout << "Item " << s.getVP (item) << " was popped" << endl;
}

cout << "\nA test on top function" << endl;
s.top ();

cout << "\nGarbage collection" << endl;
}

```

```
Pushed item 28011
Pushed item 21634
Pushed item 5848
Pushed item 9396
Pushed item 2422
Pushed item 18787
Pushed item 13173
Pushed item 23139
Pushed item 19992
Pushed item 12714
```

```
Test on pop function
Item 12714 was popped
Item 19992 was popped
Item 23139 was popped
Item 13173 was popped
Item 18787 was popped
Item 2422 was popped
Item 9396 was popped
Item 5848 was popped
Item 21634 was popped
Item 28011 was popped
```

```
A test on top function
Exception caught: Empty stack, no top element
```

```
Garbage collection
```

```
Press Enter to return to Quincy..._
```

// We modify the list class and define our Queue class

```
// A queue is implemented using a linear linked list
// with head and the tail pointer
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
typedef void * VoidPtr;
```

```
class Queue
{
```

```
    public:
```

```
        // default constructor
        Queue ();
```

```
        // destructor
        ~Queue ();
```

```
        // copy constructor
        Queue (const Queue&);
```

```
        // Basic operations
        void enqueue (VoidPtr);
        VoidPtr dequeue ();
```

```
        // Other functions
        void print () const;
        int getVP (VoidPtr) const;
        bool isEmpty () const;
```

```
        // Other static function
        static int getNo ();
```

```
    private:
```

```
        struct Node;
        typedef Node* NodePtr;
```

```

    struct Node
    {
        VoidPtr data;
        NodePtr next;
    };

    NodePtr head, tail;

    // monitor the number of nodes created
    static int no;

    void testThrowException (const NodePtr&) const;
    void addToTail (VoidPtr);
    VoidPtr removeHead ();
};

#include "Queue.h"

        // default constructor
Queue::Queue ()
{
    // when a list is empty
    head = NULL;
    tail = NULL;
}

        // destructor
Queue::~~Queue ()
{
    NodePtr temp;

    while (head != NULL)
    {
        temp = head;
        head = head -> next;

        cout << "Node with value "
             << getVP (temp -> data);
    }
}

```

```

        delete temp;
        --no;
        cout << " was deleted" << endl;
    }

    // very important
    tail = NULL;
}

// copy constructor
Queue::Queue (const Queue& alist)
{
    head = NULL;
    tail = NULL;

    NodePtr temp = alist.head;

    while (temp != NULL)
    {
        addToTail (temp -> data);
        temp = temp -> next;
    }
}

void Queue::addToTail (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;

    if (tail == NULL)
        head = temp;
    else
        tail -> next = temp;

    tail = temp;
    ++no;
}

```

```

VoidPtr Queue::removeHead ()
{
    VoidPtr item;

    try
    {
        // This function's call replaces the
        // the following if statement
        testThrowException (head);
        // if (head == NULL)
        //     throw exception ();

        NodePtr temp = head;

        head = head -> next;

        if (head == NULL)
            tail = NULL;

        item = temp -> data;
        delete temp;
        --no;
    }
    catch (exception e)
    {
        cout << "Exception caught: "
              << "Removed head failed"
              << endl;
    }

    return item;
}

```

```

void Queue::print () const
{
    NodePtr temp = head;

    while (temp != NULL)

```

```

        {
            cout << getVP (temp -> data) << endl;
            temp = temp -> next;
        }
    }

int Queue::getVP (VoidPtr item) const
{
    int *temp = static_cast <int *> (item);
    return *temp;
}

bool Queue::isEmpty () const
{
    if (head == NULL && tail == NULL)
        return true;
    else
        return false;
}

int Queue::getNo ()
{
    return no;
}

int Queue::no = 0;

void Queue::testThrowException (const NodePtr& aPtr) const
{
    if (aPtr == NULL)
        throw exception ();
}

void Queue::enqueue (VoidPtr item)
{
    addToTail (item);
}

VoidPtr Queue::dequeue ()
{

```

```

        return removeHead ();
    }

#include "Queue.h"

int main ()
{
    Queue q;

    srand (time (NULL));

    int *temp;
    VoidPtr item;

    for (int i = 1; i <= 10; i++)
    {
        temp = new int;
        *temp = rand ();
        item = temp;

        q.enqueue (item);

        cout << "Enqueued item " << *temp << endl;
    }

    cout << "\nA test on dequeue function" << endl;

    while (!q.isEmpty ())
    {
        item = q.dequeue ();

        cout << "Item " << q.getVP (item) << " was dequeued" <<
endl;
    }

    cout << "\nTest on garbage collection" << endl;

}

```

```
Enqueued item 28596
Enqueued item 12287
Enqueued item 25039
Enqueued item 24121
Enqueued item 13722
Enqueued item 775
Enqueued item 13245
Enqueued item 23445
Enqueued item 26446
Enqueued item 26322

A test on dequeue function
Item 28596 was dequeued
Item 12287 was dequeued
Item 25039 was dequeued
Item 24121 was dequeued
Item 13722 was dequeued
Item 775 was dequeued
Item 13245 was dequeued
Item 23445 was dequeued
Item 26446 was dequeued
Item 26322 was dequeued

Test on garbage collection
Press Enter to return to Quincy...
```

// An interesting application, using a stack and a queue
// we can check if a positive integer is symmetric

```
#include "Stack.h"  
#include "Queue.h"
```

```
bool isSymmetric (int);
```

```
int main ()  
{  
    int n;  
    char ok;  
  
    do  
    {  
        cout << "Enter a positive integer: ";  
        cin >> n;  
  
        if (isSymmetric (n))  
            cout << n << " is symmetric" << endl;  
        else  
            cout << n << " is not symmetric" << endl;  
  
        cout << "Continue?: ";  
        cin >> ok;  
  
        cout << endl;  
    } while (ok == 'y' || ok == 'Y');  
}
```

```
bool isSymmetric (int n)
```

```
{  
    Stack s;  
    Queue q;  
  
    int *temp;  
    VoidPtr item;  
  
    while (n > 0)  
    {
```

```

        temp = new int;
        *temp = n % 10;
        item = temp;
        s.push (item);

        temp = new int;
        *temp = n % 10;
        item = temp;
        q.enqueue (item);

        n /= 10;
    }

    VoidPtr item1, item2;

    while (!s.isEmpty () && !q.isEmpty ())
    {
        item1 = s.pop ();
        item2 = q.dequeue ();

        if (s.getVP (item1) != q.getVP (item2))
            return false;
    }

    return true;
}

```

```

Enter a positive integer: 123321
123321 is symmetric
Continue?: y

Enter a positive integer: 12321
12321 is symmetric
Continue?: y

Enter a positive integer: 123421
123421 is not symmetric
Continue?: n

```

**// If we change the getVP to return a character, we can check if a word
// is symmetric using two linked lists**

```
#include "LinkedList.h"
```

```
bool isSymmetric (char *);
```

```
int main ()
{
    char word [20];
    char ok;

    do
    {
        cout << "Enter a word: ";
        cin >> word;

        if (isSymmetric (word))
            cout << word << " is symmetric" << endl;
        else
            cout << word << " is not symmetric" << endl;

        cout << "Wish to continue: ";
        cin >> ok;

        cout << endl;
    } while (ok == 'y' || ok == 'Y');
}
```

```
bool isSymmetric (char* word)
```

```
{
    LinkedList alist1, alist2;

    char *temp;
    VoidPtr item;

    char *p = &word [0];

    while (*p != '\0')
```

```

{
    temp = new char;
    *temp = *p;
    item = temp;
    alist1.addToHead (item);

    temp = new char;
    *temp = *p;
    item = temp;
    alist2.addToTail (item);
    ++p;
}

VoidPtr item1, item2;

while (!alist1.isEmpty () && !alist2.isEmpty ())
{
    item1 = alist1.removeHead ();
    item2 = alist2.removeHead ();
    // if you put tolower the following testings, you can
    // test for non-case sensitive
    if (alist1.getVP (item1) != alist2.getVP (item2))
        return false;
}

return true;
}

```

```

Enter a word: abba
abba is symmetric
Wish to continue: y

Enter a word: madam
madam is symmetric
Wish to continue: y

Enter a word: AnMA
AnMA is not symmetric
Wish to continue: n

```

**// A more general linked list class with two important functions: insert
// and remove. In fact this is an ordered linked list that we implement**

// A linear linked list with head and tail pointers

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <ctime>
```

```
using namespace std;
```

```
typedef void * VoidPtr;
```

```
class List
```

```
{
```

```
    public:
```

```
        // default constructor
```

```
        List ();
```

```
        // destructor
```

```
        ~List ();
```

```
        // copy constructor
```

```
        List (const List&);
```

```
        // - General insert
```

```
        void insert (VoidPtr);
```

```
        // - General remove
```

```
        bool remove (VoidPtr);
```

```
        // Other functions
```

```
        void print () const;
```

```
        int getVP (VoidPtr) const;
```

```
        bool isEmpty () const;
```

```
        // Other static function
```

```
        static int getNo ();
```

```
    private:
```

```
        struct Node;
```

```

typedef Node* NodePtr;

struct Node
{
    VoidPtr data;
    NodePtr next;
};

NodePtr head, tail;

// monitor the number of nodes created
static int no;

void testThrowException (const NodePtr&) const;

// Basic operations
void addToHead (VoidPtr);
void addToTail (VoidPtr);
VoidPtr removeHead ();
VoidPtr removeTail ();

int compare (VoidPtr, VoidPtr) const;
void findPosition (NodePtr&, NodePtr&, VoidPtr) const;
bool findNode (NodePtr&, NodePtr&, VoidPtr) const;

};

#include "List.h"

// default constructor
List::List ()
{
    // when a list is empty
    head = NULL;
    tail = NULL;
}

// destructor
List::~List ()

```

```

{
    NodePtr temp;

    while (head != NULL)
    {
        temp = head;
        head = head -> next;

        //      cout << "Node with value "
        //              << getVP (temp -> data);

        delete temp;
        --no;
        //      cout << " was deleted" << endl;
    }

    // very important
    tail = NULL;
}

```

```

        // copy constructor
List::List (const List& alist)
{
    head = NULL;
    tail = NULL;

    NodePtr temp = alist.head;

    while (temp != NULL)
    {
        addToTail (temp -> data);
        temp = temp -> next;
    }
}

```

```

void List::addToHead (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
}

```

```

    temp -> next = head;

    if (head == NULL)
        tail = temp;

    head = temp;
    ++no;
}

void List::addToTail (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;

    if (tail == NULL)
        head = temp;
    else
        tail -> next = temp;

    tail = temp;
    ++no;
}

VoidPtr List::removeHead ()
{
    VoidPtr item;

    try
    {
        // This function's call replaces the
        // the following if statement
        testThrowException (head);
        // if (head == NULL)
        //     throw exception ();

        NodePtr temp = head;

        head = head -> next;
    }
}

```

```

        if (head == NULL)
            tail = NULL;

        item = temp -> data;
        delete temp;
        --no;
    }
    catch (exception e)
    {
        cout << "Exception caught: "
              << "Removed head failed"
              << endl;
    }

    return item;
}

```

```

VoidPtr List::removeTail ()
{
    VoidPtr item;

    try
    {
        // This function's call replaces the
        // the following if statement
        testThrowException (tail);
        // if (tail == NULL)
        //     throw exception ();

        NodePtr temp = tail;

        // only one node
        if (tail == head)
        {
            tail = NULL;
            head = NULL;
        }
        else
        {
            NodePtr curr = head;

```

```

        // to get the node before the tail node
        while (curr -> next != tail)
            curr = curr -> next;

        curr -> next = NULL;
        tail = curr;
    }

    item = temp -> data;
    delete temp;
    --no;
}
catch (exception e)
{
    cout << "Exception caught: "
          << "Removed tail failed"
          << endl;
}

return item;
}

```

```

void List::print () const
{
    NodePtr temp = head;

    while (temp != NULL)
    {
        cout << getVP (temp -> data) << endl;
        temp = temp -> next;
    }
}

```

```

int List::getVP (VoidPtr item) const
{
    int *temp = static_cast <int *> (item);
    return *temp;
}

```

```

bool List::isEmpty () const
{
    if (head == NULL && tail == NULL)
        return true;
    else
        return false;
}

int List::getNo ()
{
    return no;
}

int List::no = 0;

void List::testThrowException (const NodePtr& aPtr) const
{
    if (aPtr == NULL)
        throw exception ();
}

void List::insert (VoidPtr item)
{
    NodePtr prev, curr;

    findPosition (prev, curr, item);

    if (prev == NULL)
        addToHead (item);
    else if (curr == NULL)
        addToTail (item);
    else
    {
        NodePtr temp = new Node;
        temp -> data = item;

        prev -> next = temp;
        temp -> next = curr;
    }
}

```

```

        ++no;
    }
}

bool List::remove (VoidPtr item)
{
    NodePtr prev, curr;

    bool found = findNode (prev, curr, item);

    if (found)
    {
        if (prev == NULL)
            removeHead ();
        else if (curr == tail)
            removeTail ();
        else
        {
            prev -> next = curr -> next;
            no--;
            delete curr;
        }
    }

    return found;
}

int List::compare (VoidPtr vp1, VoidPtr vp2) const
{
    if (getVP (vp1) > getVP (vp2))
        return 1;
    else if (getVP (vp1) < getVP (vp2))
        return -1;
    else
        return 0;
}

void List::findPosition (NodePtr& prev, NodePtr& curr, VoidPtr item) const
{

```

```

    prev = NULL;
    curr = head;

    while (curr != NULL && compare (item, curr -> data) > 0)
    {
        prev = curr;
        curr = curr -> next;
    }
}

bool List::findNode (NodePtr& prev, NodePtr& curr, VoidPtr item) const
{
    bool found = false;

    prev = NULL;
    curr = head;

    while (!found && curr != NULL)
    {
        int k = compare (item, curr -> data);

        if (k == 0)
            found = true;
        else
        {
            prev = curr;
            curr = curr -> next;
        }
    }

    return found;
}

```

```
#include "List.h"
```

```

int main ()
{
    List alist;

```

```

srand (time (NULL));

int *temp;
VoidPtr item;

for (int i = 1; i <= 10; i++)
{
    temp = new int;
    *temp = rand ();
    item = temp;

    alist.insert (item);

    cout << "Inserted " << *temp << endl;

}

temp = new int;
*temp = 0;
item = temp;
alist.insert (item);
cout << "Inserted " << *temp << endl;

temp = new int;
*temp = 12345;
item = temp;
alist.insert (item);
cout << "Inserted " << *temp << endl;

temp = new int;
*temp = 99999;
item = temp;
alist.insert (item);
cout << "Inserted " << *temp << endl;

cout << "-----" << endl;

```

```

cout << "The list is " << endl;
alist.print ();

cout << "-----" << endl;

    cout << "After the remove of 12345" << endl;
    temp = new int;
    *temp = 12345;
    item = temp;
    alist.remove (item);
    alist.print ();

    cout << "\nAfter the remove of 0" << endl;
    temp = new int;
    *temp = 0;
    item = temp;
    alist.remove (item);
    alist.print ();

    cout << "\nAfter the remove of 99999" << endl;
    temp = new int;
    *temp = 99999;
    item = temp;
    alist.remove (item);
    alist.print ();

}

```

```

Inserted 31502
Inserted 10112
Inserted 31672
Inserted 10198
Inserted 18654
Inserted 22157
Inserted 30815
Inserted 6843
Inserted 8930
Inserted 26798
Inserted 0
Inserted 12345
Inserted 99999
-----
The list is
0
6843
8930
10112
10198
12345
18654
22157
26798
30815
31502
31672
99999
-----

```

```

After the remove of 12345
0
6843
8930
10112
10198
18654
22157
26798
30815
31502
31672
99999

```

```

After the remove of 0
6843
8930
10112
10198
18654
22157
26798
30815
31502
31672
99999

```

```

After the remove of 99999
6843
8930
10112
10198
18654
22157
26798
30815
31502
31672

```

// What is a deque?

// - A deque is a double ended queue, i.e insertion

// and deletion can be done at both ends

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <ctime>
```

```
using namespace std;
```

```
typedef void* VoidPtr;
```

```
class Deque
```

```
{
```

```
    public:
```

```
        Deque ();
```

```
        void addLeft (VoidPtr);
```

```
        void addRight (VoidPtr);
```

```
        VoidPtr deleteLeft ();
```

```
        VoidPtr deleteRight ();
```

```
        void printLeftToRight () const;
```

```
        void printRightToLeft () const;
```

```
        bool isEmpty () const;
```

```
        int getVP (VoidPtr) const;
```

```
    private:
```

```
        struct Node;
```

```
        typedef Node* NodePtr;
```

```
        struct Node
```

```
        {
```

```
            VoidPtr data;
```

```
            NodePtr prev, next;
```

```
        };
```

```
        NodePtr left, right;
```

```
};
```

```

#include "Deque.h"

Deque::Deque ()
{
    left = NULL;
    right = NULL;
}

void Deque::addLeft (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> prev = NULL;
    temp -> next = left;

    if (left == NULL)
        right = left;
    else
        left -> prev = temp;

    left = temp;
}

void Deque::addRight (VoidPtr item)
{
    NodePtr temp = new Node;
    temp -> data = item;
    temp -> next = NULL;
    temp -> prev = right;

    if (right == NULL)
        left = temp;
    else
        right -> next = temp;

    right = temp;
}

```

```

VoidPtr Deque::deleteLeft ()
{
    VoidPtr item = NULL;

    try
    {
        if (left == NULL)
            throw exception ();

        NodePtr temp = left;

        left = left -> next;

        if (left == NULL)
            right = NULL;
        else
            left -> prev = NULL;

        item = temp -> data;
        delete temp;
    }
    catch (exception e)
    {
        cout << "Exception caught: "
              << "Delete left failed"
              << endl;
    }

    return item;
}

```

```

VoidPtr Deque::deleteRight ()
{
    VoidPtr item = NULL;

    try
    {
        if (right == NULL)
            throw exception ();
    }
}

```

```

        NodePtr temp = right;
        right = right -> prev;

        if (right == NULL)
            left = NULL;
        else
            right -> next = NULL;

        item = temp -> data;
        delete temp;
    }
    catch (exception e)
    {
        cout << "Exception caught: "
              << "Delete right failed"
              << endl;
    }

    return item;
}

void Deque::printLeftToRight () const
{
    NodePtr temp = left;

    while (temp != NULL)
    {
        cout << getVP (temp -> data) << "\t";
        temp = temp -> next;
    }

    cout << endl;
}

void Deque::printRightToLeft () const
{
    NodePtr temp = right;

    while (temp != NULL)
    {

```

```

        cout << getVP (temp -> data) << "\t";
        temp = temp -> prev;
    }

    cout << endl;
}

bool Deque::isEmpty () const
{
    return (left == NULL) && (right == NULL);
}

int Deque::getVP (VoidPtr item) const
{
    return *(static_cast <int *>(item));
}

#include "Deque.h"

int main ()
{
    Deque d;
    d.deleteLeft ();
    d.deleteRight();

    int *temp;
    VoidPtr item;

    srand (time (NULL));

    for (int i = 1; i <= 8; i++)
    {
        temp = new int;
        *temp = rand () % 100;
        item = temp;

        if (*temp % 2)
            d.addLeft (item);
        else
            d.addRight (item);
    }
}

```

```

    }

    d.printLeftToRight ();
//    d.printRightToLeft ();

/*
    while (!d.isEmpty ())
    {
        cout << "Node with value ";
        cout << d.getVP (d.deleteLeft ());
        cout << " deleted" << endl;
    }
*/
    while (!d.isEmpty ())
    {
        cout << "Node with value ";
        cout << d.getVP (d.deleteRight ());
        cout << " deleted" << endl;
    }
}

```

```

// Circular linked list
// A circular linked list is pointed by a pointer
// called ptr, which points to the rightmost node of the
// list
//
// A stack can be implemented using circularly linked list
//
// An application: using stack, we can convert a positive
// integer to various conversions equivalent (binary, octal or hexa)

```

```

#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

class Stack
{
    public:
        Stack ();

        // two stack operations
        void push (char);
        char pop ();

        // Other operation
        bool isEmpty () const;

    private:
        struct Node;
        typedef Node* NodePtr;

        struct Node
        {
            char data;
            NodePtr next;
        };

        NodePtr ptr;

```

```

        void emptyStackException () const;
};

// Using stack operations, display
// a positive integer to in some specific
// base (2, 8, 16)
void display (int, int);

int main ()
{
    int n;

    srand (time (NULL));

    for (int i = 1; i <= 5; i++)
    {
        n = rand ();

        display (n, 2);
        display (n, 8);
        display (n, 16);

        cout << "-----" << endl;
    }
}

Stack::Stack ()
{
    ptr = NULL;
}

// two stack operations
void Stack::push (char item)
{
    NodePtr temp = new Node;
    temp -> data = item;

    if (ptr == NULL)
        temp -> next = temp;
}

```

```

        else
        {
            temp -> next = ptr -> next;
            ptr -> next = temp;
        }

        ptr = temp;
    }

char Stack::pop ()
{
    char item;

    try
    {
        emptyStackException ();

        NodePtr temp = ptr;

        if (ptr == ptr -> next)
            ptr = NULL;
        else
        {
            NodePtr leftNode = ptr -> next;

            NodePtr curr = ptr;

            while (curr -> next != ptr)
                curr = curr -> next;

            curr -> next = leftNode;
            ptr = curr;
        }

        item = temp -> data;
        delete temp;

    }
    catch (exception e)
    {

```

```

        cout << "Stack empty exception caught" << endl;
    }

    return item;
}

// Other operation
bool Stack::isEmpty () const
{
    return (ptr == NULL);
}

void Stack::emptyStackException () const
{
    if (ptr == NULL)
        throw exception ();
}

void display (int n, int base)
{
    char hexa [] = "0123456789ABCDEF";

    cout << "\nRepresentaion of " << n << " in base " << base << endl;

    Stack s;

    while (n > 0)
    {
        s.push(hexa[n%base]);
        n /= base;
    }

    while(!s.isEmpty())
    {
        cout << s.pop();
    }
    cout << endl;
}

```

```
Representaion of 92 in base 2
1011100

Representaion of 92 in base 8
134

Representaion of 92 in base 16
5C
-----

Representaion of 25001 in base 2
1100000110101001

Representaion of 25001 in base 8
60651

Representaion of 25001 in base 16
61A9
-----

Representaion of 24831 in base 2
1100000011111111

Representaion of 24831 in base 8
60377

Representaion of 24831 in base 16
60FF
-----
```

The use of Hash function

**// Generate k distinct integers and insert them into an array
// of size n using the hash function integer % n,
// k is smaller than n**

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
const int MAX = 11;
const int K = 7;
```

```
int hashFunction (int givenInteger, int sizeOfArray);
void initialize (int [], int);
void insertTable (int hashTable [], int size, int anInteger);
void printTable (const int [], int);
```

```
bool findNo (const int [], int, int);
```

```
int main ()
{
    int hashTable [MAX];
    int n;

    srand (time (NULL));

    initialize (hashTable, MAX);

    for (int i = 1; i <= K; i++)
    {
        n = rand ();

        cout << "Insert " << n << endl;
        insertTable (hashTable, MAX, n);
    }

    cout << endl;
```

```

        printTable (hashTable, MAX);

    }

int hashFunction (int givenInteger, int sizeOfArray)
{
    return givenInteger % sizeOfArray;
}

void initialize (int table [], int size)
{
    for (int i = 0; i < size; i++)
        table [i] = -999;
}

void insertTable (int hashTable [], int size, int anInteger)
{
    int k = hashFunction (anInteger, size);
    bool inserted = false;

    while (!inserted)
    {
        if (hashTable [k] == -999)
        {
            hashTable [k] = anInteger;
            inserted = true;
        }
        else
        {
            ++k;
            k %= size;
        }
    }
}

```

```

bool findNo (const int table [], int size, int aNo)
{
    bool found = false;
    bool endCheck = false;
    int k = aNo % size

    while (!found && !endCheck)
    {
        if (table [k] == aNo)
            found = true;
        else if (table [k] == -999)
            endCheck = true;
        else
        {
            ++k;
            k %= size;
        }
    }

    return found;
}

```

```

void printTable (const int table [], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << i << "\t";
        cout << table [i] << "\t";

        if (table [i] != -999)
            cout << table [i] % size;

        cout << endl;
    }
}

```

// A binary search tree (BST)

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

typedef void* VoidPtr;

class BST
{
    public:
        BST ();
        ~BST ();
        BST (const BST&);

        void insert (VoidPtr);
        void print () const;

        bool findNode (VoidPtr) const;

    private:
        struct Node;
        typedef Node* NodePtr;

        struct Node
        {
            VoidPtr data;
            NodePtr left, right;
        };

        NodePtr root;

        // Some more recursive functions
        void insert (NodePtr&, VoidPtr);
        bool findNode (NodePtr, VoidPtr) const;
        void inorderPrint (NodePtr) const;
        void postorderDelete (NodePtr&);
```

```

        void preorderInsert (const NodePtr&);

        int compare (VoidPtr, VoidPtr) const;
        int getVP (VoidPtr) const;
};

```

```

// A binary search tree (BST)

```

```

#include <iostream>

```

```

#include <ctime>

```

```

#include <cstdlib>

```

```

using namespace std;

```

```

typedef void* VoidPtr;

```

```

class BST

```

```

{

```

```

    public:

```

```

        BST ();

```

```

        ~BST ();

```

```

        BST (const BST&);

```

```

        void insert (VoidPtr);

```

```

        void print () const;

```

```

        bool findNode (VoidPtr) const;

```

```

    private:

```

```

        struct Node;

```

```

        typedef Node* NodePtr;

```

```

        struct Node

```

```

        {

```

```

            VoidPtr data;

```

```

            NodePtr left, right;

```

```

        };

```

```

        NodePtr root;

```

```

// Some more recursive functions
void insert (NodePtr&, VoidPtr);
bool findNode (NodePtr, VoidPtr) const;
void inorderPrint (NodePtr) const;
void postorderDelete (NodePtr&);
void preorderInsert (const NodePtr&);

int compare (VoidPtr, VoidPtr) const;
int getVP (VoidPtr) const;
};

```

```

#include "BST.h"

```

```

int main ()
{
    BST t;

    srand (time (NULL));
    int *temp;
    VoidPtr item;

    for (int i = 1; i <= 10; i++)
    {
        temp = new int;

        do
        {
            *temp = rand ();
            item = temp;
        } while (t.findNode (item) == true);

        t.insert (item);
    }

    t.print ();

    cout << "-----" << endl;
}

```

```
    cout << "Duplicate another tree" << endl;

    BST tt (t);
    tt.print ();

}
```

// Generate some uppercase letters (may be duplicated)
// and sort them in order using the BST concept

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

class BST
{
    public:
        BST ();

        void insert (char);
        void print () const;

        void preorderUpdate (int);

    private:
        struct Node;
        typedef Node* NodePtr;

        struct Node
        {
            char data;
            int count;
            NodePtr left, right;
        };

        NodePtr root;

        void insert (NodePtr&, char);
        void inorderPrint (NodePtr) const;

        void preorderUpdate (NodePtr&, int);
};

/* -----*/
```

```

BST::BST ()
{
    root = NULL;
}

void BST::insert (char item)
{
    insert (root, item);
}

void BST::print () const
{
    inorderPrint (root);
}

void BST::insert (NodePtr& root, char item)
{
    if (root == NULL)
    {
        NodePtr temp = new Node;
        temp -> data = item;
        temp -> count = 1;
        temp -> left = NULL;
        temp -> right = NULL;

        root = temp;
    }
    else if (root -> data == item)
        root -> count += 1;
    else if (root -> data > item)
        insert (root -> left, item);
    else
        insert (root -> right, item);
}

void BST::inorderPrint (NodePtr root) const
{
    if (root != NULL)

```

```

    {
        inorderPrint (root -> left);

        /*
        for (int i = 1; i <= root -> count; i++)
            cout << root -> data << " ";
        */

        cout << root -> data << " occurred "
            << root -> count
            << " times"
            << endl;

        inorderPrint (root -> right);
    }
}

```

```

void BST::preorderUpdate (int k)
{
    preorderUpdate (root, k);
}

```

```

void BST::preorderUpdate (NodePtr& root, int k)
{
    if (root != NULL)
    {
        root -> count += k;

        preorderUpdate (root -> left, k);
        preorderUpdate (root -> right, k);
    }
}

```

```

/* ----- */

```

```

int main ()

```

```

{
    BST t;

    srand (time (NULL));

    char item;

    for (int i = 1; i <= 1000; i++)
    {
        item = static_cast <char> (rand () % 26 + 65);

        t.insert (item);
    }

    t.print ();

    cout << "-----" << endl;

    cout << "Add 1000 to all counts" << endl;

    t.preorderUpdate (1000);
    t.print ();
}

```

A	occurred	42	times
B	occurred	47	times
C	occurred	35	times
D	occurred	43	times
E	occurred	37	times
F	occurred	32	times
G	occurred	41	times
H	occurred	33	times
I	occurred	48	times
J	occurred	35	times
K	occurred	38	times
L	occurred	35	times
M	occurred	34	times
N	occurred	46	times
O	occurred	37	times
P	occurred	35	times
Q	occurred	40	times
R	occurred	34	times
S	occurred	30	times
T	occurred	41	times
U	occurred	41	times
V	occurred	44	times
W	occurred	39	times
X	occurred	40	times
Y	occurred	34	times
Z	occurred	39	times

```
Add 1000 to all counts
A occurred 1042 times
B occurred 1047 times
C occurred 1035 times
D occurred 1043 times
E occurred 1037 times
F occurred 1032 times
G occurred 1041 times
H occurred 1033 times
I occurred 1048 times
J occurred 1035 times
K occurred 1038 times
L occurred 1035 times
M occurred 1034 times
N occurred 1046 times
O occurred 1037 times
P occurred 1035 times
Q occurred 1040 times
R occurred 1034 times
S occurred 1030 times
T occurred 1041 times
U occurred 1041 times
V occurred 1044 times
W occurred 1039 times
X occurred 1040 times
Y occurred 1034 times
Z occurred 1039 times
```

Some sample questions based on lecture notes and class examples applied to some applications

The following 3 .cpp files and 2 .h files that make up our program:

```
main1.cpp  (depends on openfile.h and mylib.h)
mylib.cpp  (depends on mylib.h)
mylib.h
openfile.cpp (depends on opefile.h )
openfile.h
```

In the Linux environment, you can design a makefile to execute the whole project.

- (a) Suggest what to put in the makefile called Exam.make, to produce an executable Exam.out. Your suggestion should also allow you to remove (clean) the old object files and Exam.out in case modifications are made in the above files.

(2.5 marks)

- (b) Write the command to execute Exam.make so that Exam.out is created?
(0.5 mark)

// Answer for part (a)

```
Exam.out: main.o mylib.o openfile.o
    g++ -o lab1.out main1.o mylib.o openfile.o
main1.o: main.cpp openfile.h mylib.h
    g++ -c main1.cpp
mylib.o: mylib.cpp mylib.h
    g++ -c mylib.cpp
openfile.o: openfile.cpp openfile.h
    g++ -c openfile.cpp
clean:
    rm *.o lab1.out
# End of make file
```

// Answer to part (b)

```
$ make -f Exam.make
$ ./Exam.out
```

You are given a binary file (file name is “infile.dat”) consisting of some of the following record structures:

```
struct Record
{
    // a few member fields here
};
```

Write C++ code fragment to check how many records are there in that file.

```
ifstream infile;
infile.open (“infile.dat”, ios::in | ios::binary);
Record temp;
infile.seekg (0, ios::end);
int noOfBytes = infile.tellg ();
int noOfRecords = noOfBytes / sizeof (Record);
```

The exam marks for CSCI124 were too low. The board of examiners suggested to me to add 10 marks to those students who scored less than 50 marks and to add 5 marks to those students who scored 50 marks and above. I usually stored the results of all students in a binary file (called “final.dat”) according to following struct definition:

```
struct Student
{
    char name [30];
    float labTest;
    float assignments;
    float exam;
};
```

Design a function to help me to moderate the exam marks based on the above proposed scheme. The function has the following prototype:

```
void updateExam (fstream&, char [ ]);
```

```
void updateExam (fstream& afile, char filename [ ])
{
    afile.open (filename, ios::binary | ios::in | ios::out);

    if (!afile.good ())
    {
        cout << “File opening error” << endl;
        return;
    }

    afile.seekg (0, ios::end);
    int noOfBytes = afile.tellg ();
    int noOfStudents = noOfBytes / sizeof (Student);

    Student s;
    for (int index = 1; index <= noOfStudents; index++)
    {
        afile.seekg ( (index - 1) * sizeof (Student), ios::beg);
        afile.read (reinterpret_cast <char *>(&s), sizeof (s));

        if (s.exam < 50)
            s.exam += 10;
        else
            s.exam += 5;
    }
}
```

```
        afile.seekp ( (index - 1) * sizeof (Student), ios::beg);
        afile.write (reinterpret_cast <const char *>(&s), sizeof (s));
    }

    afile.close ();
}
```

Given the following function definition:

```
char * getFruit (int k)
{
    char fruit [ ][MAX] = {"Apple", "Orange", "Durian", "Papaya", "Peach",
    "Longan"};
    return fruit [k];
}
```

You can assume that **MAX** is an integer constant and validation of **k** is done outside the function and is in range. Write comments to the above function definition and suggest how to amend the function.

The allocation for variable **fruit** is automatic. When exiting from the function, the storages becoming garbage. Two possible amendments to the function:

- (1) place a qualifier **static** before the array's definition; or
- (2) Change the return statement to:

```
char *aFruit = new char [MAX];
strcpy (aFruit, fruit [k]);
return aFruit;
```

A linear linked is constructed based on the following node structure:

```
struct Node;  
typedef Node* NodePtr;  
struct Node  
{  
    void* data;  
    NodePtr next;  
};
```

- (a) Design a function indirectly checks whether the two English words, represented by two C-string, are equal. The function has the following prototype:

```
bool checkEqual (void*, void*);
```

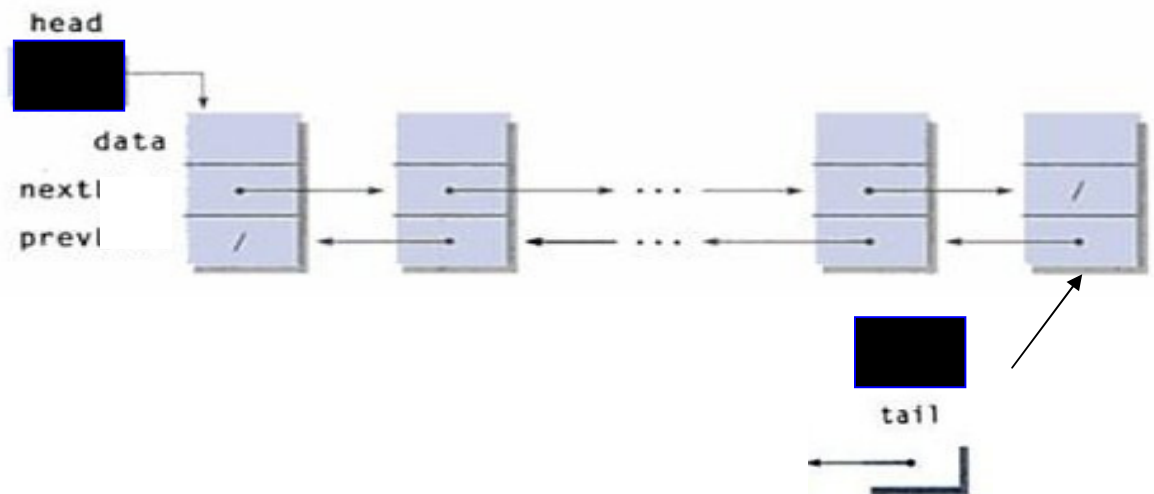
- (b) Design a function to check if a data item is inside the linear linked list. You can assume that information stored inside the list are some English words. If the data item is in, returns the reference of that node; NULL otherwise. that The function has the following prototype:

```
NodePtr findInfo (const NodePtr&, void*);
```

```
bool checkEqual (void *vp1, void *vp2)  
{  
    char str1 = static_cast <char *> (vp1);  
    char str2 = static_cast <char *> (vp2);  
    return (strcmp (str1, str2) == 0);  
}
```

```
NodePtr findInfo (const NodePtr& head, void* item)  
{  
    NodePtr curr = head;  
    While (curr != NULL && !checkEqual (curr -> data, item))  
        curr = curr -> next;  
  
    return curr;  
}
```

Given a doubly linked list as shown below. We can traverse the list from **head** to **tail** and also from **tail** to **head**. As can be seen that a queue is more suitable implemented using a double linked list.



We define the following node structure definition for the above double linked list:

```
struct Node;
typedef Node* NodePtr;
struct Node
{
    void* data;
    NodePtr next, prev;
};
```

Design a function to print out the list from the tail to the head. The function has the following prototype: (You can assume that you print out some integers)

```
void printList (const NodePtr&);
```

```
void printList (const NodePtr& tail)
{
    NodePtr temp = tail;
    while (tail != NULL)
    {
        cout << *(static_cast <int *>(tail -> data)) << "\t";
        tail = tail -> prev;
    }

    cout << endl;
}
```

The following class definition describe the binary representation of a positive integer n . The data structure used for storing the representation is a stack (of integers). You can assume that an ADT of stack is given to you (see Section E Question 4), and simply replaced the `void*` by `int` type.

```
class BinaryNumber
{
    public:
        BinaryNumber (int);

        void printBinary ();           // just print out the stack

    private:
        Stack s;                      // a stack to store the representation
        int n;                        // the number to be converted to binary
        void convertBinary ();
};
```

(a) The function `convertBinary` converts the number, n , to binary number and stores the result in the stack. Full mark will be allocated to this question if you explore bitwise operations to perform the conversion.

(b) Implement the `printBinary` function.

```
void BinaryNumber::convertBinary ()
{
    int aBit;
    while (n > 0)
    {
        abit = n % 2;
        s.push (abit);
        n /= 2;
    }
}

void BinaryNumber::printBinary ()
{
    while (!s.isEmpty ())
        cout << s.pop;
    cout << endl;
}
```

Using radix sort, sort the following integers in decreasing order: using () to denote an empty partition

7483 5289 4947 6448 2123 3987 8236 4180

(5289) (6448) (2987 4947) (8236) () () (7483 2123) () () (4120)
5289 6448 2987 4947 8236 2483 2123 4120

() (5289 2987 2483) () () () (6448 4947) (8236) (2123 4120) () ()
5289 2987 2483 6448 4947 8236 2123 4120

(2987 4947) () () () () (2483 6448) () (5289 8236) (2123 4120)
2987 4947 2483 6448 5289 5236 2123 4120

() () () (6448) (5289 5236) (4947 4120) () (2987 2483 2123)
6448 5289 5236 4947 4120 2987 2483 2123

Almost all the sorting and searching examples you have learnt in class dealing with integer arrays. The following is the selection sort given in class:

```
void selectionSort (int X [ ], int n)
{
    int i, j, smallest, temp;

    for (i = 0; i < n -1; i++)
    {
        smallest = i;
        for (j = i+1; j < n; j++)
        {
            if (X [smallest] > X [ j ])
                smallest = j;
        }

        if (smallest != i)
            swap (x [i], x [smallest]);
    }
}
```

Note that swap is a given function with prototype

```
void swap (int&, int&);
```

Suggest how to amend the above function to the following prototype:

```
void selectionSort (char **word, int size);
```

To sort an array of English words, use them in the design

```
int compare (char *word1, char *word2)
{
    return strcmp (word1, word2);
}

void swap (char* word1, char* word2)
{
    char *temp;

    strcpy (temp, word1);
    strcpy (word1, word2);
    strcpy (word2, temp);
}
```

The following class describes a BST that you with to insert a list of distinct English words. Each word is represented by a cstring.:

(5 marks)

```
class BST
{
    public:
        BST ();
        ~BST ();

        // Insert a word
        void insert (char *);

    private:

        struct Node;
        typedef Node* NodePtr;

        struct Node
        {
            char *word;
            NodePtr left, right;
        };

        NodePtr root;
};
```

Implement this class, i.e. the constructor, the destructor and the insert function.

```
BST::BST ()
{
    root = NULL;
}

BST::~~BST ()
{
    postorderDelete (root);
}

BST::insert (char *w)
{
    insertNode (root, w);
}
```

```

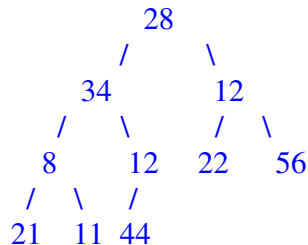
void BST::insertNode (NodePtr& root, char *item)
{
    if (root == NULL)
    {
        root = new Node;
        strcpy (root -> data, item);
        root -> left = NULL;
        root -> right = NULL;
    }
    else if (strcmp (root -> data, item) > 0)
        insertNode (root -> left, item);
    else
        insertNode (root -> right, item);
}

```

Show how to use heap sort to sort the following set of keys:

28 34 12 8 12 22 56 21 11 44

Step 1: Draw a binary tree



Step 2: Construct a heap tree



All the best

to your

future study

