```
//Thomas Gibbons
//Oct 27, 2016
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
//-------------------------------------------------------------
string morseChart[]=  { ".-","-...","-.-.","-..",".","..-.",
                        "--.","....","..",".---","-.-","-..",
                        "--","-.","---",".--.","--.-",".-.",
                        "...","-","..-","...-",".--","-..-",
                        "-.--","--..","?"};
//-------------------------------------------------------------
class message{
    private:

    protected:
        string english;
    public:
        message();
        message(string);
        ~message();
        virtual void printInfo();
};

message::message(){
    //std::cout <<  "\nObject of class message created";
    std::cout   <<  "English message to display: ";
    std::cin    >>  english;
}

message::message(string word){
    //std::cout <<  "\nObject of class message created";
    english=word;
}

message::~message(){
    //nothing needs to be freed
}

void message::printInfo(){
    std::cout   <<  "\nEnglish message: "    << english;
}

class morseCodeMessage:public message{
    private:
        string *morse;
        void translate();
        int index;
    public:
        morseCodeMessage();
        morseCodeMessage(string);
        ~morseCodeMessage();
```

```
        morseCodeMessage* next;
        void printInfo();       //new standard
};

morseCodeMessage::morseCodeMessage(string x):message(x){
    translate();
    next=NULL;
}

morseCodeMessage::morseCodeMessage():message(){
    translate();
    next=NULL;
}

morseCodeMessage::~morseCodeMessage(){
    //cout << "\nGoodbye Message: ";
    delete morse;
}

void morseCodeMessage::printInfo(){
    cout    <<  english <<  " = ";  //same as in message class
    int count=english.length();     //length of word
    int x=0;

    while(x<count)
    {
        cout    << morse[x] <<  " ";    //prints out each morse string letter by letter
        x++;
    }
    cout << endl;
}

void morseCodeMessage::translate(){
    int count=english.length(); //get length of word
    int x=0;
    morse=new string[count];    //make string for each letter of word

    while(x<count)
    {
        if (isupper(english[x]))
            english[x]=tolower(english[x]);     //convert everything to lower case
        index = english[x] - 'a';   //Use ascii character 'a' to center indexes from 0 to 25
        if (index<0 || index>25)
            index=26;
        morse[x]=morseChart[index];     //find corresponding morse string for letter
        x++;
    }
}

class messageStack{
    private:

    public:
```

```
        morseCodeMessage* stack_top;
        messageStack();
        messageStack(morseCodeMessage current_obj);
        void push(morseCodeMessage* current_obj);
        morseCodeMessage pop();
        void printStack();
        int numObjects;
};

messageStack::messageStack(){
    //initialize stack
    numObjects=0;
    stack_top=NULL;
}

messageStack::messageStack(morseCodeMessage current_obj){
    //initialize stack with first object
    numObjects=1;
    stack_top=&current_obj;
}

void messageStack::push(morseCodeMessage* current_obj){
    (*current_obj).next=stack_top;  //link two objects
    stack_top=current_obj;  //keep track of top of linked list
    numObjects++;   //keep track of objects linked
}

morseCodeMessage messageStack::pop(){
    morseCodeMessage* hold=stack_top;   //hold top to return
    stack_top=(*stack_top).next;    //change top to new top
    numObjects--;   //one less object in linked list
    return *hold;   //return old top for print
}

void messageStack::printStack(){
    while(numObjects>0){
        pop().printInfo();  //get the message on top and print
    }
}

int main(int argc, char** argv){
    //make sure their are arguments other than program name
    if(argc==1){
        cout << "\nError: Input arguments to be converted to morse code";
        return 0;
    }

    //push arguments to stack except program name
    int count=argc;
    morseCodeMessage** hold= new morseCodeMessage*[argc];
    messageStack order;
    while(count>1){
        hold[count-1]=new morseCodeMessage(argv[count-1]);
```

```
        //(*hold).printInfo();
        order.push(hold[count-1]);
        count--;
    }

    //print stack
    order.printStack();

    //free allocated objects
    //{
    count=argc;
    while(count>1){
        delete hold[count-1];
        count--;
    }

    //free allocated pointer of array of objects
    delete[] hold;

    //}
    //end successfully
    return 1;
}
```