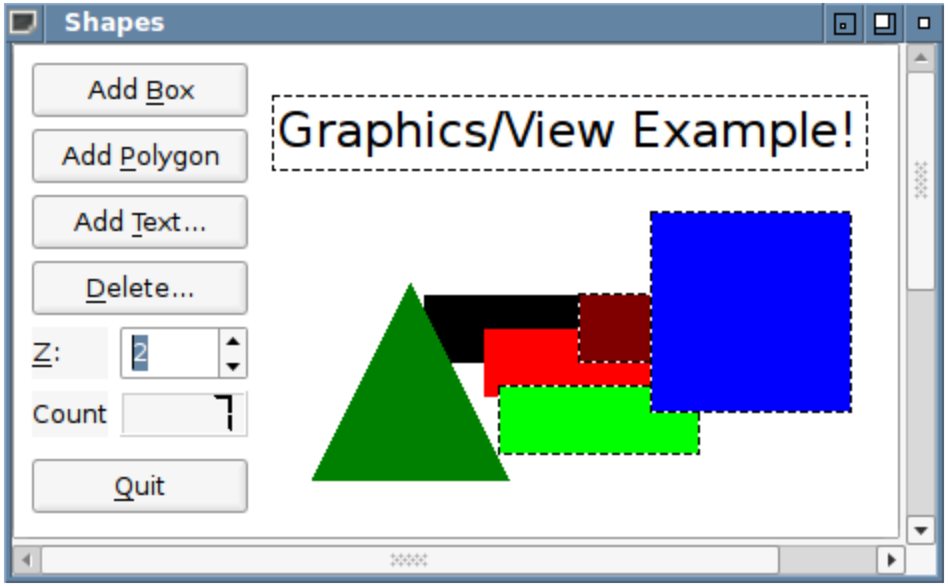
C++ and OO

- C++ classes and OO
- More Examples
- HW2

Objects


Shapes
Add Box
Add Polygon
Add Text...
Delete...
Z:
2
Count
7
Quit
Graphics/View Example!

C++ and OO

- What happens with a declaration
  - int i , j = 3;
  - Declares; allocates ; initializes
  - For a simple native type this happens automatically via a stack
- Constructor – the OO function to do this

Quiz- what is printed

- int main()
- {   int i = 9, j = 3;
-       cout << “i is “ << i <<“ j is “ << j <<endl;
-      { int i = j + 2;
-       cout << “i is “ << i <<“ j is “ << j <<endl;   }
-       cout << “i is “ << i <<“ j is “ << j <<endl;
- }

Answer

- i is 9 j is 3
- i is 5 j is 3
- i is 9 j is 3

Point and its constructor

```
class point{
public:
    point(x=0.0, y = 0.0):x(x),y(y){}  //constructor
    double getx(){return x;}
    void  setx(double val){x = v;}
…..
private:
   double x, y;
};

//note class_name():initializer list syntax
```

This pointer --self-referential

- Every class object has the implicit pointer
  - Keyword this associate with it.

- We will use it in the next slide

A special method -constructor

- point(){ x = y = 0.0;}

- or
- point(){ this -> x = 0.0; this ->y = 0.0}
- Or best
- point():x(0.0),y(0.0){}

- Default constructor – the constructor whose signature is void.

Use

- point p, q, r; // all call constructor of void signature

Constructor overloading

- It is useful to have multiple ways to initialize an object like point.

- point(double x, double y)
- {this -> x = x; this ->y = y;}

- this used to disambiguate

Argument and member confusion

- point(double x, double y)
- {this -> x = x; this ->y = y;}

  - This lets ambiguity be resolved x=x; would not work
  - Better use initialization syntax
  - point(double x, double y):x(x),y(y){}

More Constructors

Constructors: Initialize

Constructors: Convert

Constructors: Allocate

Also: constructors can check for correctness

Memory management

- new - allocator -think malloc()
- delete – deallocator – think free()

- Both work with a heap – heap is dynamically allocated memory – unlike Java not automatically garbage collected

Simple use of new

- char* s = new char[size];//get off heap
- int *p = new int(9); // single int initialized
- delete [] s; //delete an array
- delete p; //delete single element

  - These will get used with dynamic data structures in constructors and destructors

~ destructor

- Deallocator when item goes out of scope
- Syntax within class ~classname(){ …}
- Typical use is for calling delete to deallocate to the heap – what if you forget
- Answer: Bad manners -memory leak

Linked List –p 168 section5.7

- struct slistelem{ char data; slistelem* next;}
- class slist{ //singly linked list
- public:
-    slist():h(0){}   //empty list
-   ~slist() {release();} destructor
- …..more methods
- private:
-    slistelem* h;   //list head
- }

Prepend to slist

- void slist::prepend (char* c)
- {
-     slistelem* temp = new slistelem;
-     assert (temp != 0);
-     temp -> next = h; //single link
-     temp -> data = c;
-     h = temp; //update h
- }

~ destructor

- slist::~slist()
- {
- cout << “destructor called” << endl;
- //just for demonstration –debug
- release(); //march thru list with
- //deletes
- }

HW2 – some ideas

- HW2 : implement Dijkstra algorithm and use a randomly generated graph to test it.
- A simpler problem is to compute if a graph is one connected component
- Draw Unconnected graph

Unconnected graph

- O
- O O O
- O O O

Connected graph

- O
- O O O
- O O O

First draw a randomly generated Graph

- bool** graph;
- srand(time(0));//seed rand()
- graph = new bool*[size];
- for(int i = 0; i <size; ++i)
- graph[i] = new bool[size];
- //heap created 2 D array of Bool

Density is 0.19

```
for (int i = 0; i < size; ++i)
        for(int j = i; j < size; ++j)
                if (i == j) graph[i][j]= false;  //no loops
                        else   graph[i][j] = graph[j][i] = (prob() < 0.19);
```


UNIVERSITY OF CALIFORNIA SANTA CRUZ

Quiz:

- If the density is 0 the graph has no _____
- If the density is 1 the graph is c……..
- If the density is fixed say 0.1 then as the size of the graph gets larger it is _____likely to be connected.

Answer:

- If the density is 0 the graph has no edges
- If the density is 1 the graph is complete
- If the density is fixed say 0.1 then as the size of the graph gets larger it is more likely to be connected.

The is_connected algorithm

- //This algorithm is a form of breadth first search - first implemented by the author in 1968 at SLAC.
- //It was in PL1 and was a package of routines for computational graph theory.
- //See also the Wikipedia on Breadth First Search.
- //The algorithm is_connected uses a Boolean matrix representation of an undirected graph to determine if a graph is connected.

Details

- It starts with node 0 and determines which nodes can be reached from this node
- placing them in an open set and placing node 0 as the first node of a connected component.
- Each iteration adds one node to the closed set.
- This stops with either no furter nodes reachable and is_connected is false or all nodes being included in the closed set.
- The algorithm was published as a SLAC report and later a generalizion was published by Hopcroft and Tarjan in 1973.

Is_connected

```
bool is_connected(bool *graph[], int size)
{

        int old_size =0, c_size = 0;
        bool* close = new bool[size];
        bool* open = new bool[size];
        for(int i = 0; i < size; ++i)
            open[i] = close[i] = false;
        open[0] = true;
```

At this point

- Open set has node 0 on it
- Question would this work if other node is selected

- Nothing in closed set.

- Each iteration will add one node to closed set

Add to close

- while(c_size < size){
- for (int i = 0; i < size; ++i){
- old_size = c_size;
- if (open[i] && (close[i] == false)){
- close[i] = true; c_size++;

Add to open

- for(int j = 0; j < size; ++j)
- open[j] = open[j] || graph[i][j];
- }
- }

Are we done?

- We are done if have all nodes in close set
- Or if no nodes available in open set

```
if (c_size == size) return true;
        if (old_size == c_size) return false;
                        }

                    }
```

L6 – Lists

- List and code

List Element

- ```
  struct list_element{
     list_element(int n = 0, list_element* ptr = 0):
         d(n), next(ptr){}
     int  d;
     list_element* next;
  };
  ```

Or   equivalently

- ```
  class list_element{
  public:
  ```
- ```
     list_element(int n = 0, list_element* ptr = 0):
         d(n), next(ptr){}
     int  d;
     list_element* next;
  };
  ```
  ```

Quiz

- In the previous list_element constructor, what is 0 used for?

Ans: Null Pointer

- The zero value is the null pointer value.

- Recall it is important in lists to test for null; they are used as sentinel values.

- C++11 - list_element* ptr =nullptr ;
- new keyword - type safe

List

- ```
  class list{
  public:
     list():head(0), cursor(0){}
     void prepend(int n); //insert at front value n
     int get_element(){return cursor->d;}
     void advance(){ cursor= cursor-> next;}
     void print();
  private:
     list_element* head;
     list_element* cursor;
  };
  ```
  ```

prepend

- void list::prepend(int n)
```
{ if (head == 0)//empty list case
    cursor = head = new list_element(n,
head);
   else//add to front -chain
    head = new list_element(n, head);
}
```

Quiz : prepend(5)

- Draw how prepend (5) would work.

- Assume a two element list   -> 7 -> 3 ##

Answer

- 5 is on the front of the new list
- Draw here----

Print() chaining

- Void list:: print(){
```
    list_element* h = head;
    while(h != 0){//idiom for chaining
      cout << h->d  << ", ";
      h = h -> next;
    }
    cout << "###" << endl;
  }
```

- Should know how to use recursion
- Should know how to overload “<<“ for list

Use of list

- ```
  int main()
  {
    list a, b;
    a.prepend(9); a.prepend(8);
    cout << " list a " << endl;
    a.print();
    for (int i = 0; i < 40; ++i)
      b.prepend(i*i);
    cout << " list b " << endl;
    b.print();
  }
  ```

- What gets printed?
  ```

Q: What gets printed

- Follow previous code

Ans:

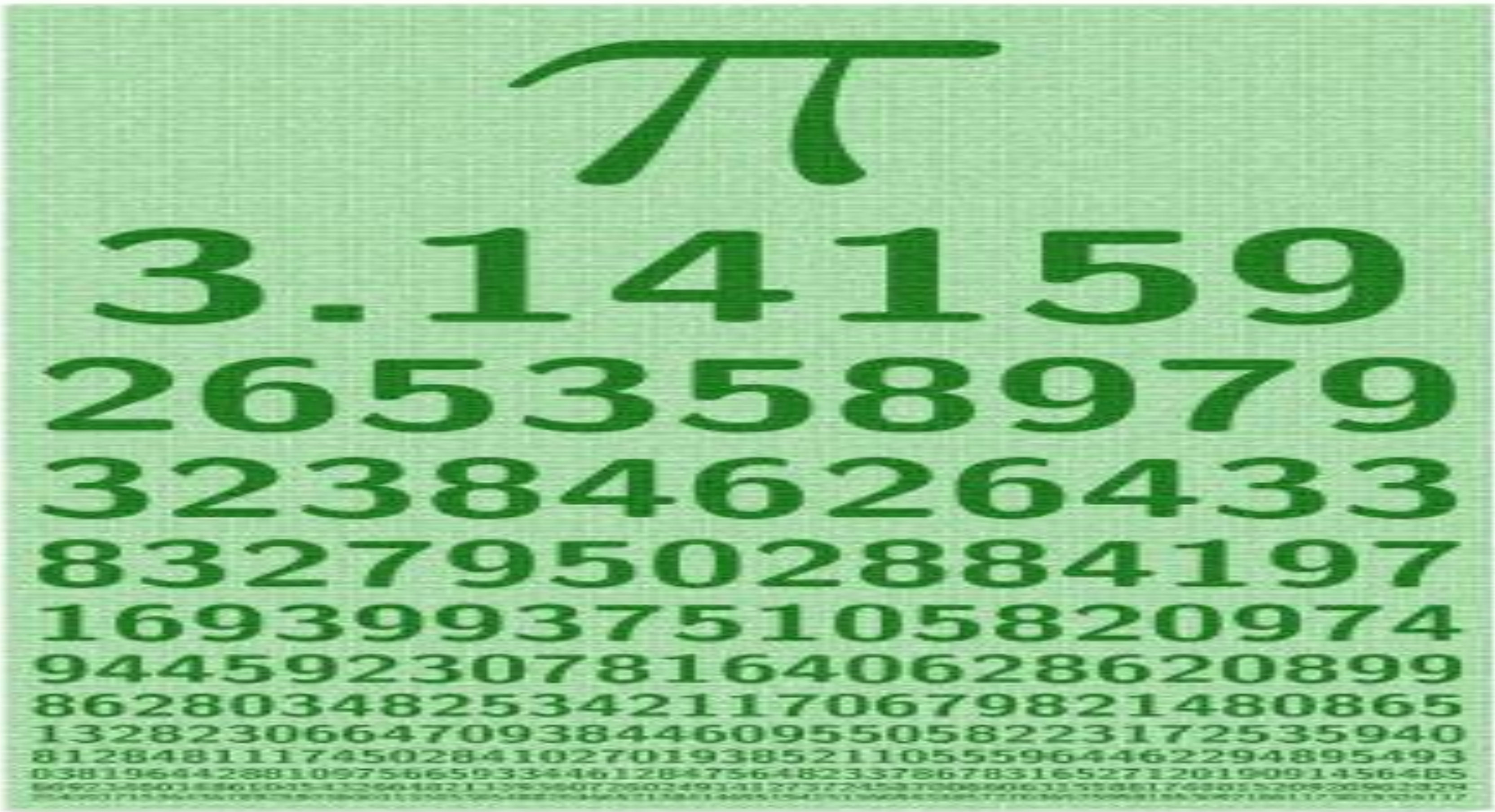
- Simulate here:(run)

More elaborate

- class list{
  public:
     list():head(0), cursor(0){}
     list(const int* arr, int n);
     list(const list& lst);//copy constructor
- ...
     ~list(); //delete
  private:
     list_element* head;
     list_element* cursor;
  };

- Deep copy v. referential copy

Deep: Pi is transcendental



Shallow: Mom's pie is tasty

Deep v. Shallow

- First we will examine the copy constructor. We want to build an equivalent list that is a "deep" copy.
- A "shallow“ copy would be a referential copy where the new list head would be the same value as the old list head.
- Shallow copy is a highly efficient form of copying (why?) but has a more limited utility than a deep copy(why?).

Copy constructor

-
```
list::list(const list& lst){
  if (lst.head == 0)
  {
    head =0; cursor =0;
  }
  else
  …set up
```
-
```
for( cursor = lst.head; cursor != 0; )
    {
       …chain and create        }
     cursor = head;
   }
}
```

More code

- else

```
{
  cursor = lst.head;
  list_element* h = new list_element();
  list_element* previous;
  head = h;
  h->d = lst.head->d;
  previous = h;
```

Chain and create

- for(cursor = lst.head; cursor != 0;)

```
    {
     h = new list_element();
     h->d = cursor->d;
     previous->next = h;
     cursor = cursor->next;
     previous = h;
    }
  cursor = head;
  }
```

~ destructor

- list::~list()
```
{
  for( cursor = head; cursor != 0; )
  {
    cursor = head->next;
    delete head;
    head = cursor;
  }
}
```

  Here the destructor chains through the list returning each list_element created by a corresponding new.

Use this list

- int main()
```
 {
    list a, b;
    int data[10] = {3,4, 6, 7, -3, 5};
    list d(data, 6);
    list e(data, 10);
    a.prepend(9); a.prepend(8);
    cout << " list a " << endl;
    a.print();
```

- ```
  for (int i = 0; i < 40; ++i)
          b.prepend(i*i);
      cout << " list b " << endl;
      b.print();
      list c(b);
      c.print();
      d.print();
      e.print();
  }
  ```

  Make sure you know where each constructor and destructor is invoked. Also what is printed?
  ```

Dynamic data Structures in STL

- The standard template library has the following data structures available and you are free to use them in your problem:
- #include <vector>
- can then use vector<type> name(size);
- vector is the most used – it is nearly efficient as araw array and is safer and expandable.

List is also available

- #include <list>
- gets you doubly linked lists

C++ More

- HW2 Questions?
  - How to average in disconnected graphs –
  - ignore them for averaging
  - Density is for entire set of edges –does not mean node degree is uniform; also for small graphs a low density leads to graph being disconnected.
- Review end of chapter short questions