# C++ Functions and Generics

- C++ Functions and Generics
- Graph Algorithms

# C++ new stuff in functions

- default parameters, variable argument lists,
- const parameters,
- multiple Types in a generic,
- operator overloading.

# Quiz – sum an array in C

- Review – how in C do you write a function to sum an array of doubles?

# Answer C – sum array

```
double sum(double data[], int size)
{
   double s = 0.0; int i;
      for(i = 0; i < size; ++i)
               s += data[i];
      return s;
}
```

# Generic Programming

- **1.** Relating to or descriptive of an entire group or class; general.
- **2.** *Biology* Of or relating to a genus.
- **3.a.** Not having a brand name: *generic soap.*
- **b.** Of or being a drug sold under or identified by its official nonproprietary or chemical name.
- **4.** *Grammar* Specifying neither masculine nor feminine gender:

# Generic Programming

- Writing code that can use an arbitrary type or types; in C++ this is done with template

# C++ generic Sum an array

- Consider writing a function that sums an array. Assume that most of time you want the sum to start with zero, but sometimes you want a different initial value.

- Initial value = 0  and add 3, 4, 6, 11  = 24
- Initial value = 24 and add 6, 7, 9  = 46

# Sum an array

```
template <class T>   //T is generic type
T sum(const T data[], int size, T s=0)
{
    for(int i = 0; i < size; ++i)
        s += data[i]; //+= must work for T
    return s;
}
```

# Default parameter

- T sum(const T data[], int size, T s=0)
- Ideas  data is not mutable use a const
- s will default to 0

- sum(scores, 92);
- sum(scores, 92, 58);
- Same function two different signatures
- Note could also default size; think about it

# C++ Program

```
int main()
{
    cout << "template  for sum()" << endl;
    int a[] = {1, 2, 3};
    double b[] = {2.1, 2.2, 2.3};
    cout << sum(a, 3) << endl;
    cout << sum(b, 3) << endl;
}
```

# C++ Program

- Elements of the array source are not be modified .
- Simple parameters, such as size, are typically call by value and can not be modified by the calling routine.

- This concept is called “const correctness.”

# Quiz – Modify program

- Iterate subtraction over the data

- Write a template function that outputs the elements of the array

# Answer

- for(int i = 0; i < size; ++i)
- s -= data[i];  //+= used for sum

- for(int i = 0; i < size; ++i)
- cout << data[i]  <<  '\t' ;
- //you can make this prettier

# Multiple Template Arguments

# Multiple template args

- We have used templates with one template parameter, but it is useful to have more that one distinct type in a template.
- More genericity – but be careful!!

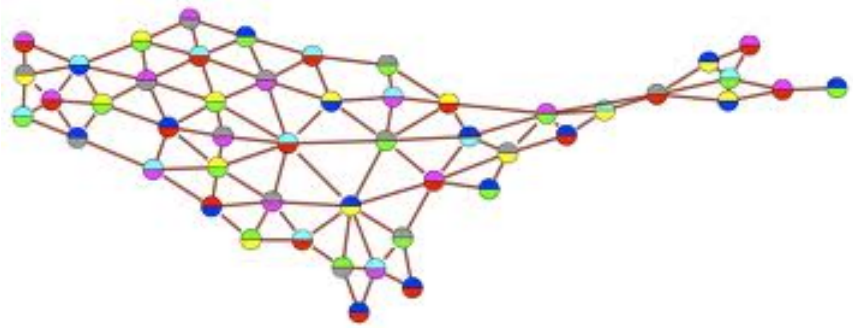
# C++ templates

- template<class T1, class T2>
- void copy(const T1 source[], T2 destination[], int size)
- {
- for(int i = 0; i < size; ++i)
- destination[i] =
- static_cast<T1>( source[i]);
- }

# C++ Casts

- More types means worrying about conversions and more signatures
- These static_cast operators are considered safe.
- The old cast operator (type) is deprecated As a reminder the other casting operators are:
- reinterpret_cast<type> highly unsafe
- dynamic_cast<type> used with classes
- const_cast<type> cast away const-ness

# Graph Theory and Algorithms



# Quiz – draw a graph

- Review  Draw the complete Graph of 4 nodes

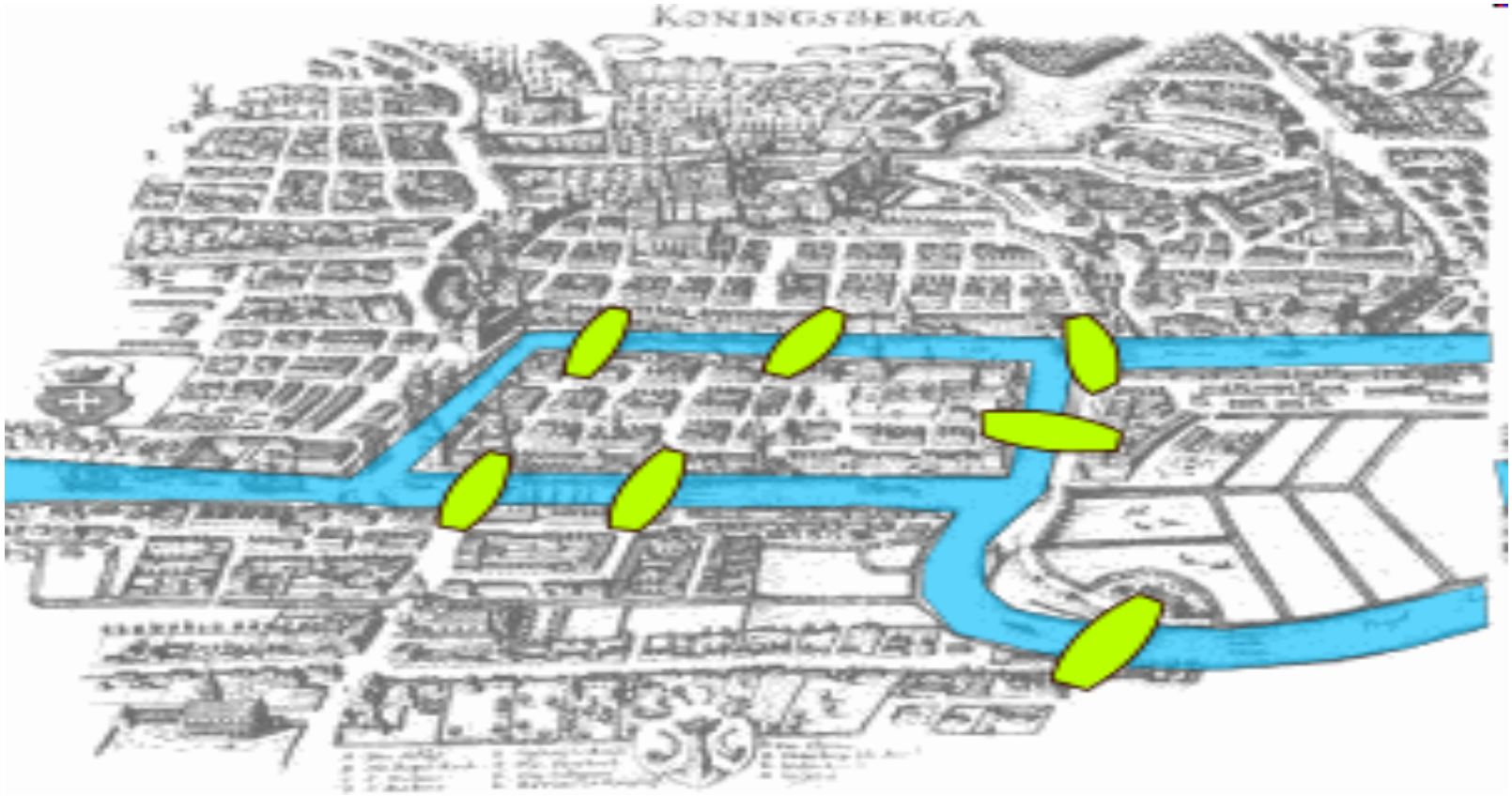
# Answer K4

- O O
- O O

# Graphs

- The **Seven Bridges of Königsberg** is a historically notable problem in mathematics. Leonhard Euler in 1735 laid the foundations of graph theory examining this problem.

# 7 Bridges Problem



# Koningsberg

- The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges.
- The problem was to find a walk through the city that would cross each bridge once and only once. The islands could not be reached by any route other than the bridges, and every bridge must have been crossed completely every time; one could not walk halfway onto the bridge and then turn around and later cross the other half from the other side.
- Euler proved that the problem has **no** solution. There could be no non-retracing continuous curve that passed through all seven of the bridges..

# Graph as a data structure

- Connectivity matrix (also distances)
- Edge List Representation
- Tradeoffs  - Graph as an ADT

# List representation

- **Definition:** A representation of a directed graph with *n* vertices using an array of *n* lists of vertices.

- List *i* contains vertex *j* if there is an edge from vertex *i* to vertex *j*.

- A weighted graph may be represented with a list of vertex / weight pairs.

- An undirected graph may be represented by having vertex *j* in the list for vertex *i* and vertex *i* in the list for vertex *j*.

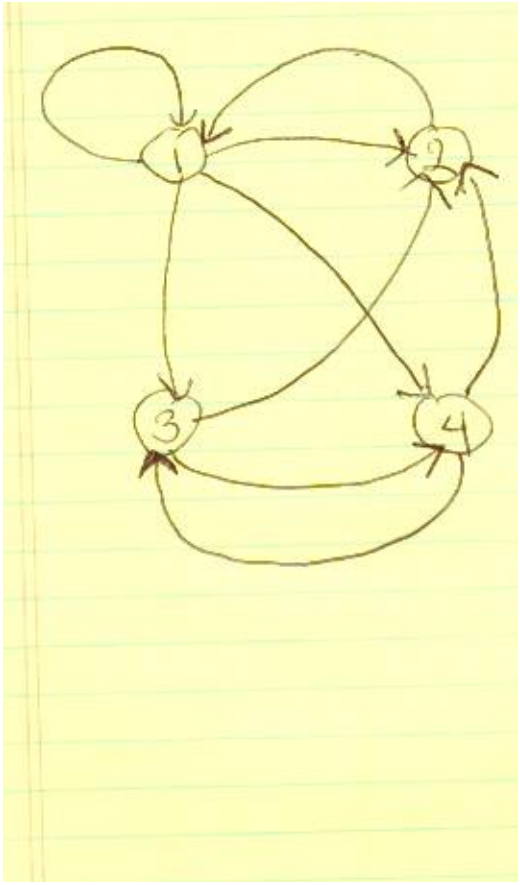
# Matrix vs List Directed Graph

- *1 2 3 4*
  *1 1 1 1 1*
  *2 1 0 0 0*
  *3 0 1 0 1*
  *4 0 1 1 0*

- *1 -> 1 -> 2 -> 3 -> 4*
  *2 -> 1*
  *3 -> 2 -> 4*
  *4 -> 2 -> 3*

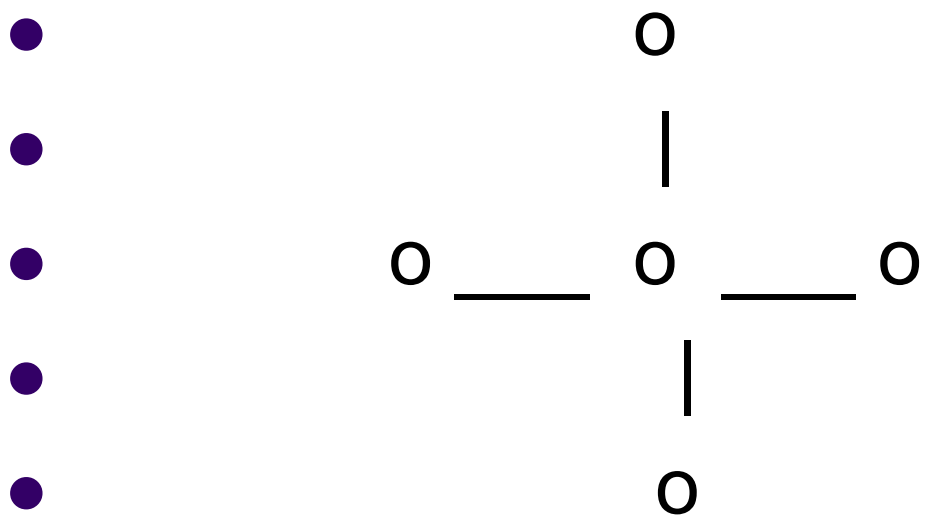
# Drawing of previous Graph

- My drawing



# Quiz

- Here is an undirected graph – generate an edge list representation and a matrix representation for it.



# Answer Matrix

- *1 2 3 4 5*
  *1 0 0 1 0 0*
  *2 0 0 1 0 0*
- *3 1 1 0 1 1*
  *4 0 0 1 0 0*
  *5 0 0 1 0 0*

# Answer Edge List

- *1 -> 3*
  *2 -> 3*
  *3 -> 1 -> 2 -> 4 -> 5*
  *4 -> 3*
- 5 -> 3

# Dijkstra Shortest Path

- Find a shortest Path Between start and destination node s to d

- Step 1 - include s in a closed set and all immediate successors of s with their distance in the open set
- Step 2 - pick the open node of list cost- say this node is n

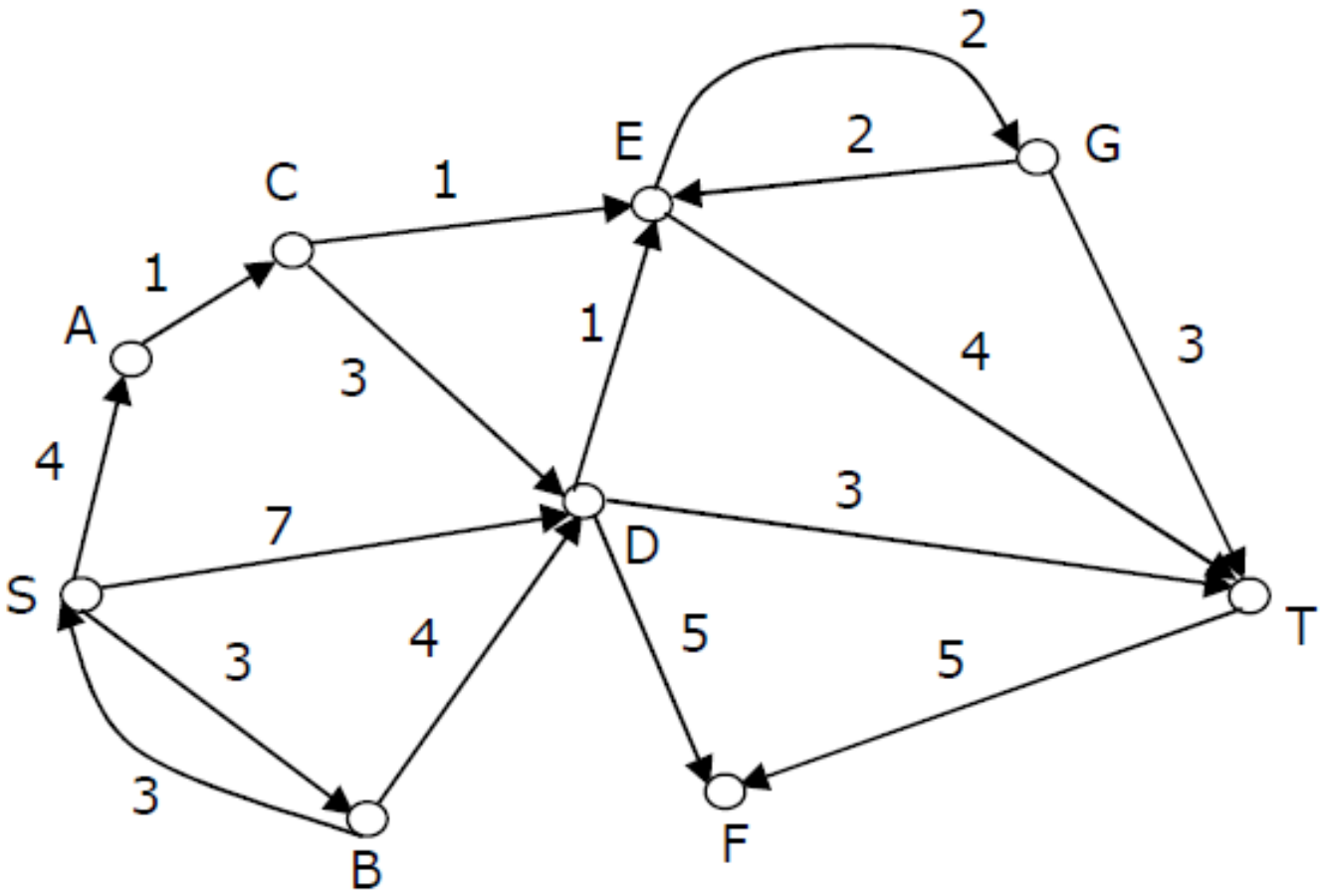
# Dijkstra

- If the node that is picked is d stop
- Otherwise compute all  succesors of n;
- Add the distance from s to n + n edge to k
- And if it improves on value in open set – store as new value
- Pick current open set distance that is a min and repeat – above steps.

# Dijkstra Termination

- If the node that is picked is d stop
- No more succesors – failed no path
- A shortest path tree should be maintained that lets you create the route

# Simulate Dijkstra by hand



# Home work Turn in

1. Programming Assignment 2-
   Dijkstra Shortest Path Algorithm

# C++ Creating Types

- C++ classes and OO
  - Point as an example
- If you have the book:
  - Chapter 4 and Chapter 5

# Quiz

- Name three native types in C

- If you have  the expression 3 / 4 what is its value?
- If you have  the expression 3.0 / 4 what is its value?

# Answers

- Any of short, int , double, char, long, long double, int* …
- 3 / 4 both int literals answer 0
- 3.0 / 4 double / int – double division 0.75

- Conclusion type matters a lot

# Enum and operator overloading example

- Enum is a simple int type
- Typedef enum color{RED, WHITE,GREEN} color;
- RED is defaulted to 0.  WHITE is 1 and GREEN is 2

# Quiz: Should you use enum?

A) Can use a named integer – so it is redundant?

B) Can use a #define - keep old C style?

C) Small related set of constants?

Pick best answer

# Answer

A) Can use a named const integer –so it is redundant? Okay

- const int TRUE = 1, FALSE = 0;

B) Can use a #define- keep old C style?- safety issues

- #define TRUE 1
- #define FALSE 0

C) Small related set of constants? – yes best answer

# Example: days

- typedef enum days{SUN, MON, TUE, WED, THU, FRI, SAT} days;
- 
- inline days operator+ (days d){
- return  static_cast<days>((static_cast<int>(d) + 1) % 7);
- }

# Days & operator overloading

- ostream& operator<<
- (ostream& out, days d){
- switch (d){
- case SUN: out << "SUN"; break;
- case MON: out << "MON"; break;
- … }
- return out;
- }

- //operator << is normally left bit shift

# Example Days

```
int main()
{
    days d = MON, e;
    e =  +d;
    cout << d << '\t' << e << endl;
}
```

# Quiz

- What native operator is <<  ?
- Can its precedence be changed ?
- How do we know which meaning to apply?

# Answer

- << left bit shift
  - int a = 5; // binary: 0000000000000101
  - int b = a << 3;
  - // binary: 0000000000101000, or 40 //decimal
- Precedence and associativity do not change

- Based on binary signature

# Why add a type?

- Types are related to domains
- So when you need to talk about widgets-you want the widget type and actions on widgets
- C had primitive forms of type extensibility

# C Type Extension

- In C you can “add “ a type using struct.
- In C++ struct is different – struct is a named scope that can include functions (methods) and have different layers of opacity (data hiding). While C++ retains struct it has the new keyword class.
- Class is almost equivalent to struct – but with different data hiding defaults.

# C type point as struct

- typedef struct point{double x, y;} point;
- void add_points(point* p1, point* p2, point* sum){
- sum->x = p1->x + p2->x;
- sum->y = p1->y + p2->y;
- }
-

# C struct

- Struct has fields – data members
- File scope functions manipulate the struct using pointer operations

# C++ point

- class point{
- public:
- double x, y;
- };

- Public, private, protected are access keywords

# Better Point

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

# Q : Review OO

- In OO why is data hidden?

# Answer

- The principle of the black box – what is under the hood should be left to experts;

- This includes representation.

# Point functions

```
point operator+ (point& p1, point& p2){
   point sum = {p1.x + p2.x, p1.y + p2.y};
   return sum;
}
 ostream& operator<< (ostream& out, point& p){
   out << "( " << p.x << ", " << p.y <<" )";
   return out;
}
.
```

# C++

```
int main()
{
  point a = {3.5, 2.5}, b = {2.5, 4.5}, c;
  cout << "a = " << a << "  b = " << b <<endl;
  cout << " sum = " << a + b << endl;
}
```

# User Defined Types

- Using point in main() looks very much like using a native type.
- Indeed this is one of our key goals in OO.
- We accomplished this by having point be a class -a user defined type. We overload the standard operators like + and << to give them appropriate "point" semantics.

# Methods

- class point{
- public:
- double getx(){return x;}
- void setx(double val){x = v;}
- Class member functions have automatic access to private members.
- p1.getx() p2.setx(2.5) p2.sety(-1.8)

# How to build an object

- Object construction is needed for OO
- Built in objects such as int double, fixed arrays are built by the compiler.
- Big part of OO/C++ mastery is how to build objects :

# A special method - constructor

- point(){  x = y = 0.0;}
- Or
- point(){ this -> x = 0.0; this ->y = 0.0}
- Default constructor – the constructor whose signature is void.

# Finally

- Constructors and destructors are our next major topic – in reading chapter 5

- To be continued.