

Discussion 1

Outline

- Lab Rules
- Basic Techniques and software
- Dijkstra Algorithm and Lab 1
- C/C++ Tutorial

Rules

- All programming are to be done on the UNIX servers (nunki.usc.edu).
- Submission rules and deadlines are strictly enforced.
- Beware that the school server may be overwhelmed towards the end of the semester
- Follow Naming conventions, file types
- The TA team will be there to help you, do not panic. Also, discussion board is the best way to communicate your problems.

TA Office Hours

- Hao Feng:
Friday 3:00pm-5:00pm, PHE 330
haofeng@usc.edu
- Amirhossein Mohajerin Ariaei:
Monday 2:00pm-4:00pm, EEB 515
a.mohajerin.a@gmail.com

TA Office Hours

- Aniket Vyas: vyas45@gmail.com
- Armen Babayan Aghan: abayana@usc.edu
- Indranil Sen: indranis@usc.edu

Asking Questions

- Use DEN forum
 - Go to DEN->Discussion Board->Forum
 - This is the most efficient way, and you can get your answers in time
 - Go to forum and see what questions other students ask

Basic techniques and software

- You can find the required software at <http://software.usc.edu>
- To log into aludra or nunki

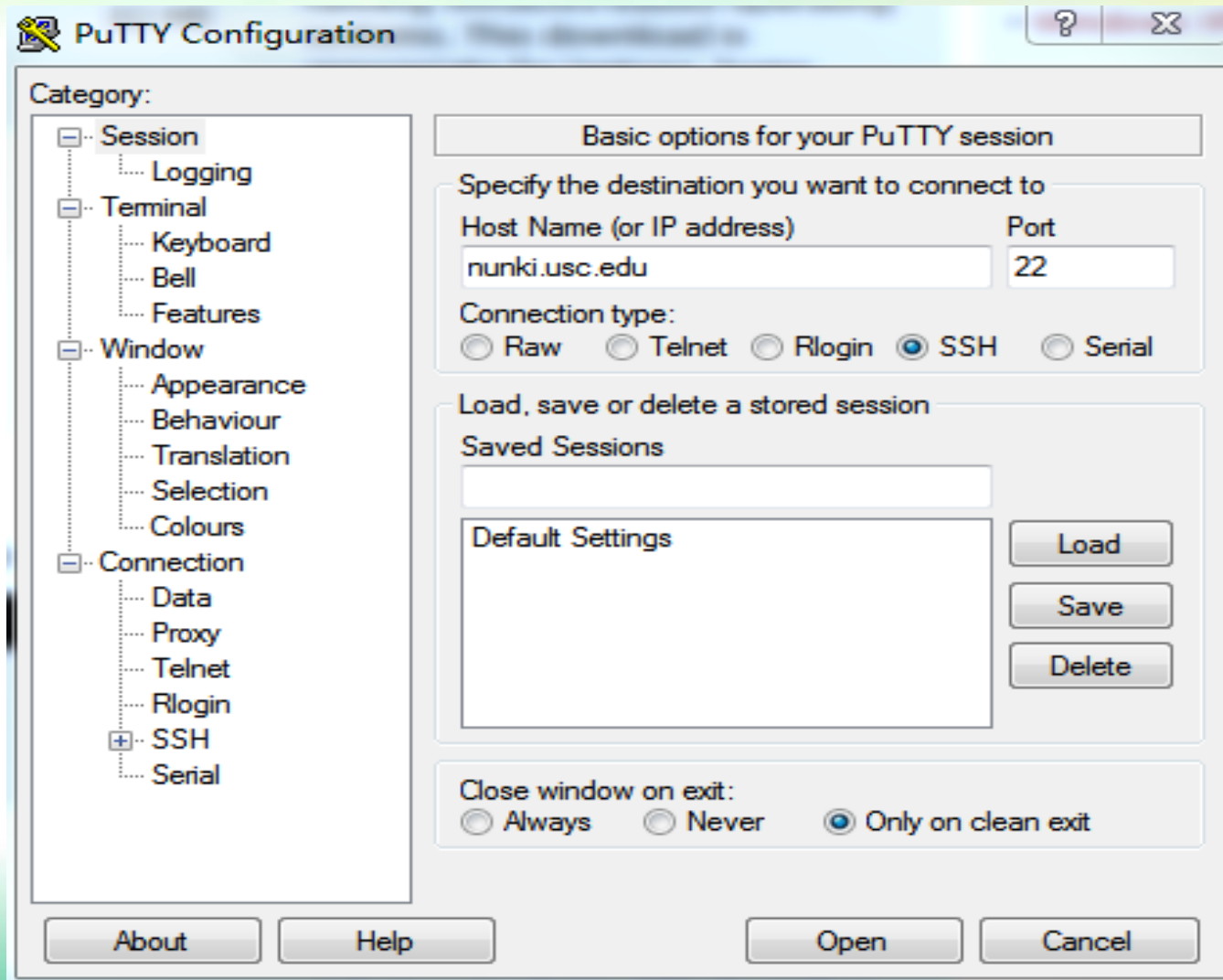
Putty or Xwin-32

- To transfer files from your local machine to your “network drive”, which is associated with your UNIX account:

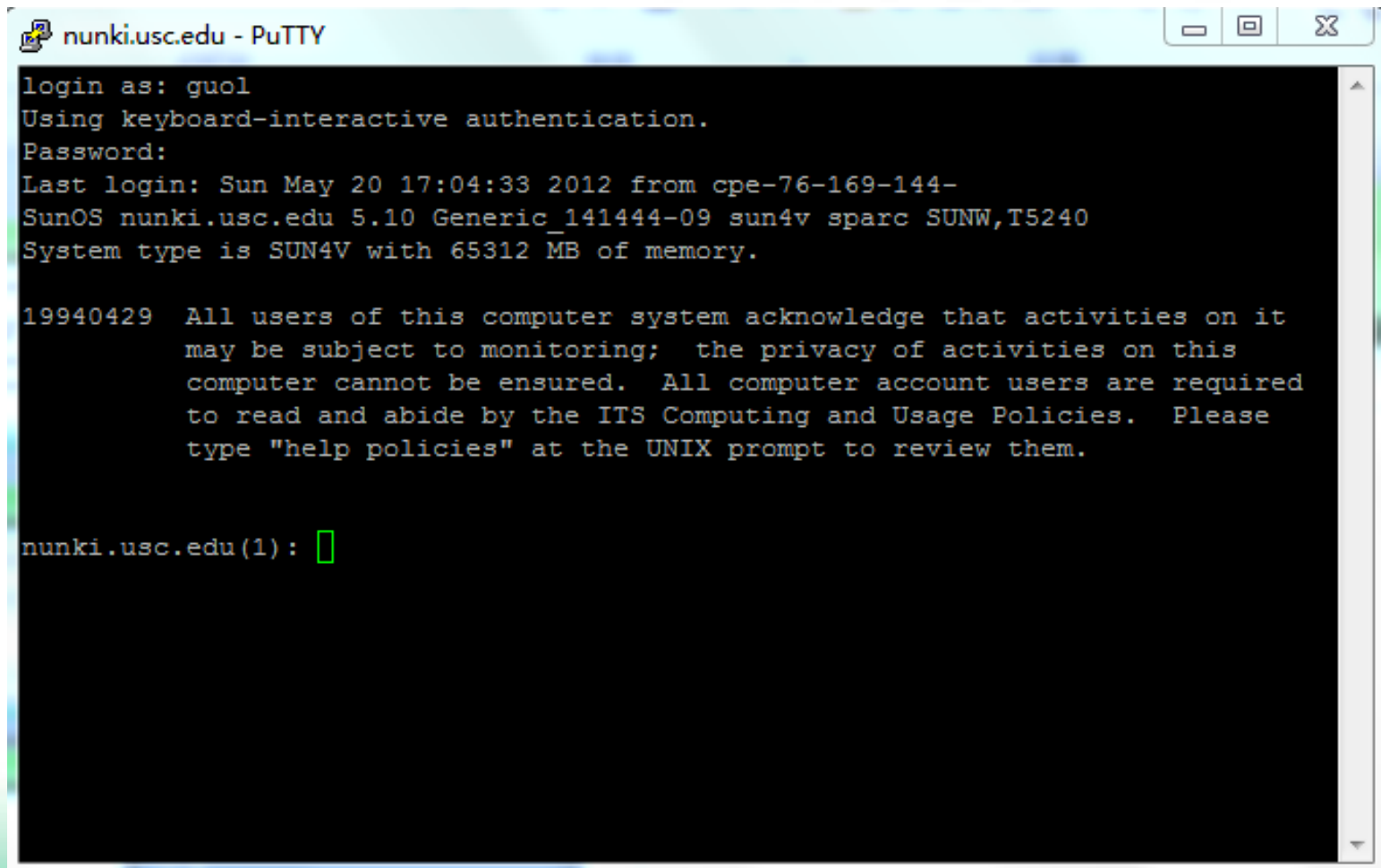
FileZilla

- To write and compile code, you do not need to download anything. You can use file editors such as “emacs” that comes with your UNIX environment

Basic techniques and software..Putty



Basic techniques and software..Putty



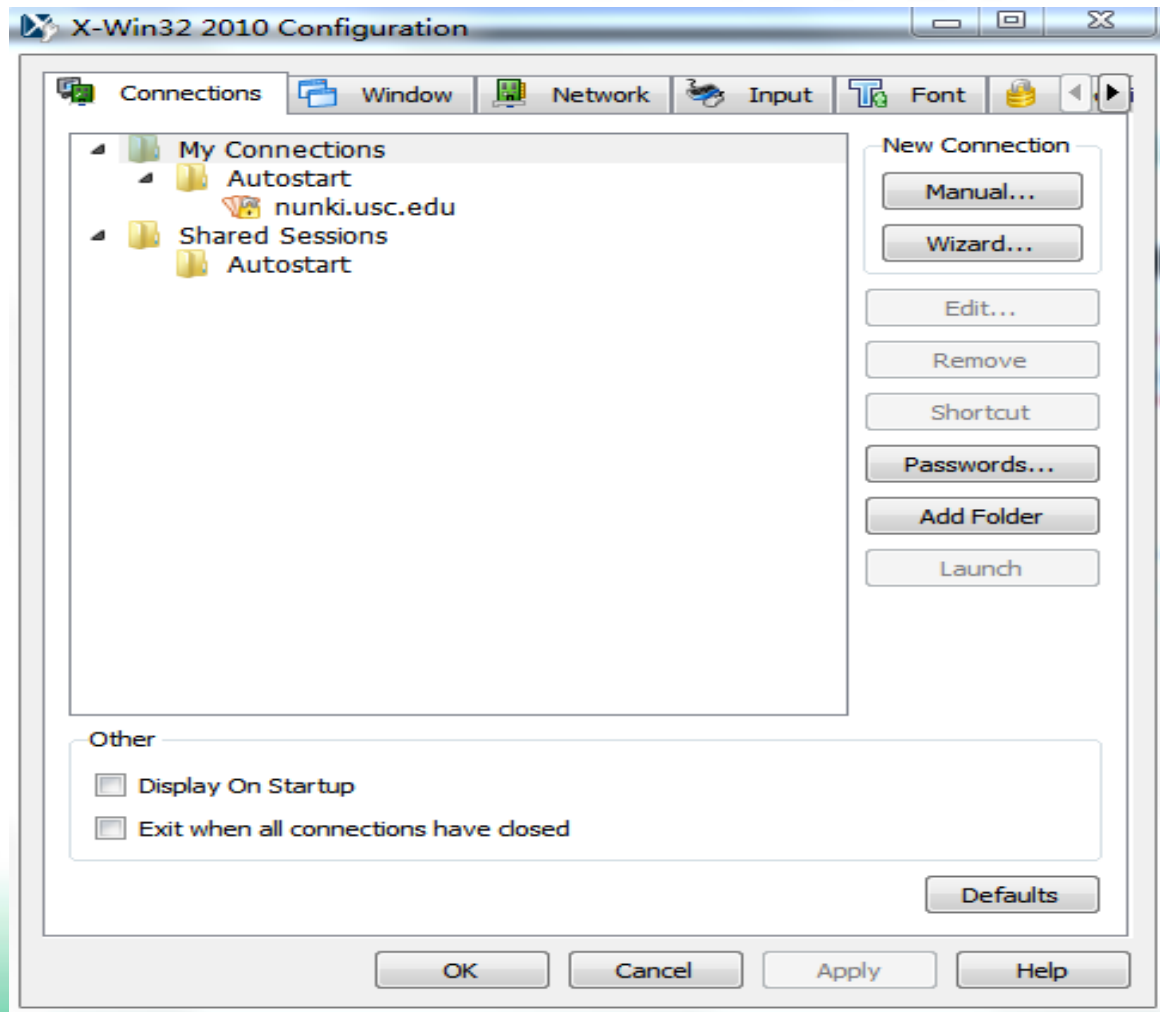
The image shows a PuTTY terminal window titled "nunki.usc.edu - PuTTY". The terminal displays the following text:

```
login as: guol
Using keyboard-interactive authentication.
Password:
Last login: Sun May 20 17:04:33 2012 from cpe-76-169-144-
SunOS nunki.usc.edu 5.10 Generic_141444-09 sun4v sparc SUNW,T5240
System type is SUN4V with 65312 MB of memory.

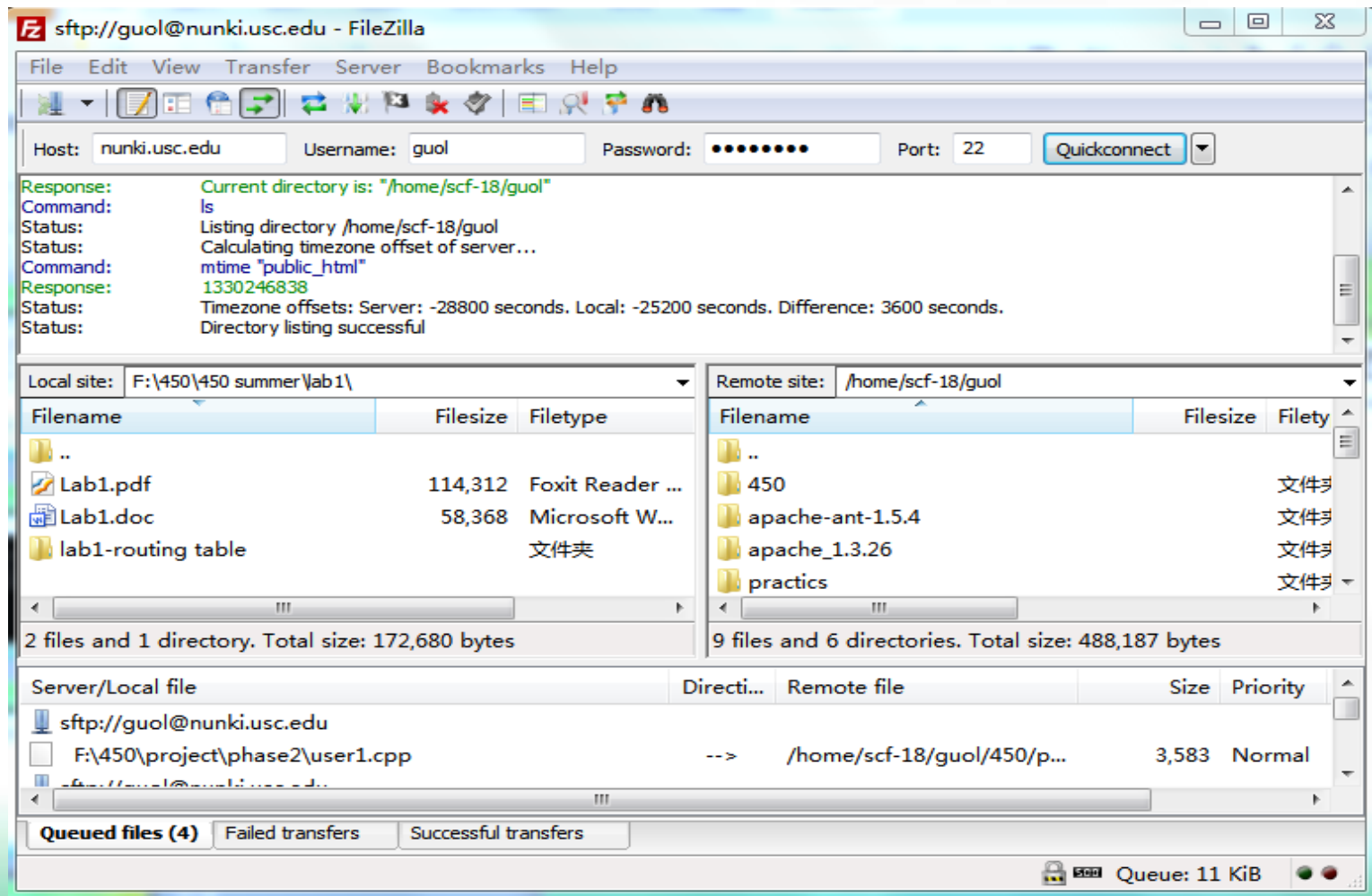
19940429  All users of this computer system acknowledge that activities on it
         may be subject to monitoring;  the privacy of activities on this
         computer cannot be ensured.  All computer account users are required
         to read and abide by the ITS Computing and Usage Policies.  Please
         type "help policies" at the UNIX prompt to review them.

nunki.usc.edu(1):
```

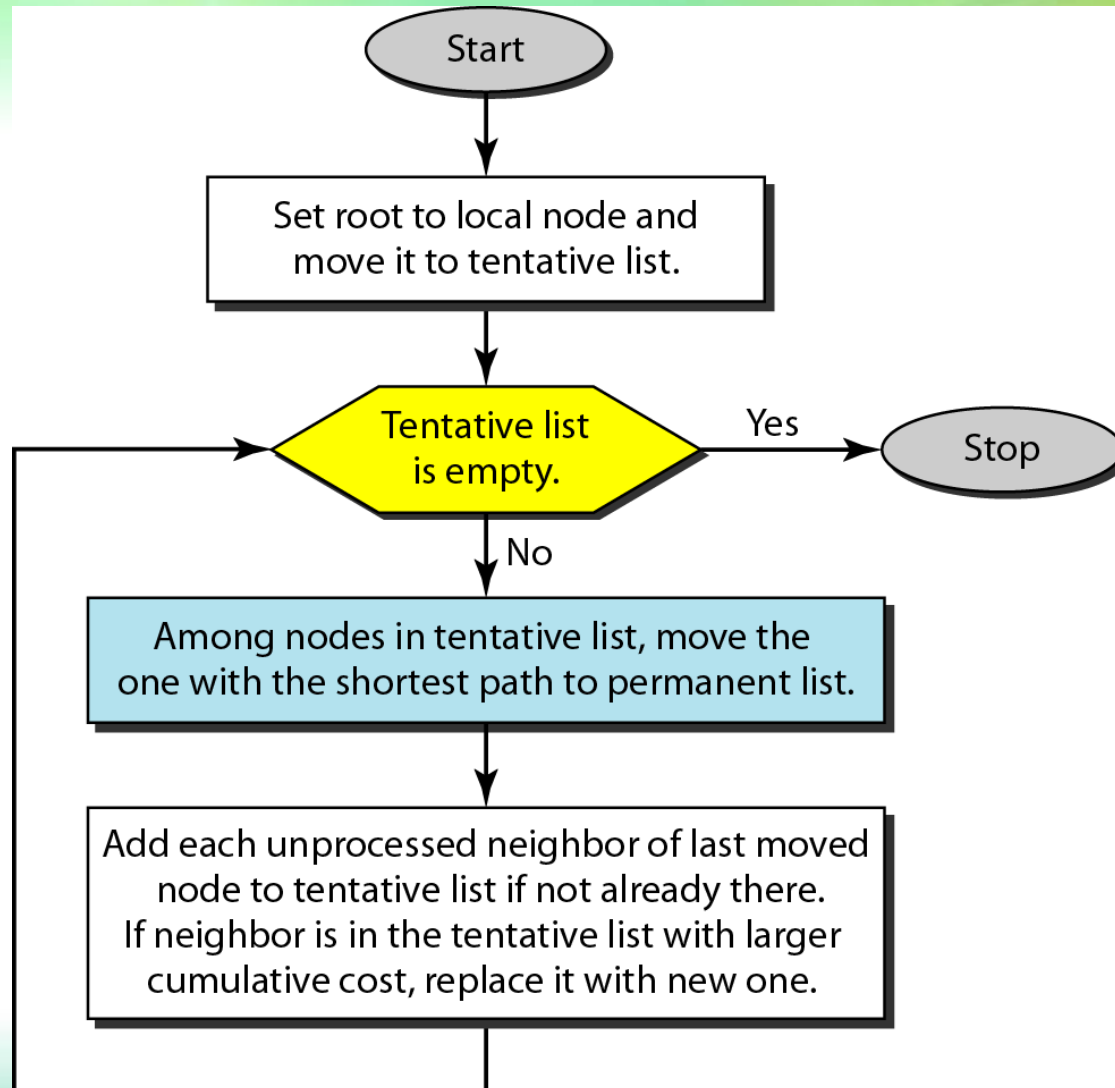
Basic techniques and software..X-Win32



Basic techniques and software..FileZilla



Dijkstra's Algorithm - Flowchart



Dijkstra's Algorithm - Notation

Dijkstra's algorithm


- Net topology, link costs known to all nodes
 - Accomplished via "link state broadcast"
 - All nodes have same info
- Computes least cost paths from one node ('source') to all other nodes
 - Gives **forwarding table** for that node
- Iterative: after k iterations, know least cost path to k dest.'s

Notation:

- $c(x,y)$: link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: current value of cost of path from source to dest. v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least cost path definitively known

Dijkstra's Algorithm - Pseudocode

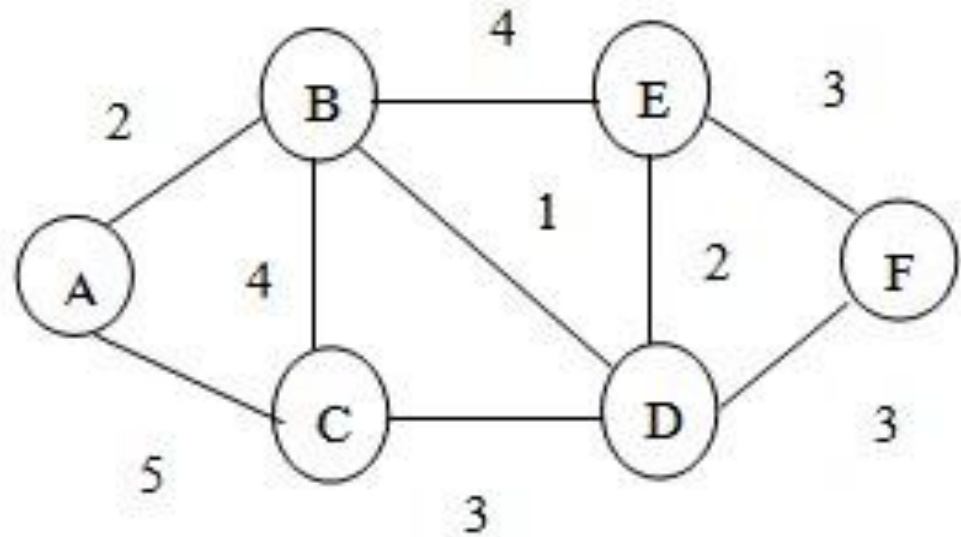
```
1  Initialization:
2  N' = {u}
3  for all nodes v
4    if v adjacent to u
5      then  $D(v) = c(u,v)$ 
6    else  $D(v) = \infty$ 
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12      $D(v) = \min( D(v), D(w) + c(w,v) )$ 
13   /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15  until all nodes in N'
```



Dijkstra's Algorithm - Initialization

- Assume A is the source

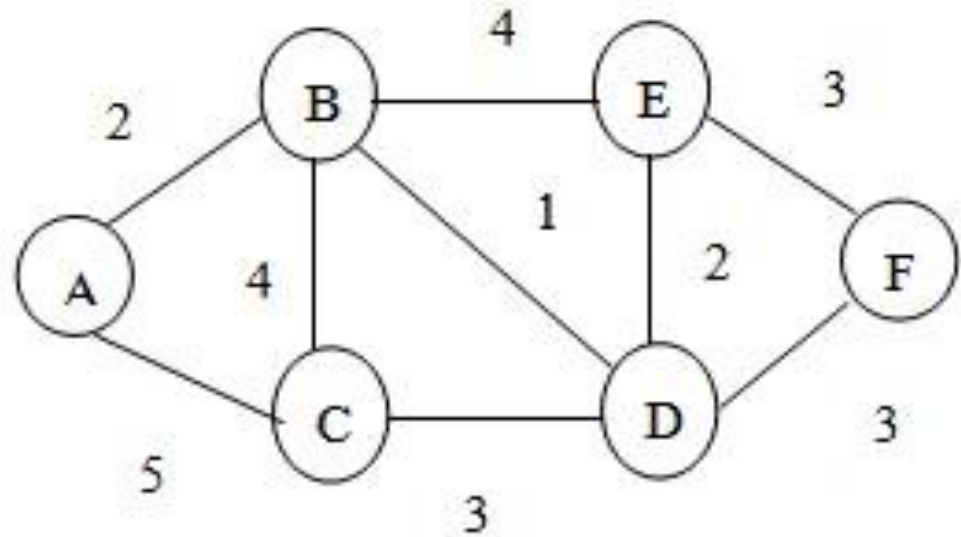
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Loop

- Assume A is the source

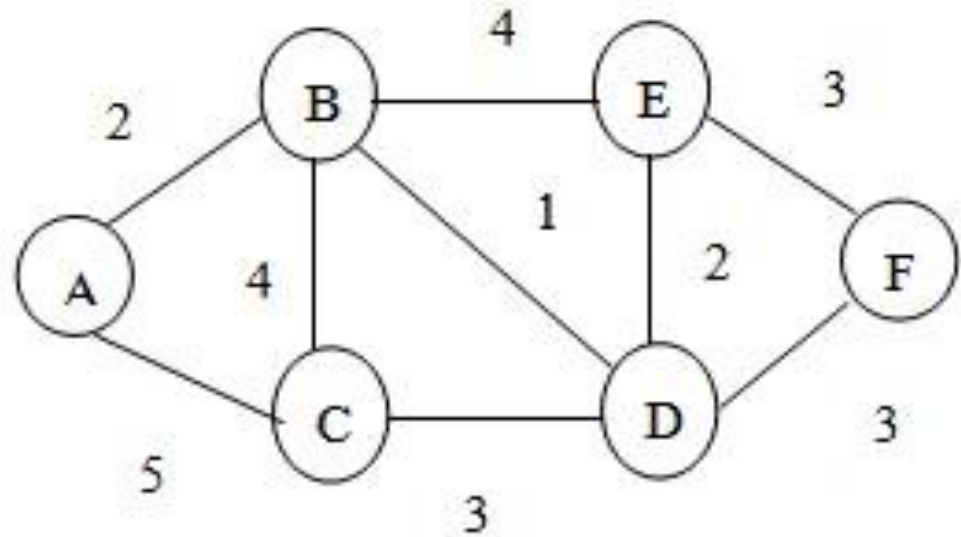
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Loop (Cont.)

- Assume A is the source

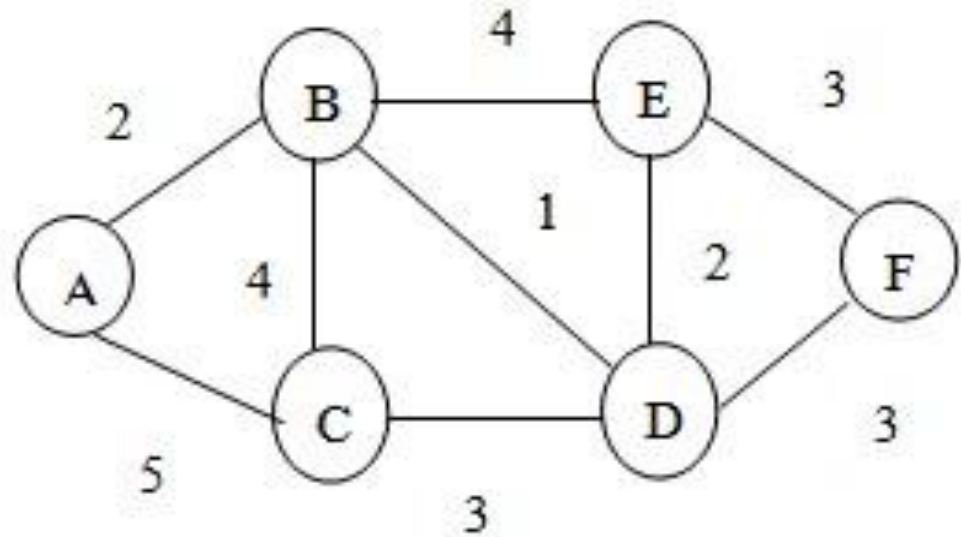
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Initialization

- Assume A is the source

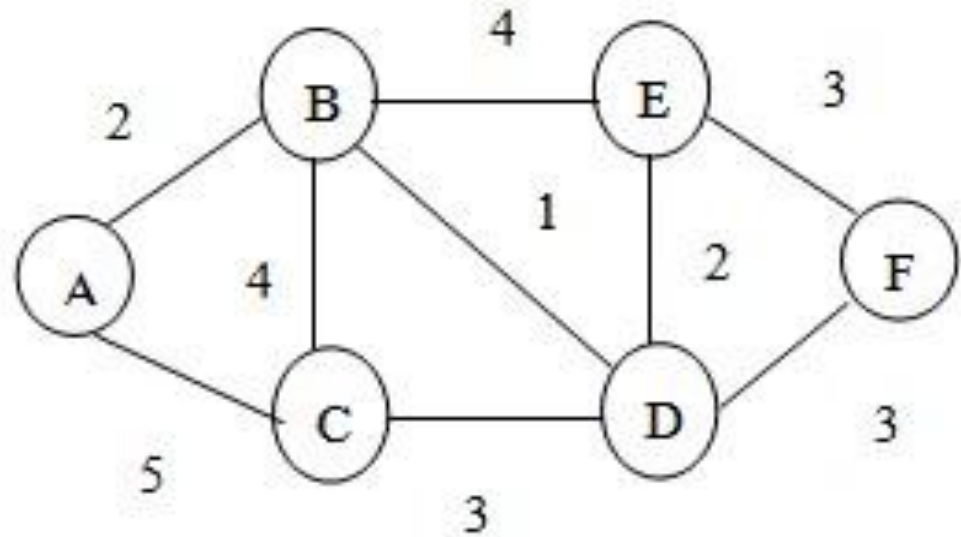
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Initialization

- Assume A is the source

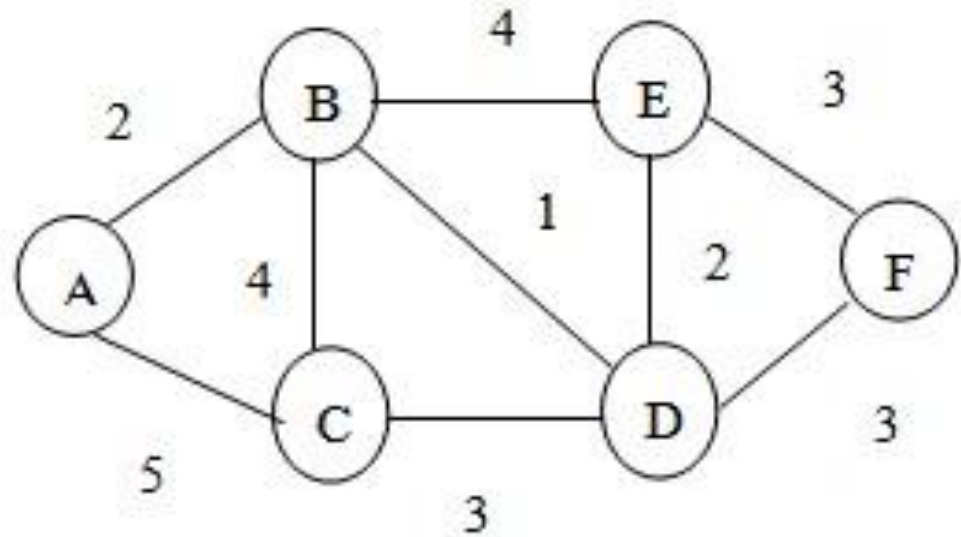
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Initialization

- Assume A is the source

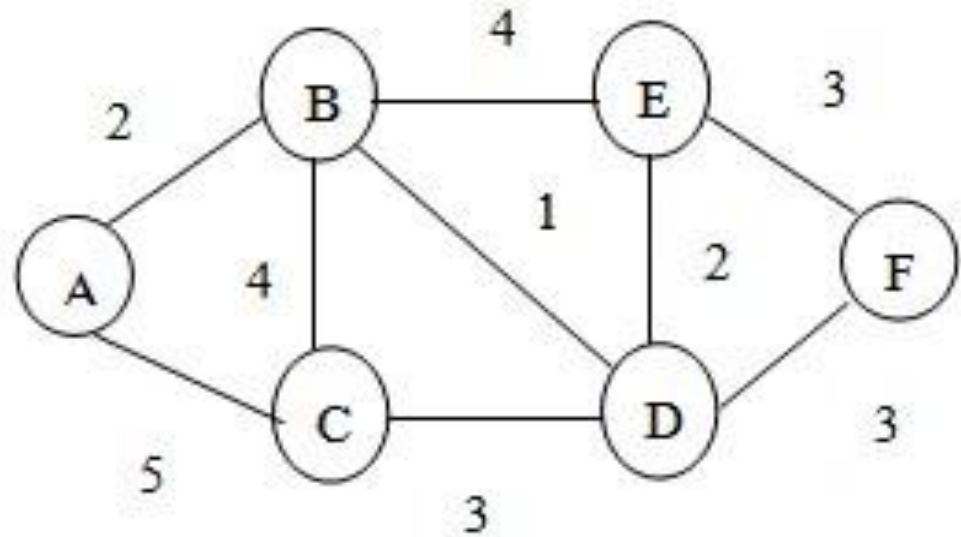
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Initialization

- Assume A is the source

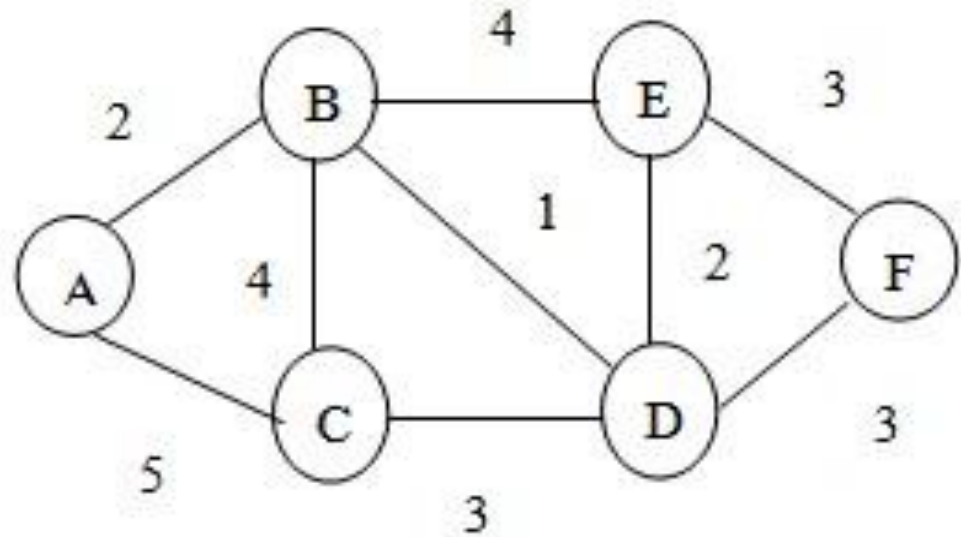
		Next Node
B	2	-
C	4	-



Dijkstra's Algorithm - Initialization

- Assume A is the source

		Next Node
B	2	-
C	4	-



Introduction to C/C++

Strategies for learning C++

- Focus on concepts and programming techniques.
(Don't get lost in language features)
- Learn C++ to become a better programmer
More effective at designing and implementing.
- C++ supports many different programming style
- Learn C++ gradually
- Don't have to know every detail of C++ to write a good C++ program.

Comments

- How to make comments:

- In C:

`a = a + b; /* comment in C */`

- In C++:

`a = a + b; // line comment`

`a = a + b; /*block comment`

`block comment`

`block comment*/`

Variable declaration

- In C: all variable definitions must occur at the beginning of a block.

Example:

```
int i;  
for (i=0; i<5; i++) { ... }
```

- In C++: variable definitions may occur at the point of use.

Example:

```
for(int i=0; i<5; i++) { ... }
```

Identifiers

- C++ reserved key word that can not be used as an identifier:
- asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t.
- and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq.
- far, huge, near (for some compiler).

Boolean

- Built-in type *bool*:
- In C: true is represented by nonzero integer values, and false by zero.
- In C++: type bool is added to represent boolean values. A bool object can be assigned the literal value true and false

Boolean

- Example:
- `int* f (int);`
- `bool flag1, flag2;`
- `flag1 = false;`
- `flag2 = f(5);`
- A zero value or a null pointer value can be converted to false implicitly; all other values are converted to true.

Constant

- In C: Constants are handled by the preprocessor through macro substitution.

```
#define MAX 10
```

```
#define MAX f(5) // wrong !
```

Declared Constants

- In C++: The keyword `const` allows explicit definition of constants objects.
- `// max and a cannot be modified after initialization`
- `const int max = 10;`
- `const int a = f(5);`
- `void f(int i, const int j)`

`// j is treated as a constant in f()`

```
{  
    i++; // ok  
    j = 2; // error  
}
```

Cast: Type Conversion

- Cast: Type conversion
- In C: (double) a;
- In C++: functional notation.
- Example:
average = double(sum) / double(count);
- Other C++ type conversion operator for OO programming:
static cast, const cast, reinterpret cast, dynamic cast

Structure

- In C++: The keyword struct denotes a type (aggregation of elements of arbitrary type).
- Two structures are different even when they have the same members.

```
struct student{  
    char name[20];  
    int id;  
};  
student s1;  
struct new_student{  
    char name[20];  
    int id;  
};  
new_student s2;  
s1 = s2; //Wrong! Two different types.
```

String

- In C, a string is a null-terminated array of characters. It can be represented in two ways;

1) An array of type char

2) By a pointer of type char.

```
char s1[] = "spring";
```

```
char s1[7] = "spring";
```

```
char s1[] = {'s','p','r','i','n','g'};
```

```
char *s2 = "fall";
```

String

- In C++, there is no built-in string data type. The C++ standard library provides library type string.
- To use type string provided by the standard library, the header string must be included.
- `#include <cstring>`

String

- Example:

```
#include <iostream>
```

```
#include <cstring>
```

```
using namespace std;
```

```
string s1;
```

```
string s2 = "hello!";
```

```
string s3 = s2;
```

```
string s4( 5, 'x');
```

```
string s5 = s2 + s3 // string concatenation
```

String

- The string type provides a variety of useful string operations.

For example:

```
String name= "windsor library";  
void m()  
{  
    string s=name.substr (0,6); //s="windsor"  
    name.replace (0,6, "windsor public");  
    //name="windsor public library";  
}
```

String

- The `c_str` function convert a string type to **C-style string**.

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    string name="windsor";
    printf("name:%s\n", name.c_str());
    cout<<"name: "+name<<endl;
    return 0;
}
```

Function Prototype

- In C++, function prototyping is required to type-check function calls.
- Format for prototype:
type name (argument_type1, argument_type2, ...);
- It does not include a statement for the function (no function body).
- It ends with a semicolon sign (;).
- In the argument enumeration it is enough to put the type of each argument.

Function Prototype

- **Version 1:** (function definition occurs before main, works even without prototype)

```
void f1(int); // function prototype (declaration )
```

```
int f2(int x) // declaration & definition
```

```
{ return x + 10;}
```

```
void f1(int x)// definition
```

```
{cout << x;}
```

```
int main()
```

```
{f1( f2(10) );}
```


Function Prototype

- **Version 2:** (function definition occurs after main, not working)

```
int main()
```

```
{f1( f2(10) );}
```

```
void f1(int); // function prototype (declaration )
```

```
int f2(int x) // declaration & definition
```

```
{ return x + 10;}
```

```
void f1(int x) // definition
```

```
{cout << x;}
```

Function Prototype

- Better approach:

```
void f1(int);
```

```
int f2(int);
```

```
int main(){
```

```
f1( f2(10) );
```

```
}
```

```
int f2(int x) { return x + 10;}
```

```
void f1(int x){cout << x;}
```

Function Prototype

- `#include <iostream>`
`using namespace std;`
`void odd (int a); //void odd (int) is enough;, void odd (int x) also OK.but`
`//not recommended.`
`void even (int a);`
`int main () {`
 `int i;`
 `do {`
 `cout << "Type a number: (0 to exit)";`
 `cin >> i; odd (i); }`
 `while (i!=0);`
 `return 0;`
`}`
`void odd (int a)`
`{ if ((a%2)!=0) cout << "Number is odd.\n"; else even (a); }`
`void even (int a)`
`{ if ((a%2)==0) cout << "Number is even.\n"; else odd (a); }`

Function Prototype

- Two or more functions maybe given the same name provided the type signature (number of the arguments AND the type of the arguments) for each function is unique.
- Example:
int multi (int, int);
double multi (double, double);
int multi (int, int, int);
- How about:
int add(int, int);
double add(int, int);

Simple I/O

- Input/output in C++ is supported by the use of I/O stream libraries.
- The stream library defines input/output for every built-in type.
- The standard output is defined as cout and standard input cin.
- “<<” put to
- “>>” get from

Simple I/O Example 1

//Input data from keyboard:

```
cin >> x;
```

```
cin >> x >> y;
```

// Output data to screen

```
cout << x;
```

```
cout << "hello world!";
```

```
cout << "The result is:" << GetResult();
```

```
cout << "x is: " << x << "y is:" << y << endl;
```

Simple I/O Example 2

```
#include <iostream>
using namespace std;
int main() {
    int id;
    float av;
    char name[20];
    cout << "Enter the id, average and the name:";
    cin >> id >> av >> name;
    cout << "ID: " << id << endl << "Name: " << name << endl <<
    "Average: " << av << endl;
    return 0;}
```

Simple I/O Example 3

- ```
int main()
{
 string str;
 cout<<"Please enter your name:";
 cin>>str;
 cout<<"Hello, "<<str<<"!\n";
 return 0;
}
```

Input: "Harry Potter", what will be the output?



## Simple I/O Example 4

- Using `getline()` function to read a whole line.

```
int main()
{
 string str;
 cout<<"Please enter your name:";
 getline(cin, str);
 cout<<"Hello, " << str << "!\n";
}
```

# File I/O

- The iostream library contains the file stream component which provides facilities for file I/O.
- Object of type **ifstream** is defined to read from a file, and object of type **ofstream** to write to a file.

```
#include <fstream>
```

```
ifstream infile;
```

```
ofstream outfile;
```

```
infile.open("input_file.name");
```

```
outfile.open("output_file.name") ;
```

# File I/O

- Operator >> and << are used in the same way as they are used in cin and cout for input/output.

```
int x;
```

```
infile >> x;
```

```
outfile << x;
```

# File I/O Example 1

- Write a program that reads an income from the file `income.in`, calculate the tax and output the income and tax to the file `tax.out`.

# File I/O Example 1 Answer

- ```
#include <fstream>
#include <iostream>
using namespace std;
const int CUTOFF = 5000;
const float RATE1 = 0.3;
const float RATE2 = 0.6;
int main(){
    int income, tax;
    ifstream infile;
    ofstream outfile;
    infile.open( "income.in" );
    outfile.open( "tax.out" );
```

File I/O Example 1 Answer Cont'd...

```
while ( infile >> income )
{
    if ( income < CUTOFF )
        tax = RATE1 * income;
    else
        tax = RATE2 * income;
    outfile << "Income = " << income << " dollors\n"
    << "Tax = " << tax << " dollors\n\n";
}
infile.close();
outfile.close();
return 0;
}
```

File I/O Example 2

- Write a program that reads lines until end-of-file from a name file, sort the names, and then write the sorted name list to another file.

File I/O Example 2 Answer

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
const int MaxSize = 1000;
void sort( string a[], int count);
```


File I/O Example 2 Answer Cont'd...

```
int main(){
    // get input from file
    string a[MaxSize];
    int count;
    ifstream infile;
    cout<< "Enter input file name: ";
    cin >> filename;
    infile.open( filename.c_str() );
    if( !infile ){
        cerr << "Can't open file " << filename << endl;
        exit(0);
    }
    int count;
    for( count = 0 ;
        count < MaxSize && getline( infile, a[count] );
        count ++ );
```

File I/O Example 2 Answer Cont'd...

```
// sort and output
sort( a, count );
ofstream outfile;
cout << "Enter output file name: ";
cin >> filename;
outfile.open( filename.c_str() );
for ( int i=0; i<count; i++ )
    outfile << a[i] << endl;
infile.close();
outfile.close();
}
```

File I/O Example 2 Answer Cont'd...

```
//Insertion sort
void sort( string a[], int count ){
    string temp;
    int i, j;
    for( i=0; i< count -1; i++ ){
        temp = a[i+1];
        for (j = i; j>=0; j--){
            if (temp < a[j] )
                a[j+1] = a[j];
            else
                break;
            a[j+1] = temp;
        }
    }
}
```

Conversion

- String Streams
- By using a string stream, we can easily convert a number to string, or a string to number using << and >>operator.
- **Example 1: From number to string:**

```
float x = 3.1415926;  
ostringstream ostr;  
ostr << x;  
string output = ostr.str();
```

Conversion

- Example 2: From string to number:

```
string number = "1234.567";
```

```
istringstream instr(number);
```

```
double x;
```

```
instr >> x;
```

Conversion

- Example 3: From string to number by using stringstream:

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main ()
{ string mystr; float price=0; int quantity=0;
  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

Conversion

- Example 4: From string to char*

```
string str="abc";
```

```
char *p=const_cast<char *>(str.c_str());
```

or

```
char *p=new char[str.length()+1];
```

```
strcpy(p, str.c_str());
```

File I/O

- We can use string streams to "break down" a line of input.
- Example:

```
#include <sstream> // must be included
#include <string>
#include <iostream>
using namespace std;
int main(){
    istringstream buf( "A test 12 12.345" );
    string s1, s2;
    int x; float y;
    buf >> s1 >> s2 >> x >> y; // white-space delimited input
    cout << s1 << endl<<s2 <<endl<< x<<endl << y<<endl; }
```


Formatting Output

- C++ stream manipulators can be used to format the output.
- The `<iomanip>` header file has to be included.
- `#include <iomanip>`

Formatting Output

```
#include<iostream>
#include<iomanip> //Include header file iomanip.
using namespace std;
int main(){ double a[5];
    for ( int i=0; i<5; i++)
        a[i] = 3.1415926 * i * i * i;
    cout << "Output using default settings" << endl;
    for ( int i=0; i<5; i++) cout << i << " " << a[i] << endl;
    cout << "Output using formatted setting" << endl;
    cout << fixed << setprecision(2); //use fixed and setprecision as combination to
                                     //set the precision of float number output.

    for ( int i=0; i<5; i++)
        cout << setw(2) << i << " " << setw(8) << a[i] << endl;
        //use setw to set the width of output.
    return 0; }
```

Formatting Output

- Output of the previous example:
- Output using default settings
 - 0 0
 - 1 3.14159
 - 2 25.1327
 - 3 84.823
 - 4 201.062
- Output using formatted setting
 - 0 0.00
 - 1 3.14
 - 2 25.13
 - 3 84.82
 - 4 201.06

Formatting Output

```
#include <iostream>
#include <iomanip>
#include <cmath> // sqrt prototype
using namespace std;
int main()
{
    double root2 = sqrt( 2.0 ); // calculate square root of 2
    int places;
    cout << "Square root of 2 with precisions 0-9." << endl
    << "Precision set by ios_base member-function "
    << "precision:" << endl;
    cout << fixed; // use fixed precision
```

Formatting Output

```
for ( places = 0; places <= 9; places++ ) {  
    cout.precision( places );  
    cout << root2 << endl;  
}  
cout << "\nPrecision set by stream-manipulator "  
<< "setprecision:" << endl;  
    // set precision for each digit, then display square root  
for ( places = 0; places <= 9; places++ )  
    cout << setprecision( places ) << root2 << endl;  
return 0;  
    } // end main
```

Formatting Output

- Square root of 2 with precisions 0-9.
- Precision set by `ios_base` member-function `precision`:
- 1
- 1.4
- 1.41
- 1.414
- 1.4142
- 1.41421
- 1.414214
- 1.4142136
- 1.41421356
- 1.414213562
-
- Precision set by stream-manipulator `setprecision`:
- 1
- 1.4
- 1.41
- 1.414
- 1.4142
- 1.41421
- 1.414214
- 1.4142136
- 1.41421356
- 1.414213562

Header File

- Header File for C++
- Header file names no longer maintain the **.h** extension.
- Header files that come from the C language is preceded by a “c” character to distinguish from the new C++ exclusive header files that have the same name. For example **stdio.h** becomes **cstdio**.
- All classes and functions defined in standard libraries are under the **std** namespace instead of being global.

Header File

- List of the standard C++ header files:
- <algorithm> <bitset> <deque> <exception>
<fstream> <functional> <iomanip> <ios>
<iosfwd> <iostream> <istream> <iterator>
<limits> <list> <locale> <map> <memory>
<new> <numeric> <ostream> <queue>
<set> <sstream> <stack> <stdexcept>
<streambuf> <string> <typeinfo> <utility>
<valarray> <vector>

Header File

- **// ANSI C++ example**

```
#include <cstdio>
using namespace std;
int main ()
{ printf ("Hello World!");
  return 0; }
```

- **// pre ANSI C++ example**

```
// also valid under ANSI C++, but deprecated
#include <stdio.h>
int main ()
{ printf ("Hello World!");
  return 0; }
```

Escape Codes

- These are special characters that are difficult or impossible to express otherwise in the source code of a program. All of them are preceded by a backslash (\). Here you have a list of some of such escape codes:

<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\t</code>	tab
<code>\v</code>	vertical tab
<code>\b</code>	backspace
<code>\f</code>	form feed (page feed)
<code>\a</code>	alert (beep)
<code>\'</code>	single quote (')
<code>\"</code>	double quote (")
<code>\?</code>	question mark (?)
<code>\\</code>	backslash (\)

Arguments passed by reference

```
// passing parameters by reference(&)
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a*=2;
    b*=2;
    c*=2;
}
int main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" <<
        z;
    return 0;
}
```

output:

x=2, y=6, z=14

Default values in parameters

```
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
    int r;
    r=a/b;
    return (r);
}
int main ()
{
    cout << divide (12);
    cout << endl;
    cout << divide (20,4);
    return 0;
}
```

output:

6
5

Object-Oriented Programming

- First-class objects - atomic types in C
 1. int, float, char
 2. have:
 - values
 - sets of operations that can be applied to them
 3. how represented irrelevant to how they are manipulated
- Other objects - structures in C
 1. cannot be printed
 2. do not have operations associated with them (at least, not directly)

Object-Oriented Idea

- Make all objects, whether C-defined or user-defined, first-class objects
- For C++ structures (called classes) allow:
 1. functions to be associated with the class
 2. only allow certain functions to access the internals of the class
 3. allow the user to re-define existing functions (for example, input and output) to work on class

Classes of Objects in C++

- Classes
 1. similar to structures in C (in fact, you can still use the struct definition)
 2. have fields corresponding to fields of a structure in C (similar to variables)
 3. have fields corresponding to functions in C (functions that can be applied to that structure)
 4. some fields are accessible by everyone, some not (data hiding)
 5. some fields shared by the entire class

Instances of Classes in C++

- A class in C++ is like a type in C
- Variables created of a particular class are instances of that class
- Variables have values for fields of the class
- Class example: Student
 1. has name, id, gpa, etc. fields that store values
 2. has functions, changeGPA, addCredits, that can be applied to instances of that class
- Instance examples: John Doe, Jane Doe
each with their own values for the fields of the class

Classes in C++

- Classes enable a C++ program to model objects that have:
 1. attributes (represented by data members).
 2. behaviors or operations (represented by member functions).
- Types containing data members and member function prototypes are normally defined in a C++ program by using the keyword **class**.

Classes in C++

- A class definition begins with the keyword `class`. The body of the class is contained within a set of braces, `{ }`; (notice the semi-colon).
- Within the body, the keywords `private:` and `public:` specify the access level of the members of the class. **Classes default to private.**
- Usually, the data members of a class are declared in the `private:` section of the class and the member functions are in `public:` section.
- Private members of the class are normally not accessible outside the class, i.e., the information is hidden from "clients" outside the class.

Classes in C++

- A member function prototype which has the very same name as the name of the class
- may be specified and is called the **constructor** function.
- The definition of each member function is "tied" back to the class by using the binary scope resolution operator (**::**).
- The operators used to access class members are identical to the operators used to access structure members, e.g., the dot operator (**.**).

Classes in C++

```
#include <iostream>
#include <cstring> // This is the same as string.h in C
using namespace std;
class Numbers // Class definition
{
    public:    // Can be accessed by a "client".
        Numbers ( ) ;    // Class "constructor"
        void display ( ) ;
        void update ( ) ;
    private:  // Cannot be accessed by "client"
        char name[30] ;
        int a ;
        float b ;
};
```

Classes Example (continued)

```
Numbers::Numbers ( ) // Constructor member function
{
    strcpy (name, "Unknown") ;
    a = 0;
    b = 0.0;
}
```

```
void Numbers::display ( ) // Member function
{
    cout << "\nThe name is " << name << "\n" ;
    cout << "The numbers are " << a << " and " << b
    << endl ;
}
```

Classes Example (continued)

```
void Numbers::update ( ) // Member function
{
    cout << "Enter name" << endl ;
    cin.getline (name, 30) ;
    cout << "Enter a and b" << endl ;
    cin >> a >> b;
}
```

Classes Example (continued)

```
int main ( )    // Main program
{
    Numbers no1, no2 ;    // Create two objects of
                           // the class "Numbers"

    no1.update ( ) ;    // Update the values of
                        // the data members

    no1.display ( ) ;    // Display the current
    no2.display ( ) ;    // values of the objects
}
```

Example Program Output

```
r1tel (~) freuler 53> example.out
```

The name is Rick Freuler

The numbers are 9876 and 5.4321

The name is Unknown

The numbers are 0 and 0

More Detailed Classes Example

```
#include <iostream>
#include <cstring>
using namespace std;
class Numbers    // Class definition
{
    public:
        Numbers (char [ ] = "Unknown", int = 0, float = 0.0) ;
        void display ( ) ;
        void update ( ) ;
    private:
        char name[30];
        int a;
        float b;
};
```

More Detailed Classes Example (continued)

```
Numbers::Numbers (char nm[ ], int j, float k )
```

```
{  
    strcpy (name, nm) ;  
    a = j ;  
    b = k ;  
}
```

```
void Numbers::update ( )
```

```
{  
    cout << "Enter a and b" << endl ;  
    cin >> a >> b ;  
}
```

More Detailed Classes Example (continued)

```
void Numbers::display( )
{
    cout << "\nThe name is " << name << '\n' ;
    cout << "The numbers are " << a << " and " << b
        << endl ;
}
int main ( )
{
    Numbers no1, no2 ("John Demel", 12345, 678.9);
    no1.display ( ) ;
    no2.display ( ) ;
}
```

More Detailed Example Program Output

```
r1rhl (~) freuler 76> example.out
```

The name is Unknown

The numbers are 0 and 0

The name is John Demel

The numbers are 12345 and 678.9

Reference

- C++ Language Tutorial:

<http://cplusplus.com/doc/>

<http://newdata.box.sk/bx/c/>