C++

Session #9

#9

- Makefile
- File - I/O
- Sockets

Makefile

Makefile is a program building tool which runs on Unix, Linux, and their flavors. It aids in simplifying building program executables that may need various modules. To determine how the modules need to be compiled or re-compiled together, make takes the help of user-defined makefiles.

Makefile

Compiling the source code files can be tiring, especially when you have to include several source files and type the compiling command every time you need to compile. Makefiles are the solution to simplify this task.

Makefiles are special format files that help build and manage the projects automatically.

Makefile

It is very common that a final binary will be dependent on various source code and source header files. Dependencies are important because they let the make Known about the source for any target.

```
target [target...] : [dependent ....]
[ command ...]
```

Files - I/O

So far, we have been using the iostream standard library, which provides **cin** and **cout** methods for reading from standard input and writing to standard output respectively.

fstream is another C++ standard library like iostream and is used to read and write on files.

Files - I/O

These are the data types used for file handling from the fstream library:

- ofstream
  - It is used to create files and write on files.
- ifstream
  - It is used to read from files.
- fstream
  - It can perform the function of both ofstream and ifstream which means it can create files, write on files, and read from files.

Files - I/O

A file MUST be opened before you can read from it or write to it. Either ofstream or fstream object may be used to open a file for writing. And ifstream object is used to open a file for reading purpose only.

Files - I/O

We need to tell the computer the purpose of opening our file. For e.g.- to write on the file, to read from the file, etc. These are the different modes in which we can open a file.

- ios::app
  - opens a text file for appending. (appending means to add text at the end).
- ios::ate
  - opens a file for output and move the read/write control to the end of the file.
- ios::in
  - opens a text file for reading.
- ios::out
  - opens a text file for writing.
- ios::trunc
  - truncates the content before opening a file, if file exists.

Files - I/O

You can combine two or more of these values by ORing them together. For example if you want to open a file in write mode and want to truncate it in case that already exists

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc );
```

Files - I/O

When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.

Files - I/O

Both istream and ostream provide member functions for repositioning the file-position pointer. These member functions are seekg ("seek get") for istream and seekp ("seek put") for ostream.

Files - I/O

Both istream and ostream provide member functions for repositioning the file-position pointer. These member functions are seekg ("seek get") for istream and seekp ("seek put") for ostream.

Files - I/O

The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be ios::beg (the default) for positioning relative to the beginning of a stream, ios::cur for positioning relative to the current position in a stream or ios::end for positioning relative to the end of a stream.

Files - I/O

```
// position to the nth byte of fileObject (assumes ios::beg)
fileObject.seekg( n );

// position n bytes forward in fileObject
fileObject.seekg( n, ios::cur );

// position n bytes back from end of fileObject
fileObject.seekg( n, ios::end );

// position at end of fileObject
fileObject.seekg( 0, ios::end );
```

Sockets

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

Sockets

Server side

- int sockfd = socket(domain, type, protocol)
- int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);
- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
- int listen(int sockfd, int backlog);
- int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

Sockets

Socket

- **sockfd**: socket descriptor, an integer
- **domain**: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
- **type**: communication type
  - SOCK_STREAM: TCP(reliable, connection oriented)
  - SOCK_DGRAM: UDP(unreliable, connectionless)
- **protocol**: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

Sockets

Bind

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

Sockets

Listen

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

Sockets

Accept

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Sockets

Client

- int sockfd = socket(domain, type, protocol)
- int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Sockets

Connect

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Sockets

Stream Sockets

Stream sockets provide reliable two-way communication similar to when we call someone on the phone. One side initiates the connection to the other, and after the connection is established, either side can communicate to the other.

In addition, there is immediate confirmation that what we said actually reached its destination.

Stream sockets use a Transmission Control Protocol (TCP), which exists on the transport layer of the Open Systems Interconnection (OSI) model. The data is usually transmitted in packets. TCP is designed so that the packets of data will arrive without errors and in sequence.

HTTP, ssh, smtp are all using stream sockets because they are extremely reliable.

Sockets

Datagram Sockets

Communicating with a datagram socket is more like mailing a letter than making a phone call. The connection is one-way only and unreliable.

If we mail several letters, we can't be sure that they arrive in the same order, or even that they reached their destination at all. Datagram sockets use User Datagram Protocol (UDP). Actually, it's not a real connection, just a basic method for sending data from one point to another.

Datagram sockets and UDP are commonly used in networked games and streaming media.