**COMPILING & LINKING**

g++ hello.cpp ---> compiles and generates the executable file

g++ -S hello.cpp ---> compiles but generates assembly source file (hello.s)

g++ -c hello.cpp ---> compiles and assemble (hello.o --> object code)

g++ -o main hello.cpp ---> compiles and links

1. hello.cpp (source file)
2. c++ preprocessor (header files)
3. compiler
4. hello.s
5. assembler
6. hello.o (object code)
7. linker (takes object codes, system files (or system libraries) requirement for OS which provides some functionalities to our program)
8. executable file (hello)

**LINKING**

1) Static

* If we are appending all system libraries (object codes) on to our program, if we link them all together by adding for each program; THIS IS THE STATIC LINKING.
* 1 fridge for each room
  + update knowledge is very hard
* g++ -o main main.o -static (this links libstdc++ to your object code)
  + If you don't use -static, it is dynamic linking.

2) Dynamic

* If we have system library reference code (stub) for all system libraries, if we have only 1 reference for each system library, then we append only this reference and we link this reference with the all program; THIS IS THE DYNAMIC LINKING.
* 1 fridge for all home

*Static library is connected at compile time, while dynamic linking happens at runtime.*

**STRINGS**

**C STRING**

* char t[6];

INVALID FOR C STRINGS

t = "Hello";

* char t[6] = "Hello"; ----> VALID FOR C STRINGS
* strcat concatenates 2 strings.

What are the parameters of main?

int main(int argc, char \*argv[]){...}

parameters are argument count and argument values.

./main.exe "Hello" "user\_id" --> passing parameters while executing

1 2 3

There are 3 arguments.

User should call me with these 3 arguments.

if (argc != 3) cerr << "argc is wrong";

argv[0] = main.exe

argv[1] = Hello

argv[2] = user\_id

**C++ STRING CLASS**

getline:

string line;

cout << “Enter a string: “;

getline(cin, line);

cout << line;

getline prototypes:

1. istream& getline(istream& ms, string& str, char delimeter);
2. istream& getline(istream& ms, string& str);

You can initialize your string as:

1. string s = “Hello”;
2. string s1(“Hello”);

Instead of strcat, we use “+” operator in C++:

string s1 = “Hello”;

string s2 = “Mert”;

s1 + s2 🡪 “HelloMert”

s1 + “ “ + s2 🡪 “Hello Mert”

you can split your string like this

substring method:

* string s1 = “Hello there Mert”, delimeter = “he”, token = s1.substr(0, s1.find(delimeter));
* token 🡪 Hello t
* string s2 = s1.substr(12, 4); 🡪 Start from 12th index and take 4 letters
  + string s2 is “Mert”
  + If you don’t give second parameter, it starts from 12th index and takes until the end (It would be “Mert” again for this example.).

comparison: You can simply compare with “==” operator.

find and length method:

* string str = “Hello there”, str1 = “lo”;
* str.find(str1); 🡪 returns index of the first occurrence (3)
* str.length(); 🡪 returns 11

**STREAM BUFFER**

*//There are input stream (istream) and output stream (ostream) classes in C++.*

STREAMBUFFER ---> Managed by OS

For each OS, there are some streambuffers.

They are taking input or giving output.

Input stream buffers are taking the input from the any input device.

char ch1, ch2;

cin.get(ch1);

cin.get(ch2);

input StreamBuffer

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c |  |

Let’s say keyboard is our input device.

User inputs abc. It goes right to the input streambuffer.

Firstly input streambuffer pointer points the beginning.

cin is input stream OBJECT for C++.

ch1

cin.get(ch1); 🡪 will take a and write it to your variable :

a

Then a is removed from our input streambuffer.

|  |  |  |  |
| --- | --- | --- | --- |
|  | b | c |  |

Now pointer shows the next character of the input streambuffer.

ch2

cin.get(ch2); 🡪 will take b and write it to your variable :

b

After assignment of b, we have this:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | c |  |

Next get will point to the c.

char ch1, ch2;

ch1 = cin.peek();

cin.get(ch2);

cin.putback(‘d’);

cin.get(ch2);

input StreamBuffer (again assume that user enters abc

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c |  |

peek doesn’t change the pointer of the input streambuffer. ch1

a

After ch1= cin.peek(); , ch1 will be a and:

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c |  |

ch2

After cin.get(ch2); , ch2 will be a and:

a

|  |  |  |  |
| --- | --- | --- | --- |
|  | b | c |  |

After cin.putback(‘d’);

|  |  |  |  |
| --- | --- | --- | --- |
| d | b | c |  |

ch2

d

After cin.get(ch2), ch2 will be d and;

|  |  |  |  |
| --- | --- | --- | --- |
|  | b | c |  |