C++ STL, GDB & Makefile Tutorial

DS221_2022

Agenda

1. STL(Standard Template Library)
   - Vector
   - Stack
   - MAP
2. GDB
3. Makefile

STL

Standard Template Library

- Set of C++ template classes to provide common programming data structures and functions.

- Components of STL
    - **Algorithms** {Sorting/Searching}
    - **Containers**
        1. *Sequence Containers {Vector, List, Dequeue}*
        2. *Associative Containers {Set, Map, Multimap}*
        3. *Derived Containers {Stack, Queue, Priority Queue}*
    - **Functions** {Functors: transform()}
    - **Iterators** {used for working upon sequence of values}

Ref: “https://www.geeksforgeeks.org/the-c-standard-template-library-stl/”

Vector

- Can be understood as dynamic arrays.
- Storage is being handled automatically by containers.
- Vector elements are placed in contiguous storage.

vector.begin() : Returns iterators pointing to the first element.

vector.end() : Returns iterators pointing to the last element.

vector.size() : Returns number of elements.

vector.push_back() : push element from back of vector.

vector.pop_back() : pop/remove element from back of vector.

vector.insert() : inserts new elements before the element at specified position.

vector.swap(): swap contents of one vector with another vector of same type.

vector.swap(): swap contents of one vector with another vector of same type.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    // Declaring a vector
    vector<int> vector1;

    cout<<"Initialising a vector with 10
    elements ranging from from 0 to 9"<<endl;
    for(int i=0;i<10;i++){
        vector1.push_back(i);
    }
    //Check Vector size
    cout<<"Vector size = "<<vector1.size()<<
                          endl;

    // Change value at any index
    cout<<"Changing element at 6th index to 50"
    <endl;
    vector1[5]=50;

    // Print Vectors
    cout<<"Printing Vector"<<endl;
    for(auto i = vector1.begin();i!=vector1.end(
        );i++){
        cout<<*i<<"\t";
    }
    cout<<endl;

    // Access values in vector with at()
    cout<<"Vector element at position 1 using
    at() method="<<vector1.at(1)<< endl;

    // Access values in vector with [] operator
    cout << "which is same as accessing using []
    operator "<<vector1[1]<<endl;

    // Use insert() to add value at any
                  specific index
    cout<<"Inseting 25 to 2nd position"<<endl;
    vector1.insert(vector1.begin()+2,25);

    // Calculate new vector size
    cout<<"New vector size = "<<vector1.size()<<
                           endl;
    vector1.pop_back();

    // Calculate new vector size
    cout<<"New vector size = "<<vector1.size()<<
                           endl;

    // Print vector using index
    for(int i=0;i<vector1.size();i++){
        cout<<vector1[i]<<"\t";
    }
    cout<<endl;

    // Sorting a vector with STL algorithm
    cout<<"Sorting vector"<<endl;
    sort(vector1.begin(),vector1.end(),greater<
         int>());
    for(auto i = vector1.begin();i!=vector1.end(
        );i++){
        cout<<*i<<"\t";
    }
    cout << endl;

```

Stack

- Container adaptors with LIFO(Last In First Out) type of working.
- Stack uses encapsulated object of either vector , deque(default) or list.

stack.size() : Returns the size of stack.

stack.empty() : Returns whether the stack is empty.
stack.top() : Returns a reference to the top most element of stack.
stack.push(A) : Adds element A to the top of stack.

stack.pop() : Deletes the top most element of stack.

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

int main () {
    // Initialising a stack in STL, default container - <deque>
    stack<int> mystack;
    // Initialising a stack with a different container
    stack<int, vector<int> > myVectorStack;
    cout<<endl;

    cout<<"----- Initialising Stack -------"<<endl;
    // Check if the stack is currently empty
    cout<<"STACK SIZE - "<<mystack.size()<<endl;
    cout<<"Is Stack Empty ? - "<<(mystack.empty() ? "yes" : "no")<<endl<<endl;

    // Pushing elements onto the stack [This function uses the push_back function of the underlying container]
    cout<<"------ Pushing 10 elments onto the stack ---------"<<endl;
    for(int x = 0; x < 10; x++)  {
        cout<<"Pushing.. value = "<<x+1<<endl;
        mystack.push(x+1);
    }
    cout<<endl;

    cout<<"Is Stack Empty ? - "<<(mystack.empty() ? "yes" : "no")<<endl;

    // Getting the size of stack in STL
    cout<<"STACK SIZE - "<<mystack.size()<<endl;

    // Getting the top of stack in STL Stack object
    cout<<"CURRENT STACK TOP - "<<mystack.top()<<endl<<endl;

    // Popping elements from the stack;
    cout<<"------ Popping 5 elements from the stack ---------"<<endl;
    for(int x = 0; x < 5; x++) {
        cout<<"Popping... value = "<<mystack.top()<<endl;
        mystack.pop();
    }
    cout<<endl;

    cout<<"Is Stack Empty ? - "<<(mystack.empty() ? "yes" : "no")<<endl;
    cout<<"STACK SIZE - "<<mystack.size()<<endl;
    cout<<"CURRENT STACK TOP - "<<mystack.top()<<endl<<endl;

    cout<<"--------------- Clearing the Stack -----------------"<<endl;
    while(!mystack.empty()) {
        cout<<"Popping... value = "<<mystack.top()<<endl;
        mystack.pop();
    }
    cout<<endl;

    cout<<"Is Stack Empty ? - "<<(mystack.empty() ? "yes" : "no")<<endl;
    cout<<"STACK SIZE - "<<mystack.size()<<endl;
    cerr<<"CURRENT STACK TOP (Will give a segmentation fault since stack is empty) - "<<mystack.top()<<endl;

    return 0;
}
```

Map

- Maps are associative containers that store elements in a mapped fashion.
- map (internally implemented with BBT) O(log n) vs unordered_map (internally implemented with hashing) O(1)

map.begin() :Returns an iterator to the first element in map.

map.end() : Returns an iterator to the last element of the map.

map.size() : Returns the number of elements in map.

map.max_size() : Returns the maximum number of elements that the map can hold.

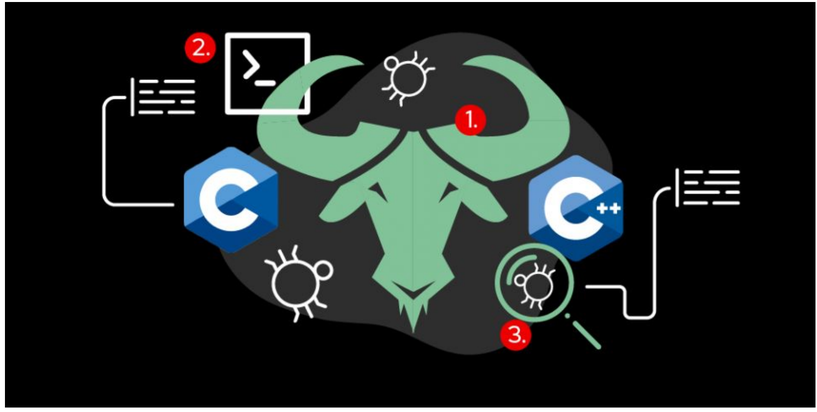
map.insert() : Adds new element to the map.

```cpp
#include <iostream>
#include <map>
#include <utility>

using namespace std;

typedef map<char,int> Map;

void print_map(Map mp) {
    cout<<endl;
    cout<<"- MAP -"<<endl;
    // Using begin() and end() function to traverse the map
    for(Map::iterator it = mp.begin(); it != mp.end(); it++) {
        cout<<it->first<<" --> "<<it->second<<endl;
    }
    cout<<"Map Size - "<<mp.size()<<endl<<endl;;
}

int main() {

    map<char, int> mymap;

    cout<<"------ Initialising Map ---------"<<endl;
    // checking the map size
    cout<<"MAP SIZE - "<<mymap.size()<<endl;
    // checking if the map is empty
    cout<<"Is Map empty ? - "<<(mymap.empty() ? "yes" : "no")<<endl<<endl;

    cout<<"------ Inserting values into map --------"<<endl;
    cout<<"Inserting.."<<endl<<endl;

    // Inserting with pair
    mymap.insert(pair<char, int>('a', 10));
    mymap.insert(pair<char, int>('b', 20));

    // Inserting with make_pair
    mymap.insert(make_pair('c', 30));
    mymap.insert(make_pair('d', 40));

    // Other way
    mymap['e']=20;

    print_map(mymap);
    cout<<"------ Erasing values from map --------"<<endl;
    cout<<"Erasing entry with key (a).."<<endl;

    // erasing values from the map
    mymap.erase('a');
    cout<<"Erasing entry with key (c).."<<endl;
    mymap.erase('c');

    print_map(mymap);

    cout<<"------ Clearing Map --------"<<endl;
    cout<<"Clearing..."<<endl;
    // clearing all entries
    mymap.clear();

    print_map(mymap);
    cout<<endl;

    return 0;
}
```

Ref : “ https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/”

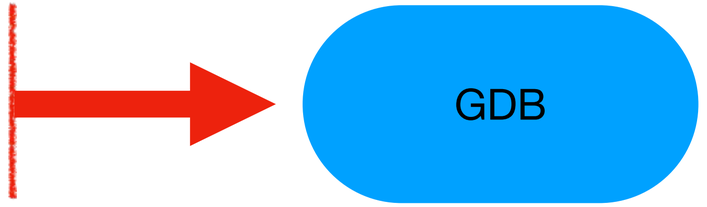
GDB (GNU Debugger)

Debugger for languages: C/C++/Fortran/OpenCL



Credit: “Redhat Developers”

- CLI(Command Line Interface) `$gdb executable`
- For GUI, seamlessly integrated to popular IDE’s (VSCode, Eclipse, Emacs).
- Debugging Techniques:
    - Reactive : Actions we take after encountering bug. `Debugging Tools`
    - Preemptive : Actions we can take before encountering a bug. `checking/comparing values`
- Programming Errors:
    - Build Errors: `error: expected ';' before '}' token`
    - Runtime Errors: `warning: division by zero [-Wdiv-by-zero]`
    - Logic Errors: `result = 20` {Expected : result = 25}



Ref: “https://uchicago-cs.github.io/debugging-guide/"

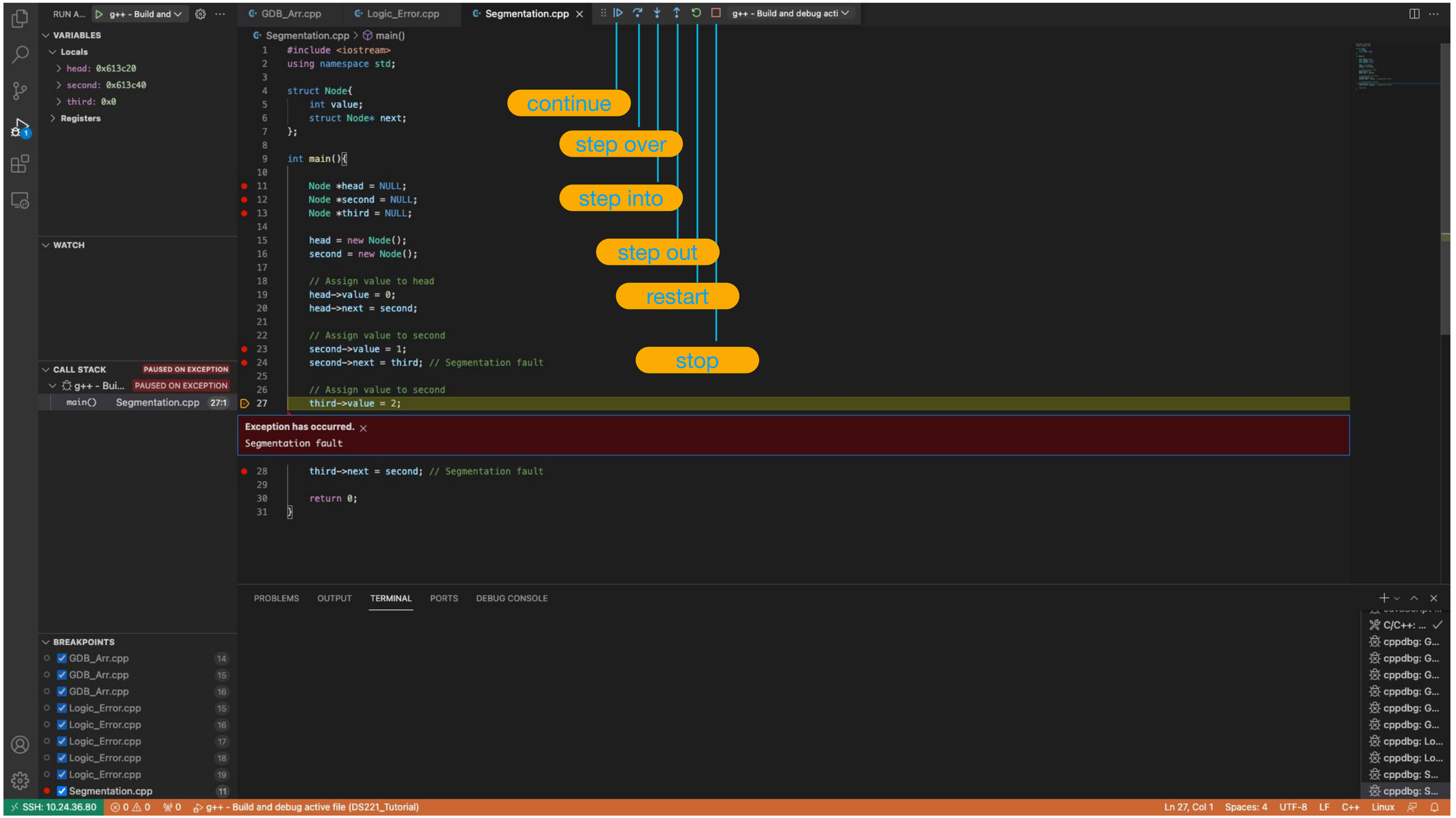
GDB (GNU Debugger)

Useful Commands

- Help (gdb) help
- Start program (gdb) run
- Set breakpoint (gdb) break line_number
- Delete all breakpoint (gdb) delete
- Execute current and step to next line (gdb) step
- Continue program after breakpoint (gdb) continue
- List source code from current line number (gdb) list
- (gdb) print expression

Ref: “https://web.eecs.umich.edu/~sugih/pointers/summary.html”

GDB (GNU Debugger) VSCode Interface



GDB (GNU Debugger)

Logic Error (Ex: Logical Error)

- Logics can cause program to produce unexpected results.
- Program terminates normally.
- Ex: Calculating Average of two float values.

```
1  #include <iostream>
2  using namespace std;
3
4  float avg_floatE(float a, float b){
5      return a + b / 2 ;
6  }
7
8  float avg_float(float a, float b){
9      return (a + b )/ 2 ;
10 }
11
12 int main(){
13     float a = 10.5;
14     float b  = 10.5;
15     cout << "Average float Values " << endl;
16     float c  = avg_floatE(a,b);
17     cout << c << endl;
18     float d = avg_float(a,b);
19     cout << d << endl;
20     return 0;
21 }
```

GDB (GNU Debugger)

Runtime Error (Ex: Segmentation fault)

- Most common condition, causes program to crash
- Caused when program is trying to read/write an illegal memory location.
- Ex: illegal memory access in Singly Linked list.

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node{
5      int value;
6      struct Node* next;
7  };
8
9  int main(){
10
11     Node *head = NULL;
12     Node *second = NULL;
13     Node *third = NULL;
14
15     head = new Node();
16     second = new Node();
17
18     // Assign value to head
19     head->value = 0;
20     head->next = second;
21
22     // Assign value to second
23     second->value = 1;
24     second->next = third;
25
26     // Assign value to second
27     third->value = 2; // Segmentation fault
28     third->next = second;
29
30     return 0;
31 }
```

Makefile

- A makefile is **a text file that contains instructions for how to compile and link (or build) a set of source code files**.
- Provide a way for separate compilation.
- *Makefile* structure:

```
target:  dependencies
[tab]    action                    #shell commands
```

- Naming: ***makefile*** or ***Makefile*** are standard, or we can use "***make -f filename***".
- Running make: "***make***", or "***make target_name***" – if you want to make a target that is not the first one.

https://www.geeksforgeeks.org/makefile-in-c-and-its-applications/

Thanks!

jyotshnar@iisc.ac.in

https://github.com/Ghanshyamchandra74/DS221_2021

Slides from **Ghanshyam Chandra** (ghanshyamc@iisc.ac.in)

References

1. https://www.cplusplus.com/reference/stl/
2. https://en.cppreference.com/w/cpp/container
3. https://www.topcoder.com/thrive/articles/Power%20up%20C++%20with%20the%20Standard%20Template%20Library%20Part%20One
4. https://www.hackerearth.com/practice/notes/c-stls-when-to-use-which-stl/
5. https://www.geeksforgeeks.org/makefile-in-c-and-its-applications/