Stack Unwinding in C++

Difficulty Level : Medium • Last Updated : 25 Nov, 2021

Stack Unwinding is the process of removing function entries from function call stack at run time. The local objects are destroyed in reverse order in which they were constructed.

Stack Unwinding is generally related to Exception Handling. In C++, when an exception occurs, the function call stack is linearly searched for the exception handler, and all the entries before the function with exception handler are removed from the function call stack. So, exception handling involves Stack Unwinding if an exception is not handled in the same function (where it is thrown). Basically, Stack unwinding is a process of calling the destructors (whenever an exception is thrown) for all the automatic objects constructed at run time.

For example, the output of the following program is:

CPP

```cpp
// CPP Program to demonstrate Stack Unwinding
#include <iostream>
using namespace std;

// A sample function f1() that throws an int exception
void f1() throw(int)
{
    cout << "\n f1() Start ";
    throw 100;
    cout << "\n f1() End ";
}

// Another sample function f2() that calls f1()
void f2() throw(int)
{
    cout << "\n f2() Start ";
    f1();
    cout << "\n f2() End ";
}

// Another sample function f3() that calls f2() and handles
// exception thrown by f1()
void f3()
{
    cout << "\n f3() Start ";
    try {
        f2();
    }
    catch (int i) {
        cout << "\n Caught Exception: " << i;
    }
    cout << "\n f3() End";
}

// Driver Code
int main()
{
    f3();

    getchar();
    return 0;
}
```

Output

```
f3() Start
f2() Start
f1() Start
Caught Exception: 100
f3() End
```

Explanation:

AD

- When f1() throws exception, its entry is removed from the function call stack, because f1() doesn't contain exception handler for the thrown exception, then next entry in call stack is looked for exception handler.
- The next entry is f2(). Since f2() also doesn't have a handler, its entry is also removed from the function call stack.

- The next entry in the function call stack is f3(). Since f3() contains an exception handler, the catch block inside f3() is executed, and finally, the code after the catch block is executed.

Note that the following lines inside f1() and f2() are not executed at all.

```
cout<<"\n f1() End ";  // inside f1()

cout<<"\n f2() End ";  // inside f2()
```

If there were some local class objects inside f1() and f2(), destructors for those local objects would have been called in the Stack Unwinding process.

> ***Note:*** *Stack Unwinding also happens in Java when exception is not handled in same function.*

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

61

Related Articles

1. How to detect Stack Unwinding in a Destructor in C++?
2. stack empty() and stack size() in C++ STL
3. stack swap() in C++ STL
4. stack top() in C++ STL
5. Stack push() and pop() in C++ STL
6. stack emplace() in C++ STL
7. Implementing Stack Using Class Templates in C++
8. How to implement a Stack using list in C++ STL
9. Stack-buffer based STL allocator