CS246—Known C++20 Issues

September 4, 2023

1 Compilation errors

1. Importing `<iostream>` before `<sstream>` when defining a class that uses dynamic memory allocation to initialize a data field causes the compilation error:

```
$ g++20 t.cc
t.cc: In destructor ‘C::~C()’:
t.cc:8:26: internal compiler error: in build_op_delete_call, at cp/call.c:7143
    8 |         ~C() { delete [] arr;}
      |                          ^~~
0x7f33d1316d8f __libc_start_call_main
        ../sysdeps/nptl/libc_start_call_main.h:58
0x7f33d1316e3f __libc_start_main_impl
        ../csu/libc-start.c:392
Please submit a full bug report,
```

Doesn't compile	*Compiles*
`import <iostream>;`	`import <sstream>;`
`import <sstream>;`	`import <iostream>;`
`struct C {`	`struct C {`
`    int * arr;`	`    int * arr;`
`    C() : arr{new int[5]} {}`	`    C() : arr{new int[5]} {}`
`    ~C() { delete [] arr;}`	`    ~C() { delete [] arr;}`
`};`	`};`
`int main() {`	`int main() {`
`}`	`}`

2. Until the compiler is fixed, there is no current workaround for the compilation error produced by deleting an (`istream *`).

```
$ cat t.cc
import <iostream>;
using namespace std;

int main() {
    istream * in;
    delete in;
}
$ g++20 t.cc
t.cc: In function ‘int main()’:
t.cc:8:12: error: invalid use of non-static member function ‘std::basic_istream<_CharT, _Traits>::~bas
```

```
    8 |     delete in;
      |            ~~
In file included from /usr/include/c++/11/iostream:40,
of module /usr/include/c++/11/iostream, imported at t.cc:3:
/usr/include/c++/11/istream:103:7: note: declared here
  103 |       ~basic_istream()
      |       ^
```

2 Run-time errors

1. Initializing a `std::string` with a (`const char *`) sometimes causes a segmentation fault when the resulting string is used to initialize a new string object using a copy constructor:

   ```
   $ cat t.cc
   import <iostream>;
   import <string>;
   using namespace std;

   int main() {
     string op{"hello"};
     string newstr = op;
   }
   $ ./a.out
   Segmentation fault
   $ gdb ./a.out
   (gdb) run
   0x0000555555556434 in std::__cxx11::basic_string<char, std::char_traits<char>, s
   import <string>;
   td::allocator<char> >::_Alloc_hider::_Alloc_hider (this=0x0, __dat=0x10 <error: Cannot access memory a
   168               : allocator_type(std::move(__a)), _M_p(__dat) { }
   ```

   Can be fixed by either removing the importation of `<iostream>` (not very helpful) or using copy assignment instead:

   ```
   import <string>;
   import <iostream>;
   using namespace std;

   int main() {
     string op{ "hello" };
     string newstr;
     newstr=op;
   }
   ```

2. Assigning a (`char*`) to an empty `std::string` will sometimes result in a segmentation fault in the library code. If this occurs, wrap your (`char*`) value/variable with a `std::string` constructor call.

May fault

```
import <string>;
using namespace std;

int main( int argc, char * argv[] ) {
    string arg;
    for ( int i = 0; i < argc; ++i ) {
        arg = argv[i];
    }
}
```

Works reliably

```
import <string>;
using namespace std;

int main( int argc, char * argv[] ) {
    string arg;
    for ( int i = 0; i < argc; ++i ) {
        arg = string{argv[i]};
    }
}
```

3. Passing (or returning) a string "by value" sometimes causes a segmentation fault. Try changing the type to be a constant reference instead.

May fault

```
import <iostream>;
import <string>;
using namespace std;

//  test-harness operators
enum Op {NONE, CONSTRUCT, DELETE, READ,
    PRINT, POP_BACK, PUSH_BACK};

// converts a one-character input command into
// its corresponding test-harness operator
Op convertOp(string opStr) {
  switch(opStr[0]) {
   case 'c': return CONSTRUCT;
   case 'd': return DELETE;
   case 'r': return READ;
   case 'p': return PRINT;
   case 'b': return POP_BACK;
   case 'B': return PUSH_BACK;
   default: return NONE;
  }
}

int main() {
    string command;
    while ( cin >> command ) {
        switch( convertOp( command ) ) {
            case CONSTRUCT:
                // ...
        } // switch
    } // while
} // main
```

Works reliably

```
import <iostream>;
import <string>;
using namespace std;

//  test-harness operators
enum Op {NONE, CONSTRUCT, DELETE, READ,
    PRINT, POP_BACK, PUSH_BACK};

// converts a one-character input command into
// its corresponding test-harness operator
Op convertOp(const string & opStr) {
  switch(opStr[0]) {
   case 'c': return CONSTRUCT;
   case 'd': return DELETE;
   case 'r': return READ;
   case 'p': return PRINT;
   case 'b': return POP_BACK;
   case 'B': return PUSH_BACK;
   default: return NONE;
  }
}

int main() {
    string command;
    while ( cin >> command ) {
        switch( convertOp( command ) ) {
            case CONSTRUCT:
                // ...
        } // switch
    } // while
} // main
```

4. Using `std::getline` to read into a default-initialized `std::string` will sometimes segment fault. Explicitly initializing the string seems to avoid the problem fairly reliably.

May fault

```
string line;
while ( getline(cin, line) ) { ... }
```

Works reliably

```
string line = "";
while ( getline(cin, line) ) { ... }
```

5. String concatenation segment faults when system libraries are compiled in one order, and not in another.

May fault

```
ctkierst@ubuntu2204-004: ~ $ rm -fr gcm.cache/
ctkierst@ubuntu2204-004: ~ $ g++20h string iostream
ctkierst@ubuntu2204-004: ~ $ g++20 t.cc
ctkierst@ubuntu2204-004: ~ $ ./a.out
Segmentation fault
ctkierst@ubuntu2204-004: ~ $ cat t.cc
import <iostream>;
import <string>;
using namespace std;

string intToString() {
        string hi = "hi";
        hi += "hello";
        return hi;
}

int main(){
        cout << intToString() << endl;
        cout << intToString().length() << endl;
}
```

Works reliably

```
ctkierst@ubuntu2204-004: ~ $ rm -fr gcm.cache/
ctkierst@ubuntu2204-004: ~ $ g++20h iostream string
ctkierst@ubuntu2204-004: ~ $ g++20 t.cc
ctkierst@ubuntu2204-004: ~ $ ./a.out
hihello
7
ctkierst@ubuntu2204-004: ~ $ cat t.cc
import <iostream>;
import <string>;
using namespace std;

string intToString() {
        string hi = "hi";
        hi += "hello";
        return hi;
}

int main(){
        cout << intToString() << endl;
        cout << intToString().length() << endl;
}
```