Codillon: Always-Executable Assembly Editing

Emma Sudo[1], Jiahao Zhang[2], Trip Master[1], Keith Winstein[1]

[1]Computer Science Department, Stanford University

[2]School of Electronics Engineering and Computer Science, Peking University

Stanford Computer Science

Motivation: Frustrating First Steps

One line in this 15-line C++ Program causes the compiler to output hundreds of errors… can you spot it?

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

template <typename T, int N>
struct Bomb {
    using value_type = typename T::type;

    Bomb<value_type, N-1> next;
};

template <typename T>
struct Bomb<T, 0> {
};

int main() {
     Bomb<std::vector<int>, 20> b;
     (void) b;
}
```

```
error.cpp:6:36: error: no type named 'type' in 'std::vector<int>'
    6 |     using value_type = typename T::type;
      |                                  ~~~~~~^~~~
error.cpp:16:33: note: in instantiation of template class 'Bomb<std::vector<int>, 20>' requested here
   16 |      Bomb<std::vector<int>, 20> b;
      |                                 ^
error.cpp:6:39: error: type 'int' cannot be used prior to '::' because it has no members
    6 |     using value_type = typename T::type;
      |                                       ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 19>' requested here
    8 |     Bomb<value_type, N-1> next;
      |                           ^
error.cpp:16:33: note: in instantiation of template class 'Bomb<std::vector<int>, 20>' requested here
   16 |      Bomb<std::vector<int>, 20> b;
      |                                 ^
error.cpp:6:39: error: type 'int' cannot be used prior to '::' because it has no members
    6 |     using value_type = typename T::type;
      |                                       ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 18>' requested here
    8 |     Bomb<value_type, N-1> next;
      |                           ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 19>' requested here
error.cpp:16:33: note: in instantiation of template class 'Bomb<std::vector<int>, 20>' requested here
   16 |      Bomb<std::vector<int>, 20> b;
      |                                 ^
error.cpp:6:39: error: type 'int' cannot be used prior to '::' because it has no members
    6 |     using value_type = typename T::type;
      |                                       ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 17>' requested here
    8 |     Bomb<value_type, N-1> next;
      |                           ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 18>' requested here
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 19>' requested here
error.cpp:16:33: note: in instantiation of template class 'Bomb<std::vector<int>, 20>' requested here
   16 |      Bomb<std::vector<int>, 20> b;
      |                                 ^
error.cpp:6:39: error: type 'int' cannot be used prior to '::' because it has no members
    6 |     using value_type = typename T::type;
      |                                       ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 16>' requested here
    8 |     Bomb<value_type, N-1> next;
      |                           ^
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 17>' requested here
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 18>' requested here
error.cpp:8:27: note: in instantiation of template class 'Bomb<int, 19>' requested here
error.cpp:16:33: note: in instantiation of template class 'Bomb<std::vector<int>, 20>' requested here
   16 |      Bomb<std::vector<int>, 20> b;
      |                                 ^
error.cpp:6:39: error: type 'int' cannot be used prior to '::' because it has no members
```

Compilers output a confusing stream of error messages in response to syntax and validity errors.

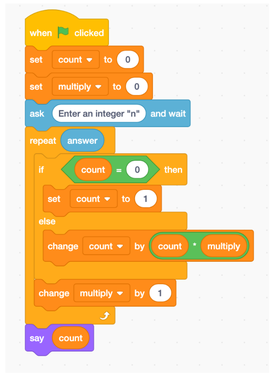
These messages make learning to program unnecessarily difficult, but this is how students are currently taught at the introductory level.

Researchers have created pedagogical editors in the past. We are building on this work to create a new pedagogical development environment for an introductory CS-EE course.

Prior Work

Scratch (Mitchel Resnick and Yasmin Kafai 2003)

Scratch is a block-based structure editor that is great for basic programs because all editor states are executable. However, complex programs require many blocks, which makes going from one state to another difficult (high viscosity). For example, the factorial function is 5 lines long in C++ but requires 17 blocks in Scratch.



Factorial Function in Scratch

Javardise (André L. Santos 2020)

Javardise is a structure editor for a subset of Java. Javardise only allows edit actions that ensure the program conforms to this modified Java's syntax. Semantic errors are not fixed, so the editor cannot guarantee an always-executable state. Like Scratch, it can be difficult to get from one state to another.

```
public static long factorial ( int n ) {
  if ( n == 0 ) {
    return 1 ;
  }
  else {

    return n *  factorial ( n - 1.0 ) ;
  }
}
```

incompatible types: possible lossy conversion from double to int

Factorial Function with Type Error in Javardise

Hazel (Cyrus Omar 2017)

Hazel is a structure editor for a co-designed functional programming language. It guarantees that every incomplete program is executable using "typed-holes." Compared with Hazel, we think there remains an opportunity to make error states more understandable.

```
let f = fun n ->
  if n == 0 then
    1
  else
    n * f(n-1)

in f(2)

≡ 2

Γ EXP / Number literal : Int
```

Correct Factorial Function in Hazel

```
let f = fun n ->
  if x == 0 then
    1
  else
    n * f(n-1)

in f(2)

≡ if x == 0 then
    1
  else
    2 * (<f+>)(2 - 1)

Γ EXP / Variable reference x not found
```

Factorial Function with Symbolic Identifier Error in Hazel

***Codillon* is a text editor where every incomplete program is executable and errors are closely isolated to their origin.**

***Codillon* makes learning to program easier by allowing students to focus on problem-solving instead of syntax.**

Approach

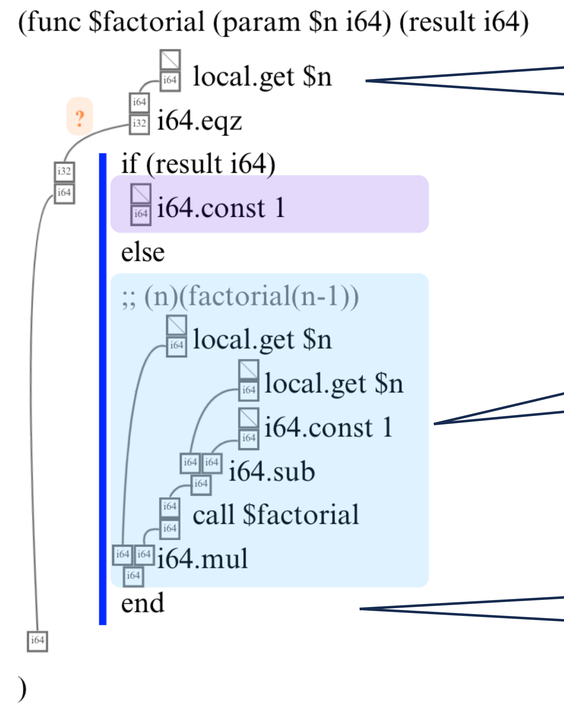
In *Codillon*, students write programs in a modified WebAssembly (Wasm) text format. Wasm is a typed assembly language for a virtual stack machine.

We achieve an always-executable state by enforcing two rules.

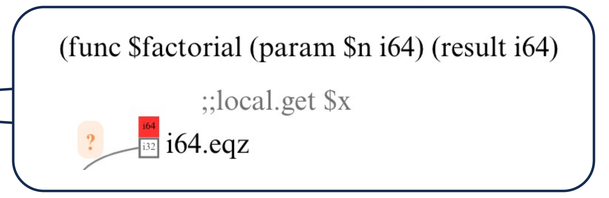
First, each line must be well-formed. Malformed lines are commented out until they are fixed.
Second, we always show the data flow of the program next to the instructions. When there are data flow mismatches or holes, we patch the error behind the scenes, mark it in the UI, and keep evaluating the rest of the program.

Editing in Codillon

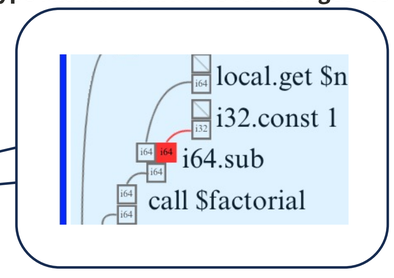
Correct Factorial Function in *Codillon*



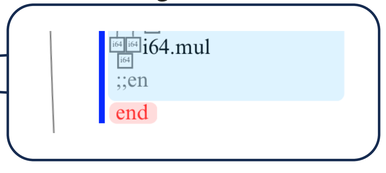
Symbolic Identifier Error: User changes "n" to "x"



Type Mismatch Error: User changes "i64" to "i32"



Frame Matching Error: User deletes "d" in "end"



Further Work

- Is it possible to circumvent certain run-time errors?
- Display number of execution steps
- Debugging slider to simulate step-by-step execution

References

- Omar et al. (2017) - *Toward Semantic Foundations for Program Editors* (SNAPL)
- Resnick et al. (2009) - *Scratch: Programming for All* (Communications of the ACM)
- Santos (2020) - *Javardise: A Structured Code Editor for Programming Pedagogy in Java* (Programming)