TERM PAPER ON

COMPILER ERROR MESSAGES: WHAT THEY SAY AND WHAT THEY MEAN

BY

GROUP 9

COMPILER CONSTRUCTION

(COSC 408)

SUBMITTED TO DR.K.A BAKARRE

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

GROUP 9 MEMBERS

S/N	NAMES	REG NO	SIGN
1.	SADIQ KHALIL ABUBAKAR	U12CS2019	______
2.	HARUNA ABDULLAHI KAWO	U12CS2022	______
3.	HARUNA YUSUF MURTALA	U12CS2028	______
4.	YAHAYA AUWALU	U12CS2023	______
5.	UMAR ADAM	U12CS2018	______
6.	IDRIS ZAINAB SALEH	U12CS2024	______
7.	SULEIMAN HABIBA YUSUF	U12CS2026	______
8.	MUKHTAR UMMA RUMA	U12CS2027	______
9.	JIBRIL MUSA KHALIL	U12CS2032	______
10.	DANYARO AMINU IBRAHIM	U12CS2033	______
11.	ILYASU KHADIJA ABUBAKAR	U12CS2017	______
12.	MARYAM IDRIS	U12CS2025	______
13.	MARYAM DAHIRU	U12CS2029	______
14.	AHMED AMINA	U12CS2035	______
15.	ISAH ABDULRAHMAN ABDULKAD	U12CS2036	______
16.	ALIYU KABIRU	U12CS2037	______

Abstract

This paper focuses on compiler error messages: what they say and what they mean.

A Programmer, no matter his level of expertise makes errors, programmers are humans and that entails that programmers make errors. Profound errors are made in code snippets and compilers generate and complain about these errors in programs, compiler error messages are cryptic in nature, meaning they are difficult to understand and resolve.

There have been advances in compilation speed and the speed of the resulting executable program and nobody is paying attention to this cryptic nature of error messages which affects programmers profoundly.

In this paper, we will look at compiler error messages, challenges novice programmers face in learning how to program, the paper also looks at actual compiler error messages and how they supposed to have been described to help novice programmers in debugging programs, suggests what should help novice programmers, some errors java programmers make, and principle of compiler error message design.

1. Introduction

If we look at today's state-of-the-art compilers, research and advances in the field of compiler design and construction focuses on implementing new features of a programming language, or developing compilers for new programming languages. There are also efforts to improve optimization techniques, so that compiled code uses less space or runs faster. Other projects aim to develop compilers that run faster. All these are commendable and interesting topics of study, but it is striking that there is little concern on devising techniques to help the user of the system (the programmer) to do their job properly. This issue is ignored, not only in compiler textbooks (no matter how recent or advanced they may be), but also, and most importantly, by current research directions.

In 1999, Alexandrescu, a C++ expert and developer, wrote an open letter addressed to C++ compiler vendors and the C++ community with a short proposal "to make diagnostic messages generated by C++ compilers easier to read and understand in the presence of templates". Three compiler experts from three different companies were offered the opportunity to respond to Alexandrescu's letter. They admit the problem and one of the responders, Jonathan Caves, from Microsoft Corporation, acknowledged that it is a sad but true fact that diagnostics are one of the most overlooked aspects of compiler development and gave three main reasons for this. First, it is the historic concern about memory requirements; compilers' performance would degrade if information required for better error messages were kept. Second, compiler developers are obviously familiar with the compiler they build, and they are the ones who write the error messages themselves by using the language specification to provide a succinct description for an error situation. Unfortunately, their messages are unintelligible for the average user. In this sense, this situation in compiler development is not much different to the general case of designers who end up knowing their product so well that they cannot think the same way as someone who does not know what they know. And the third reason has to do with how new compiler releases are planned and developed, where the list of new features are prioritized, but "better error messages" is always in the lowest priority group and the rarely addressed.

The rest of the paper is organized as follows; section 2.1 compiler error messages, section 3.1 is on challenges novice programmers face, section 4.1 is on errors java programmers make, section 5.1 talks about actual compiler error messages, section 6.1 is on helping novice programmers in understanding error messages and section 7.1 concludes the paper.

2.1 Compiler Error Messages

Compilation error refers to a state when a compiler fails to compile a piece of computer program source code, either due to errors in the code, or, more unusually, due to errors in the compiler itself.

A compilation error message often helps programmers in debugging the source code for possible errors.

To be useful, a compiler should detect all errors in the source code and report them to the user. These errors could be:

- Lexical errors: e.g., badly formed identifiers or constants, symbols which are not part of the language, badly formed comments, etc.
- Syntactic errors: chains of syntactic units that do not conform to the syntax of the source language.
- Semantic errors: e.g., operations conducted on incompatible types, undeclared variables, double declaration of variable, reference before assignment, etc.
- Run-time errors: errors detectable solely at run time, pointers with null value or whose value is outside allowed limits, or indexing of vectors with unsuitable indices, etc. To be useful, a compiler should detect all errors in the source code and report them to the user.

These errors are further grouped into two, i.e. compilation and execution errors:

Compilation Errors

Lexical errors: e.g., badly formed identifiers or constants, symbols which are not part of the language, badly formed comments, etc.

Syntactic errors: chains of syntactic units that do not conform to the syntax of the source language.

Semantic errors: e.g., operations conducted on incompatible types, undeclared variables, double declaration of variable, reference before assignment, etc.

Execution Errors

Run-time errors: errors detectable solely at run time, pointers with null value or whose value is outside allowed limits, or indexing of vectors with unsuitable indices, etc.

3.1 Challenges Novice Programmers Face

When new programmers are starting out with programming, it's easy to run into problems that make them wonder how anyone has ever managed to write a computer program. But the fact is, just about everyone else who's learned to code has had that experience and wondered the same thing, when they were starting out.

Learning to program is hard enough, but it's easy to get tripped up before you even begin. First you need to choose a programming language, a compiler and a programming tutorial that covers the language you chose and that works with the compiler that you set up. This is all very complicated, and all before you even start to get to the fun parts.

The following are challenges novice programmers face:

Compiler Error Messages

This may seem like a small thing, but because most beginners aren't familiar with the strictness of the format of the program (the syntax); beginners tend to run into lots of complaints generated by the compiler. Compiler errors are notoriously cryptic and verbose, and by no means were written with newbies in mind.

There are a few basic principles you can use to navigate the thicket of messages. First, often times a single error causes the compiler to get so confused that it generates dozens of messages, always start with the first error message. Second, the line number is a lie. Well, maybe not a lie, but you can't trust it completely. The compiler complains when it first realizes there is a problem, not at the point where the problem actually occurred. However, the line number does indicate the last possible line where the error could have occurred, the real error may be earlier, but it will never be later.

Debugging

Debugging is a critical skill, but most people aren't born with a mastery of it. Debugging is hard for a few reasons; first, it's frustrating. You just wrote a bunch of code, and it doesn't work even though you're pretty sure it should. Second, it can be tedious; debugging often requires a lot of effort to narrow in on the problem, and until you have some practice, it can be hard to efficiently narrow it down. One type of problem, segmentation faults, are a particularly good example of this, many programmers try to narrow in on the problem by adding in print statements to show how far the program gets before crashing, even though the debugger can tell them exactly where the problem occurred. Which actually leads to the last problem, debuggers are yet another confused, difficult to set up tool, just like the compiler. If all you want is your program to work, the last thing you want to do is go set up another tool just to find out why.

Designing a Program

When you're just starting to program, design is a real challenge. Knowing how to think about programming is one piece, but the other piece is knowing how to put programs together in a way that makes it easy to modify them later. Ideas like "commenting your code", "encapsulation and data hiding" and "inheritance" don't really mean anything when you haven't felt the pain of not having them. The problem is that program design is all about making things easier for your future self, sort of like eating your vegetables. Bad designs make your program inflexible to future changes, or impossible to understand after you've written. Frequently, bad design exposes too many details of how something is implemented, so that every part of the program has to know all the details of each other section of the program.

4.1 Errors Java Programmers Make

Whether you program regularly in Java, and know it like the back of your hand, or whether you're new to the language or a casual programmer, you'll make mistakes. It's natural, it's human, and guess what? You'll more than likely make the same mistakes that others do, over and over again. Here's a list of errors that we all seem to make at one time or another, how to spot them, and how to fix them.

Null pointers

Null pointers are one of the most common errors that Java programmers make. Compilers can't check this one for you, it will only surface at runtime, and if you don't discover it, your users certainly will. When an attempt to access an object is made, and the reference to that object is null, a NullPointerException will be thrown. The cause of null pointers can be varied, but generally it means that either you haven't initialized

an object, or you haven't checked the return value of a function. Many functions return null to indicate an error condition, but unless you check your return values, you'll never know what's happening. Since the cause is an error condition, normal testing may not pick it up, which means that your users will end up discovering the problem for you. If the API function indicates that null may be returned, be sure to check this before using the object reference.
Another cause is where your initialization has been sloppy, or where it is conditional.

For example, examine the following code, and see if you can spot the problem.

```
public static void main(String args[])

{

// Accept up to 3 parameters

String[] list = new String[3];

int index = 0;

while ( (index < args.length) && ( index < 3 ) )

{

list[index++] = args[index];

}

// Check all the parameters

for (int i = 0; i < list.length; i++)

{

if (list[i].equals "-help")

{

// .........

}

else

if (list[i].equals "-cp")

{

// .........
}

// else .....

}

}
```

This code (while a contrived example), shows a common mistake. Under some circumstances, where the user enters three or more parameters, the code will run fine. If no parameters are entered, you'll get a NullPointerException at runtime. Sometimes your variables (the array of strings) will be initialized, and other times they won't. One easy solution is to check before you attempt to access a variable in an array that it is not equal to null.

Capitalization errors

This is one of the most frequent errors that we all make. It's so simple to do, and sometimes one can look at an uncapitalized variable or method and still not spot the problem. I myself have often been puzzled by these errors, because I recognize that the method or variable does exist, but don't spot the lack of capitalization. While there's no silver bullet for detecting this error, you can easily train yourself to make less of them. There's a very simple trick you can learn: - all methods and member variables in the Java API begin with lowercase letters all methods and member variables use capitalization where a new word begins e.g. getDoubleValue() If you use this pattern for all of your member variables and classes, and then make a conscious effort to get it right, you can gradually reduce the number of mistakes you'll make. It may take a while, but it can save some serious head scratching in the future.

Forgetting that Java is zero-indexed

If you've come from a C/C++ background, you may not find this quite as much a problem as those who have used other languages. In Java, arrays are zero-indexed, meaning that the first element's index is actually 0. Confused? Let's look at a quick example.

```
// Create an array of three strings

String[] strArray = new String[3];

// First element's index is actually 0

strArray[0] = "First string";

// Second element's index is actually 1

strArray[1] = "Second string";

// Final element's index is actually 2

strArray[2] = "Third and final string";
```

In this example, we have an array of three strings, but to access elements of the array we actually subtract one. Now, if we were to try and access strArray[3], we'd be accessing the fourth element. This will cause an ArrayOutOfBoundsException to be thrown ; the most obvious sign of forgetting the zero-indexing rule. Other areas where zero-indexing can get you into trouble are with strings. Suppose you wanted to get a character at a particular offset within a string. Using the String.charAt(int) function you can look this information up - but under Java, the String class is also zero-indexed. That means than the first character is at offset 0, and second at offset 1. You can run into some very frustrating problems unless you are aware of this - particularly if you write applications with heavy string processing. You can be working on the wrong character, and also throw exceptions at runtime. Just like the ArrayOutOfBoundsException, there is a string equivalent. Accessing beyond the bounds of a String will cause a StringIndexOutOfBoundsException to be thrown, as demonstrated by this example.

```
public class StrDemo

{

public static void main (String args[])

{

String abc = "abc";

System.out.println ("Char at offset 0 : " + abc.charAt(0) );

System.out.println ("Char at offset 1 : " + abc.charAt(1) );

System.out.println ("Char at offset 2 : " + abc.charAt(2) );

// This line should throw a StringIndexOutOfBoundsException

System.out.println ("Char at offset 3 : " + abc.charAt(3) );

}

}
```

Note too, that zero-indexing doesn't just apply to arrays, or to Strings. Other parts of Java are also indexed, but not always consistently. The java.util.Date, and java.util.Calendar classes start their months with 0, but days start normally with 1.

Confusion over passing by value, and passing by reference
This can be a frustrating problem to diagnose, because when you look at the code, you might be sure that it’s passing by reference, but find that it’s actually being passed by value. Java uses both, so you need to understand when you're passing by value, and when you're passing by reference. When you pass a primitive data type, such as a char, int, float, or double, to a function

then you are passing by value. That means that a copy of the data type is duplicated, and passed to the function. If the function chooses to modify that value, it will be modifying the copy only. Once the function finishes, and control is returned to the returning function, the "real" variable will be untouched, and no changes will have been saved. If you need to modify a primitive data type, make it a return value for a function, or wrap it inside an object. When you pass a Java object, such as an array, a vector, or a string, to a function then you are passing by reference. Yes, a String is actually an object, not a primitive data type. So that means that if you pass an object to a function, you are passing a reference to it, not a duplicate. Any changes you make to the object's member variables will be permanent, which can be either good or bad, depending on whether this was what you intended.

Comparing two objects (== instead of .equals)

When we use the == operator, we are actually comparing two object references, to see if they point to the same object. We cannot compare, for example, two strings for equality, using the == operator. We must instead use the .equals method, which is a method inherited by all classes from java.lang.Object.

Here's the correct way to compare two strings.

```
String abc = "abc"; String def = "def";

// Bad way

if ( (abc + def) == "abcdef" )

{

......

}

// Good way

if ( (abc + def).equals("abcdef") )

{

.....

}
```

4.1 Actual Compiler Error Messages

In this section, we give examples of actual compiler error messages, which enable us to illustrate many points concerning the problems that programmers of all levels may experience. These messages are from the C++ GNU compiler (g++) and were mostly collected from the interaction with the Computer Science students in laboratory sessions of the course "Advanced Programming", at Jaume I University (Castellón, Spain), when it was taught in the first semester of the 2002-2003 academic years. We as freshers of 2011 computer science, department of mathematics, Ahmadu Bello University, Zaria Nigeria, were all tied to Java programming language in 200 level first and second semesters, and are now familiar to the Java compiler error messages and how to debug Java programs. The choice of using the C++ GNU compiler (g++) is to show actual compiler error messages that some of us are not familiar with since we only treated Java and not C++, and how strange this error messages will look.
The messages discussed here correspond to the version of the g++ compiler used in that academic year. Since the error messages for these other versions have not changed significantly, current students in this course are facing basically the same problem. For each message, we use the following structure for our analysis:

1. The error message, as given by the compiler

2. A small piece of the source code, around the (supposedly) offending code. Lines of this code are preceded with the symbol "?" denoting "suspicious" code. The particular line of code that the compiler points to as the location of the error is underlined.

3. The diagnostic, that is, a simple explanation of where the problem in the source code is and why the compiler complains. Notice that more often than not, this diagnostic cannot be derived easily and directly from the error message, but only after a lot of thought or thanks to previous experience. For those readers with some background in C++, it could be an interesting exercise to think about a possible diagnosis after seeing the error message and the code snippet, but before reading further.

4. An alternative error message, which could be more appropriate, because it leads more directly to the true diagnosis of the problem. Please, notice that different people may disagree on how helpful and informative different messages are. Then, the spirit of these alternative messages is to suggest that better ones are possible, without claiming the ones provided are absolutely good.

5. A comment about why the message error is difficult, or confusing or problematic, which principles of human-computer interaction seem to be violated or ignored, and what could be done about it.

The code snippets shown here are deliberately simple, because we want to focus on the error messages and the context giving rise to them. For a clearer presentation, the example error messages are introduced under headings summarizing what the main problem with those messages is, these headings are:

- Unclear, not-to-the-point messages
- Misleading messages
- Same logical error, different error messages
- Internal-detail messages
- Same error messages, many possible logical errors.

Unclear, Not-to-the-Point Messages

Error Message 1:

ANSI C++ forbids declaration ‘ostream’ with no type ‘ostream’ is neither function nor method; cannot be declared friend parse error before ‘&’

Offending Code:

```
?class SavingAccount {

?friend ostream & operator<< (ostream &os, const SavingAccount &sA);

? } ;
```

Diagnostic: The problem is just that the programmer forgot to include the header file iostream.h, thus the compiler does not know what ostream is.

Alternative Message:

I do not know what ‘ostream’ is. Perhaps you forgot to include a header file (maybe ‘ostream.h’)

Comments:

The reader may think that the original error message is not that difficult. And it is not, but only once you have found it and solved it several times. Another issue is the last part of the message: parse error before ‘&’. But, which ‘&’ does this refer to? There are three ‘&’ symbols in the same line of code and the programmer might wrongly focus on one of them without realizing there are others. Regarding the suggested alternative, it is neither difficult nor unreasonable for the compiler to suggest what the missing header is, given the operator (<<) that is being declared, and the fact that the well-known ostream keyword is present in the source code. Notice that the alternative message is anthropomorphic in that the compiler seems to be alive and is able to speak and think (“I do not know...”).

Misleading Messages

Error Message 2:

cannot initialize friend function ‘<<’ friend declaration not in class definition

Offending Code:

```
? friend ostream & operator<<

(ostream &os,const SavingAccount & sA) {

? os << ‘‘this is a saving account’’;

? return os;

? }
```

Diagnostic:

The keyword friend is used inside the declaration of a class giving the friendship to a particular function, not when defining that function. It is easy to make this mistake because one usually takes the header of a function from its declaration, possibly by copy and paste. The error message is easier to understand once one understands not only the rule stating “friendship is given, not taken”, but also who gives the friendship to whom.

Alternative Message:

The keyword friend should not be here (just remove it) remember: friendship is granted, not taken

Comments: The first part of the original message is certainly misleading: what is the compiler trying to “initialize”? The second part, though, has at least something to do with the true diagnostic. But would programmers notice or read this second part or would they just strive to understand the first sentence first?

The proposed alternative message states very directly that the keyword friend should not be where it appears, and it clearly indicates the required action to fix the error (remove this keyword). The second line provides a short, gentle reminder of why friend is not correct here. If a programmer needed more explanations, the message would be extended by additional help levels. For instance, the next level could briefly explain the implication that declaring a function friend within the definition of a class has and why it does not make sense to do it outside the class.

Same Logical Error, Different Error Messages.

Error Message 3:

In method ‘float SavingAccount::getInterestRate()’: parse error before ‘ { ’

Offending Code:

```
? float SavingAccount::getInterestRate()
? {

? return rate;

?

? SavingAccount::SavingAccount() {}
```

Diagnostic:

In this and the following two examples, the cause of the error is the same: a missing ‘ } ’ for closing the body of a preceding function (here, getInterestRate()). However, the error messages differ depending on the particular place where this character is missing.

Alternative Message:

A function declaration inside a function body is not possible. Did you forget ‘ } ’ to close the body of the previous function definition?

Comments:

Because these sample code fragments are simple, the source of the problem can be quite easily identified. However, this is not necessarily always the case, and when code in the body of a function is longer, and it has several pairs of braces, ‘’-‘’, and indenting is not done properly, it is not rare to miss some ‘ } ’ that matches a previous ‘ { ’, without realizing it is missing.

In such circumstances, the current error message may not be very helpful. Here the problem lies in the lack of precision locating the source of error. Programmers tend to look for potential mistakes locally, probably in the same line or the line just before the offending line. Fortunately, in this error message there is a good clue to help us find out where the problem is: it seems that the error is found while parsing float SavingAccount::getInterestRate(), which is suspicious considering where the offending line is.

Error Message 4:

In method ‘float SavingAccount::getInterestRate()’: parse error before ‘,’

Offending Code:

```
?float SavingAccount::getInterestRate()

{

?return rate;

?
```

?SavingAccount::SavingAccount(string

owner , float initialBalance)

? : owner(owner),

balance(initialBalance)

? {}

Diagnostic:

The same as in Error 3.

Alternative Message:

A function declaration inside a function body is not possible. Did you forget ‘ } ’ to close the body of the previous function definition?

Comments:

The puzzling point here is that one could try to see what is wrong just before the comma (,). Typically this message is issued when a type name has been misspelled, or a necessary header has not been included. For instance, in this line of code, one may wonder whether the compiler recognizes string, whereas the actual problem is logically far from this (again, a missing ‘ } ’ in the body of the previous function).

Error Message 5:

In method ‘SavingAccount::SavingAccount ()’: declaration of ‘float SavingAccount::getInterestRate()’ outside of class is not definition

Offending Code:

? SavingAccount::SavingAccount() {

?

? float SavingAccount::getInterestRate()

{

? return rate;

? }

Diagnostic:

The same as in Error 3.

Alternative Message:

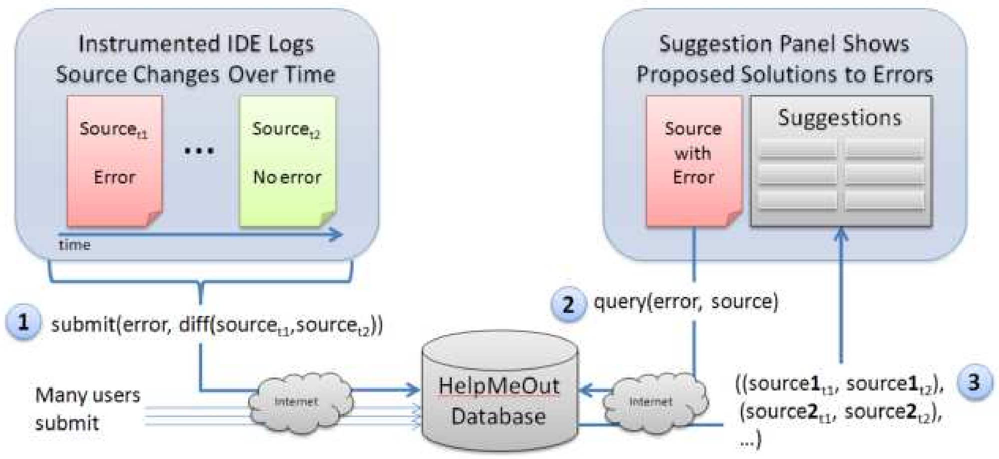
A function declaration inside a function body is not possible. Did you forget ‘ } ’ to close the body of the previous function definition?

Comments: The error message here is quite different to (and longer than) the other two. The sentence combines the concepts of declaration and definition that not every programmer (especially novices ones) may clearly distinguish, or at least Not without an extra, conscious, mental effort. Good human-computer interfaces are designed to minimize the required mental effort. In particular, programmers have enough work with their demanding primary task (programming itself), without also having to deal with an annoying secondary task (understanding compiler error messages and fixing the underlying problems A missing ‘ } ’ in a definition of a member function inside its class (“inline” definition) yields another error message: parse error at end of input with the offending line being a line beyond the last line in the file. The spatial locality principle does not hold here: the source of the problem in the code may be arbitrarily far from the offending line (the end of file!). Of course, spatial locality can be forced (facilitated) by a well-known trick: divide & conquer. We mean that one can, selectively, get rid of pieces of code so that the wrong code is eventually found. But here it is probably faster to use the temporal locality: where has the programmer been editing recently (i.e., since the last time the source file was compiled)? But most novice programmers do not know either of these tricks (spatial and temporal locality).

5.1 Helping Novice Programmers in Understanding Error Messages

Interpreting compiler error messages is challenging for novice programmers. Presenting examples how other programmers have corrected similar errors may help novices understand and correct such errors. This paper introduces HelpMeOut, a social recommender system that aids debugging by suggesting solutions that peers have applied in the past. HelpMeOut comprises IDE instrumentation to collect examples of code changes that fix compiler errors; a central database that stores fix reports from many users; and a suggestion interface that, given an error, queries the database and presents relevant fixes to the programmer.

The web has fundamentally changed how programmers write and debug code. For example, more code gets written by opportunistically modifying found examples. By and large, the authoring tools have not caught up to this change, most still make the assumption that writing and debugging code happens individually, in isolation. As a result, most social exchanges around programming tasks happen through online forums and blogs. We believe that there is significant latent value in integrating social aspects of development and debugging directly into authoring tools. As a step into the direction of social authoring tools, this paper introduces HelpMeOut, a social recommender system that aids novices with the debugging of compiler error messages by suggesting successful solutions to the errors that other programmers have found. It is known that novice programmers have difficulty interpreting compiler errors. We hypothesize that presenting relevant solution examples makes it easier for novices to interpret and correct error messages, as example modification has a lower expertise barrier for end users than creation from scratch.



Instrumented IDE Logs Source Changes Over Time
Source$_{t1}$
Error
...
Source$_{t2}$
No error
time
1 submit(error, diff(source$_{t1}$,source$_{t2}$))
Many users submit
Internet
HelpMeOut Database
Internet
Suggestion Panel Shows Proposed Solutions to Errors
Source with Error
Suggestions
2 query(error, source)
((source1$_{t1}$, source1$_{t2}$), (source2$_{t1}$, source2$_{t2}$), ...)
3

Fig 1: HelpMeOut suggests corrections to compile time errors based on IDE instrumentation.

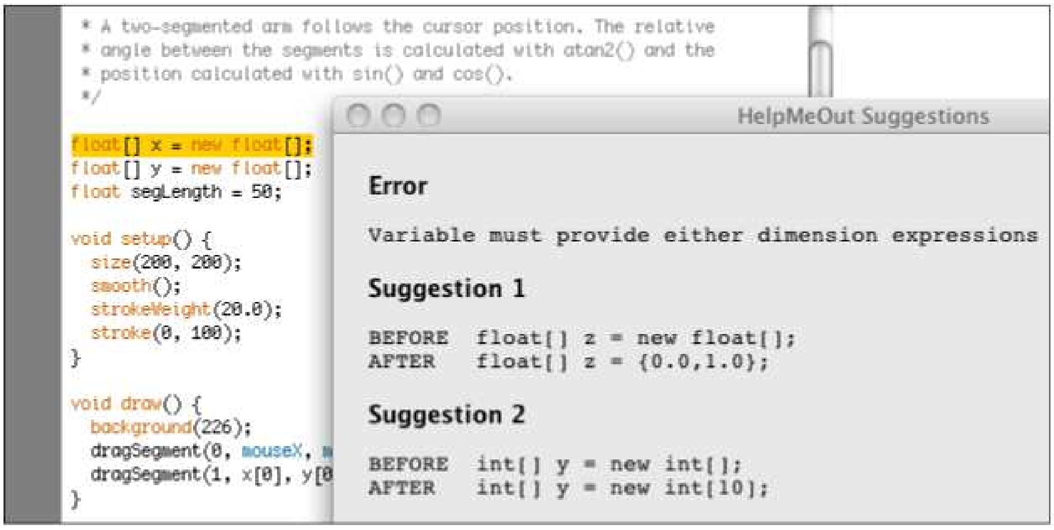
The HelpMeOut system realizes example-centric debugging by augmenting an existing development environment (IDE). HelpMeOut comprises three components: An observation component that tracks code evolution over time, collecting modifications that take source code from an error state to an error-free state (a "fix"). An online database which stores fixes and which can be queried for most relevant examples, given a compile time error and code context. A suggestion interface inside an IDE that, given a compiler error, queries the online database and then presents a list of possible fixes to the user.

SCENARIO

The following short scenario demonstrates how HelpMeOut can aid users: Jim works on a visual animation that displays a graphical sprite following his mouse. In his code, he incorrectly initializes a variable array:

```
float[] x = new float[];
```

When trying to compile his code he receives the error message "Variable must provide either dimension expressions or an array initializer." Not sure what either of the two options are, he consults the HelpMeOut suggestion panel. Looking over the suggestions, he sees that he can either add a size to the right-hand side of his variable intialization, or provide explicit values. He copies the second suggested fix, and pastes the line into his code.



```
* A two-segmented arm follows the cursor position. The relative
* angle between the segments is calculated with atan2() and the
* position calculated with sin() and cos().
*/

float[] x = new float[];
float[] y = new float[];
float segLength = 50;

void setup() {
  size(200, 200);
  smooth();
  strokeWeight(20.0);
  stroke(0, 100);
}

void draw() {
  background(226);
  dragSegment(0, mouseX, m
  dragSegment(1, x[0], y[0
}
```

HelpMeOut Suggestions

Error

```
Variable must provide either dimension expressions
```

Suggestion 1

```
BEFORE   float[] z = new float[];
AFTER    float[] z = {0.0,1.0};
```

Suggestion 2

```
BEFORE   int[] y = new int[];
AFTER    int[] y = new int[10];
```

Fig 2: The HelpMeOut suggestion panel shows possible corrections for a reported compiler error.

Online forums can also help novice programmers in resolving problems with code fragments, novice programmers post error messages and expert programmers help in resolving their issues with compiler error messages. Examples of online forums for programmers are: www.stackoverflow.com, www.programmingforums.org, www.codingforums.com, etc

6.1 Principles for Compiler Error Message Design

Why do people commit errors? Three reasons are given by Shneiderman: lack of knowledge, incorrect understanding, or inadvertent slips. It can be readily observed that all of these can be true for programmers interacting with a machine. Well-designed compiler error messages should provide help, not obstacles. In contrast, poorly-designed messages affect both novices and experts.

The following is the set of proposed principles and next to each are some heuristics they are related to the following

- Clarity and brevity (aesthetic and minimalist design, recognition rather than recall).
- Specificity (recognition rather than recall; help user recognize, diagnose and recover from errors).
- Context-insensitivity (consistency and standards).
- Locality (flexibility and efficiency of use).
- Proper phrasing (match between system and the real world).
- Consistency (consistency and standards).
- Suitable visual design (aesthetic and minimalist design; error prevention)
- Extensible help (help and documentation).

7.1 Conclusion

Programming is an academic discipline. Furthermore, programming is a skill requiring novice programmers to utilize multiple types of learning simultaneously.

Novice programmers lack the knowledge and skills of programming experts. The knowledge of novices tends to be context specific rather than general. Novices spend little time in planning and testing code, and try to correct their programs with small local fixes. Some approaches exist that address the problem to some extent or from some point of view but it’s from other concerned individuals and not from those involved in the compiler design and construction.

Compiler error messages should be redesigned in a way that is easy to comprehend by anyone who wants to vent into programming, by such redesign will novice programmers find it easy to understand and debug their programs.

.

References

http://www.hindawi.com/journals/ahci/2010/602570/

http://www.javacoffeebreak.com/articles/toptenerrors.html

http://elm-lang.org/blog/compiler-errors-for-humans

http://arantxa.ii.uam.es/~modonnel/Compilers/11_ErrorDetection.pdf

http://hci.stanford.edu/publications/2010/helpmeout/hartmann-chi10-helpmeout.pdf

http://www.wseas.us/e-library/conferences/2013/Morioka/EDU/EDU-01.pdf