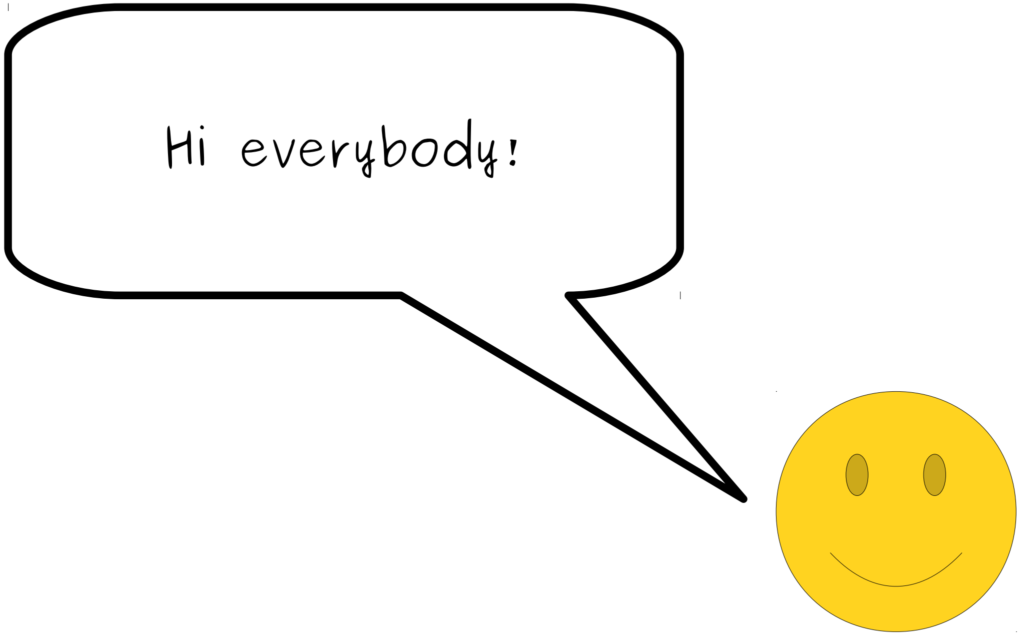
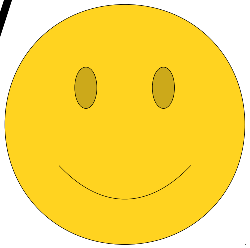
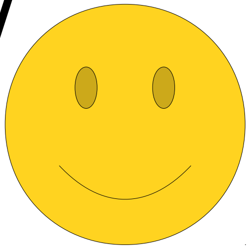
Assignment 0: Using the Debugger


Hi everybody!

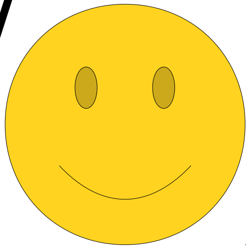
As part of Assignment 0, we'd like you to get a little bit of practice using the debugger in Qt Creator.



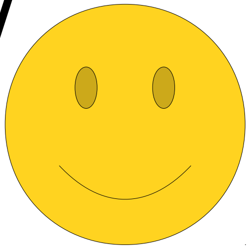
The debugger is a tool you can use to help see what your program is doing as you run it.

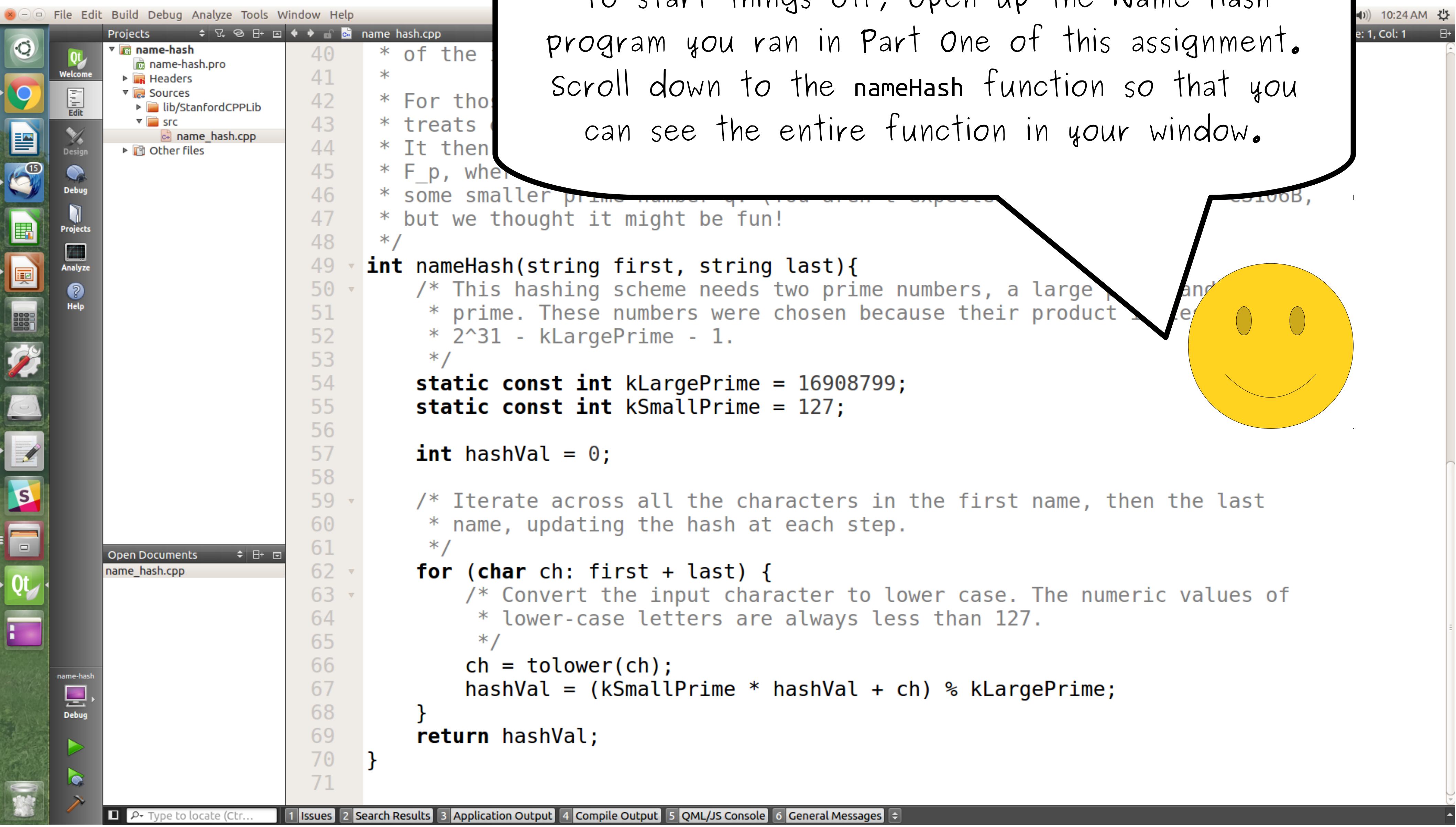


It's really useful for helping find errors in your programs, and the more practice you get with it, the easier it'll be to correct mistakes in the programs you write.



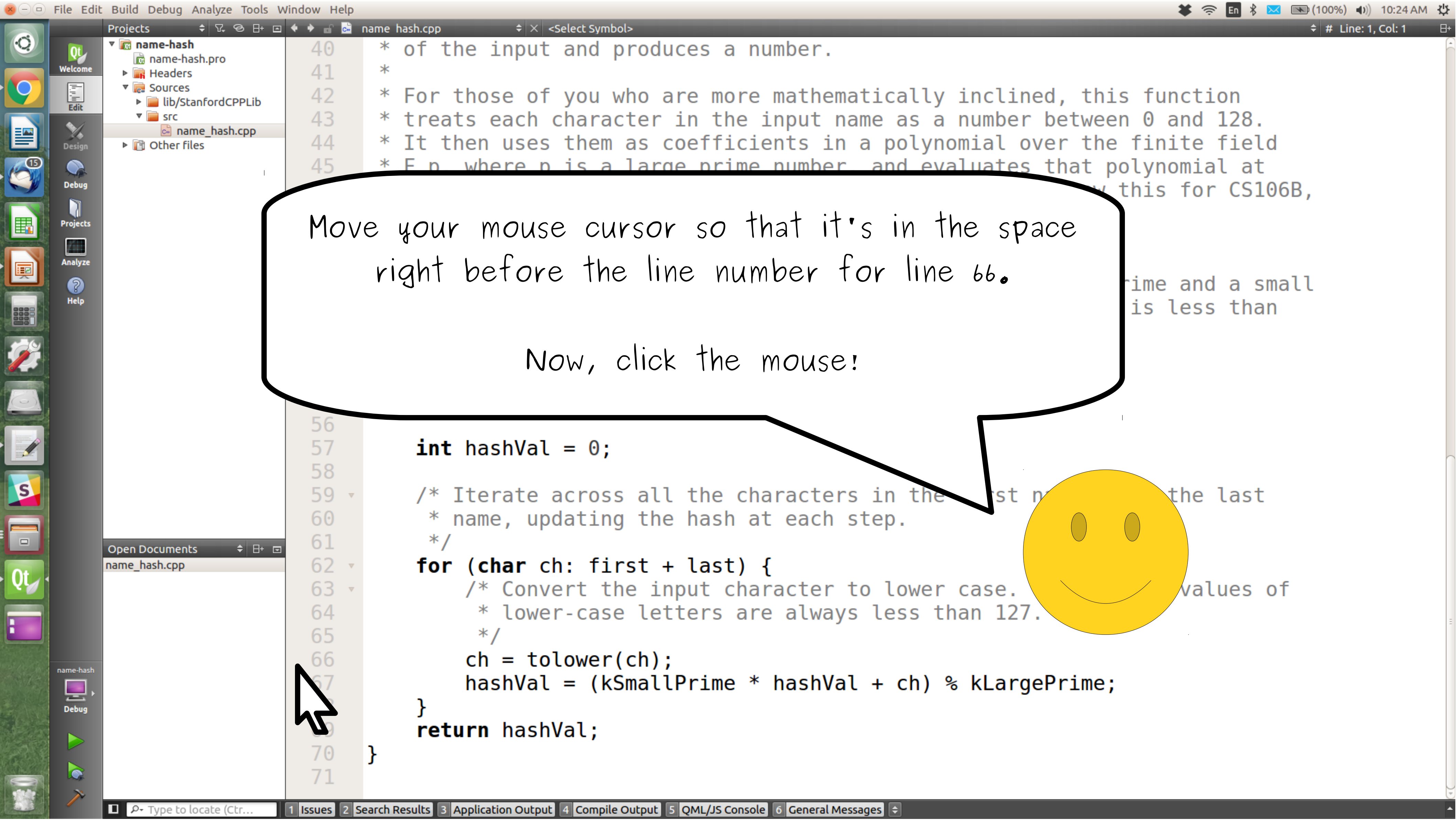
Think of this guide as a little tutorial walkthrough to help give you a sense of how to use the debugger and how to make sense of what you're seeing.





To start things off, open up the Name Hash program you ran in Part One of this assignment. Scroll down to the **nameHash** function so that you can see the entire function in your window.

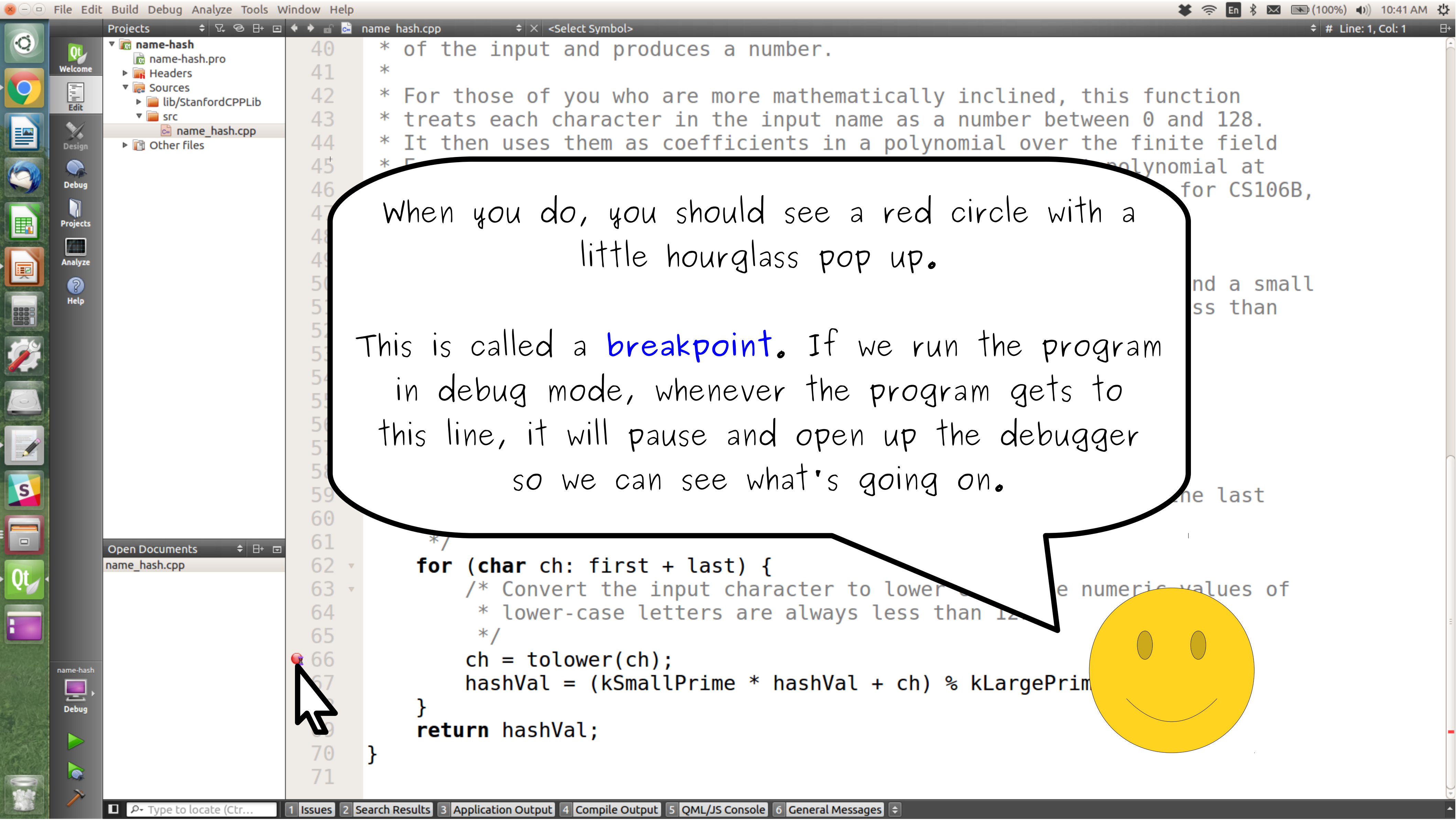
```
 * of the
 *
 * For tho
 * treats
 * It then
 * F_p, whe
 * some smaller
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large
     * prime. These numbers were chosen because their product
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the last
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values of
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```



```
40   * of the input and produces a number.
41   *
42   * For those of you who are more mathematically inclined, this function
43   * treats each character in the input name as a number between 0 and 128.
44   * It then uses them as coefficients in a polynomial over the finite field
45   * F_p, where p is a large prime number, and evaluates that polynomial at
                                                  w this for CS106B,

                                                  ime and a small
                                                  is less than

56
57       int hashVal = 0;
58
59       /* Iterate across all the characters in the                st n          the last
60        * name, updating the hash at each step.
61        */
62       for (char ch: first + last) {
63           /* Convert the input character to lower case.                        values of
64            * lower-case letters are always less than 127.
65            */
66           ch = tolower(ch);
67           hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
         }
         return hashVal;
70   }
71
```

Move your mouse cursor so that it's in the space right before the line number for line 66.

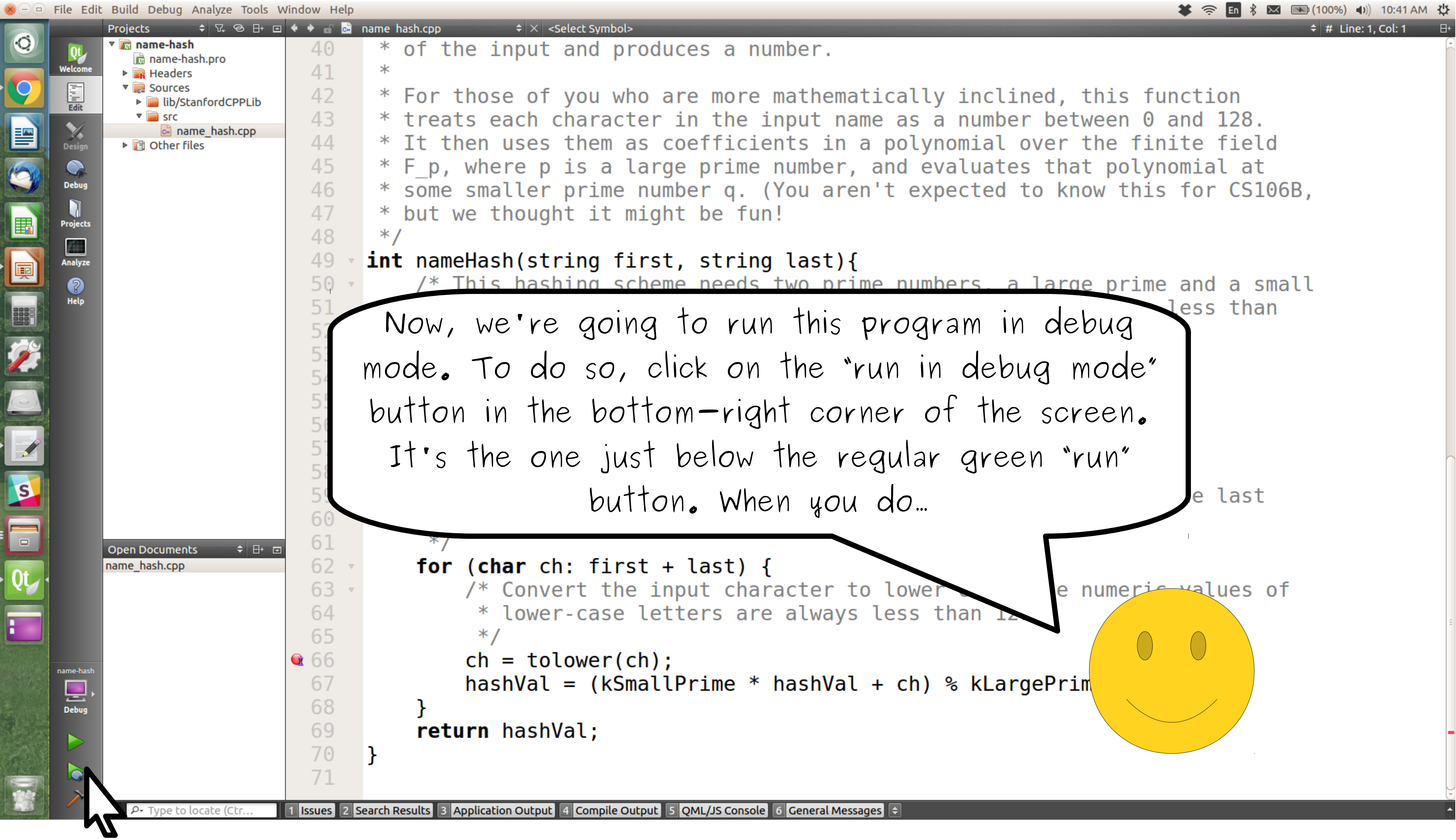
Now, click the mouse!



When you do, you should see a red circle with a little hourglass pop up.

This is called a **breakpoint**. If we run the program in debug mode, whenever the program gets to this line, it will pause and open up the debugger so we can see what's going on.

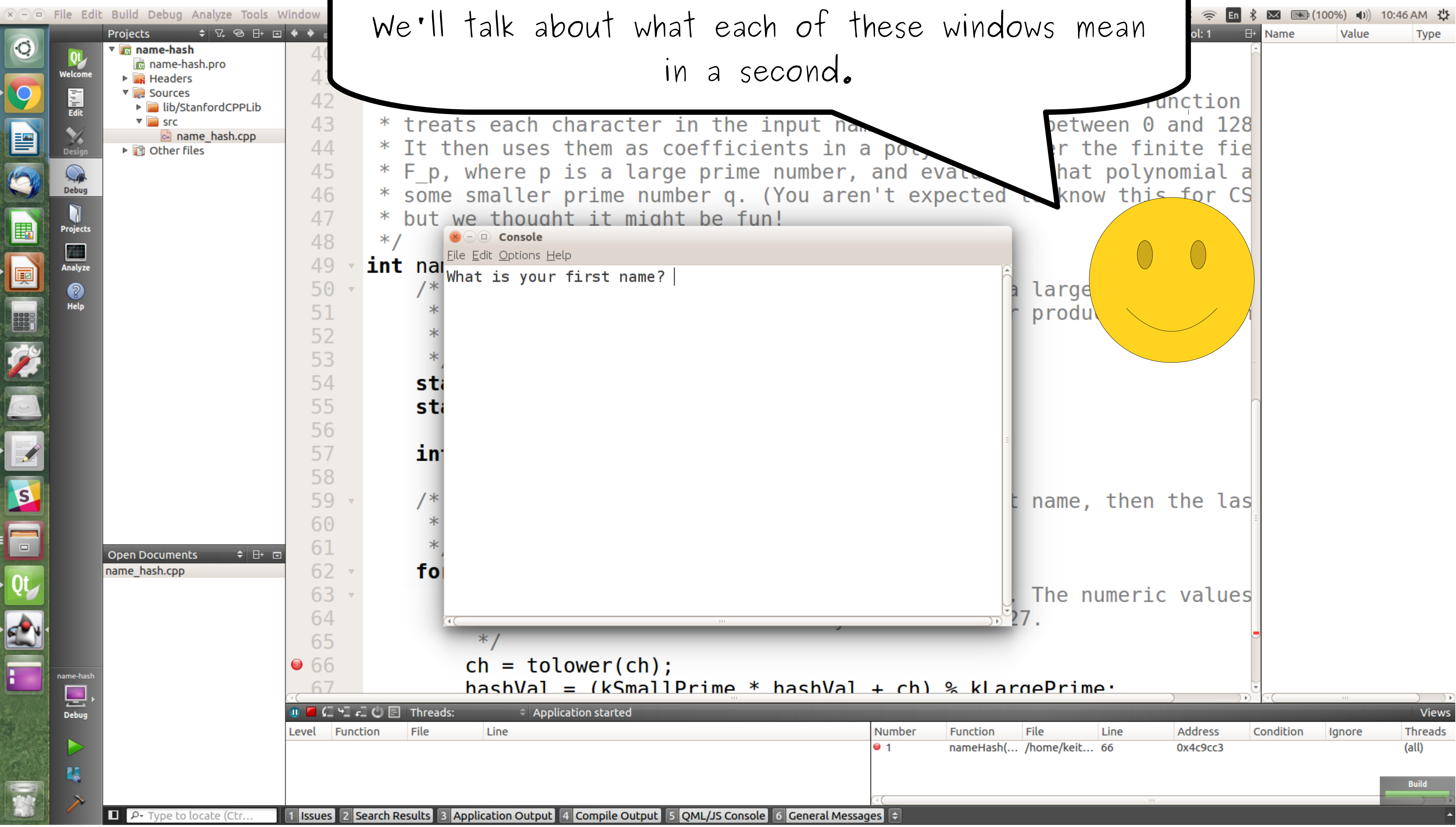
```
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128.
 * It then uses them as coefficients in a polynomial over the finite field
 * F                                                  olynomial at
                                                     for CS106B,

                                                     nd a small
                                                     ss than

                                                     he last
   */
    for (char ch: first + last) {
        /* Convert the input character to lower       e numeric values of
         * lower-case letters are always less than 12
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrim
    }
    return hashVal;
}
```



```
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128.
 * It then uses them as coefficients in a polynomial over the finite field
 * F_p, where p is a large prime number, and evaluates that polynomial at
 * some smaller prime number q. (You aren't expected to know this for CS106B,
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a small
                                                                  less than

                                                                  e last

      */
    for (char ch: first + last) {
        /* Convert the input character to lower              e numeric values of
         * lower-case letters are always less than 12
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrim
    }
    return hashVal;
}
```

Now, we're going to run this program in debug mode. To do so, click on the "run in debug mode" button in the bottom-right corner of the screen. It's the one just below the regular green "run" button. When you do…

… you should see something like this! Notice that a bunch of extra panels popped up in Qt Creator. We'll talk about what each of these windows mean in a second.



```
                                                              unction
 * treats each character in the input nam              between 0 and 128
 * It then uses them as coefficients in a pol          r the finite fie
 * F_p, where p is a large prime number, and eval      hat polynomial a
 * some smaller prime number q. (You aren't expected t  know this for CS
 * but we thought it might be fun!
 */
int nam
    /*                                                   a large
     *                                                   r produ
     *
     */
    sta
    sta

    int

    /*                                                   t name, then the las
     *
     */
    for
                                                         The numeric values
                                                         27.
        */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
```

```
What is your first name?
```

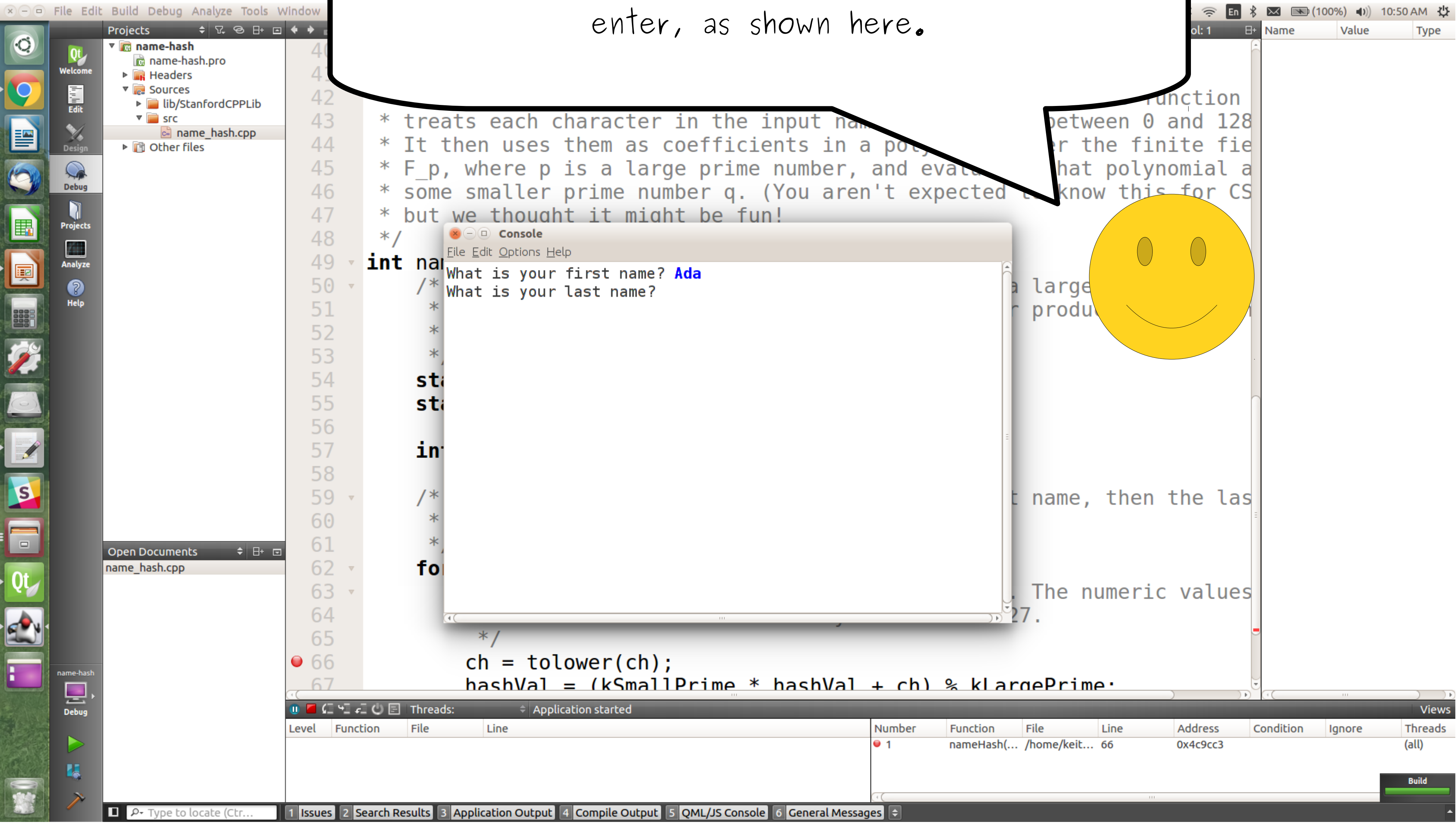
Application started

Level	Function	File	Line

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Name	Value	Type

In the meantime, type in the first name Ada and hit enter, as shown here.



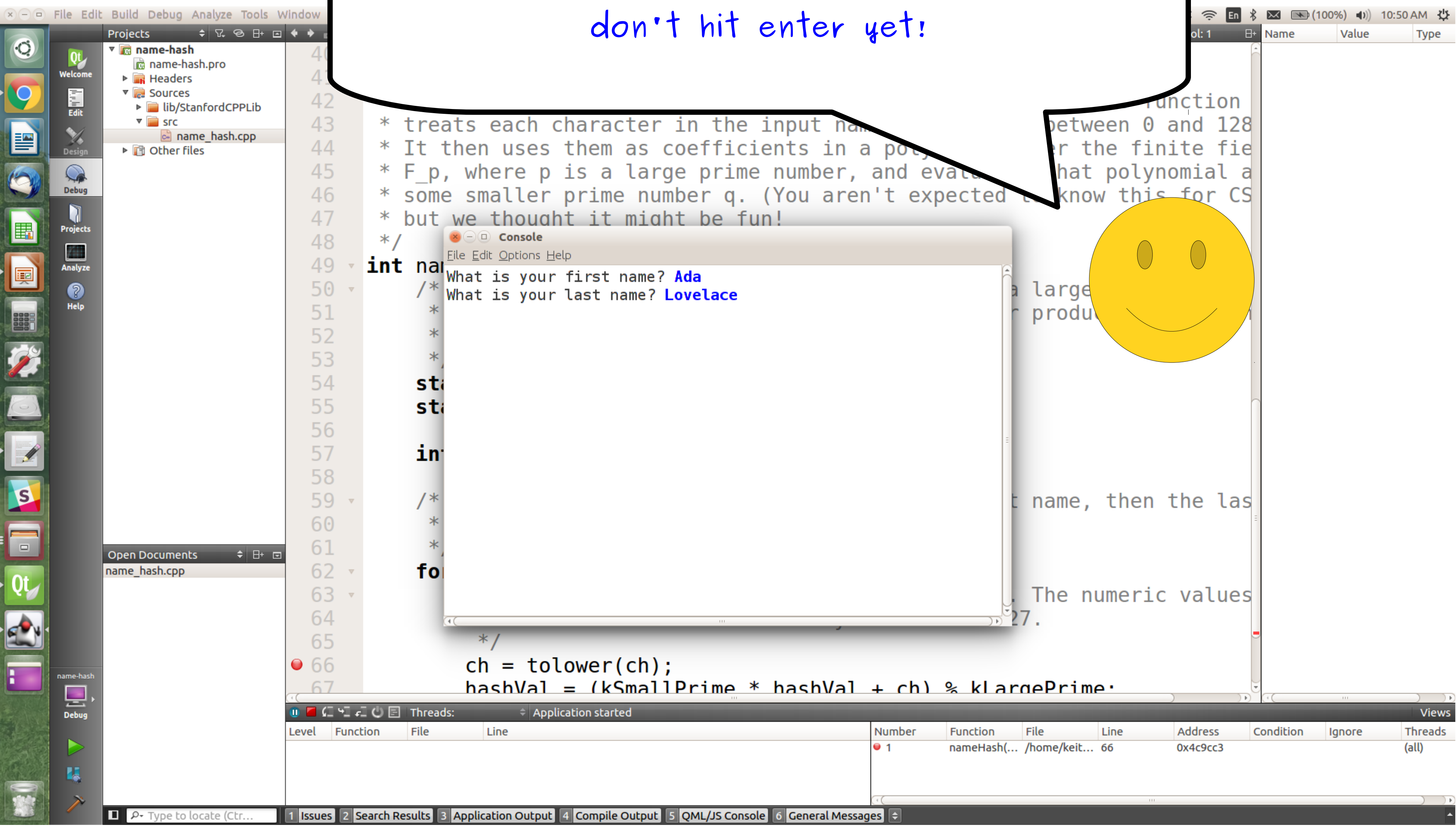
```
What is your first name? Ada
What is your last name?
```

```
                                                              function
 * treats each character in the input na                     between 0 and 128
 * It then uses them as coefficients in a pol                r the finite fie
 * F_p, where p is a large prime number, and eval            hat polynomial a
 * some smaller prime number q. (You aren't expected t       know this for CS
 * but we thought it might be fun!
 */
int na
    /*                                                       a large
     *                                                       produ
     *
     *
    st
    st

    in

    /*                                                       t name, then the las
     *
     *
    fo
                                                             . The numeric values
                                                             27.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
```

Application started

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Now, type in "Lovelace" as a last name, but **don't hit enter yet!**

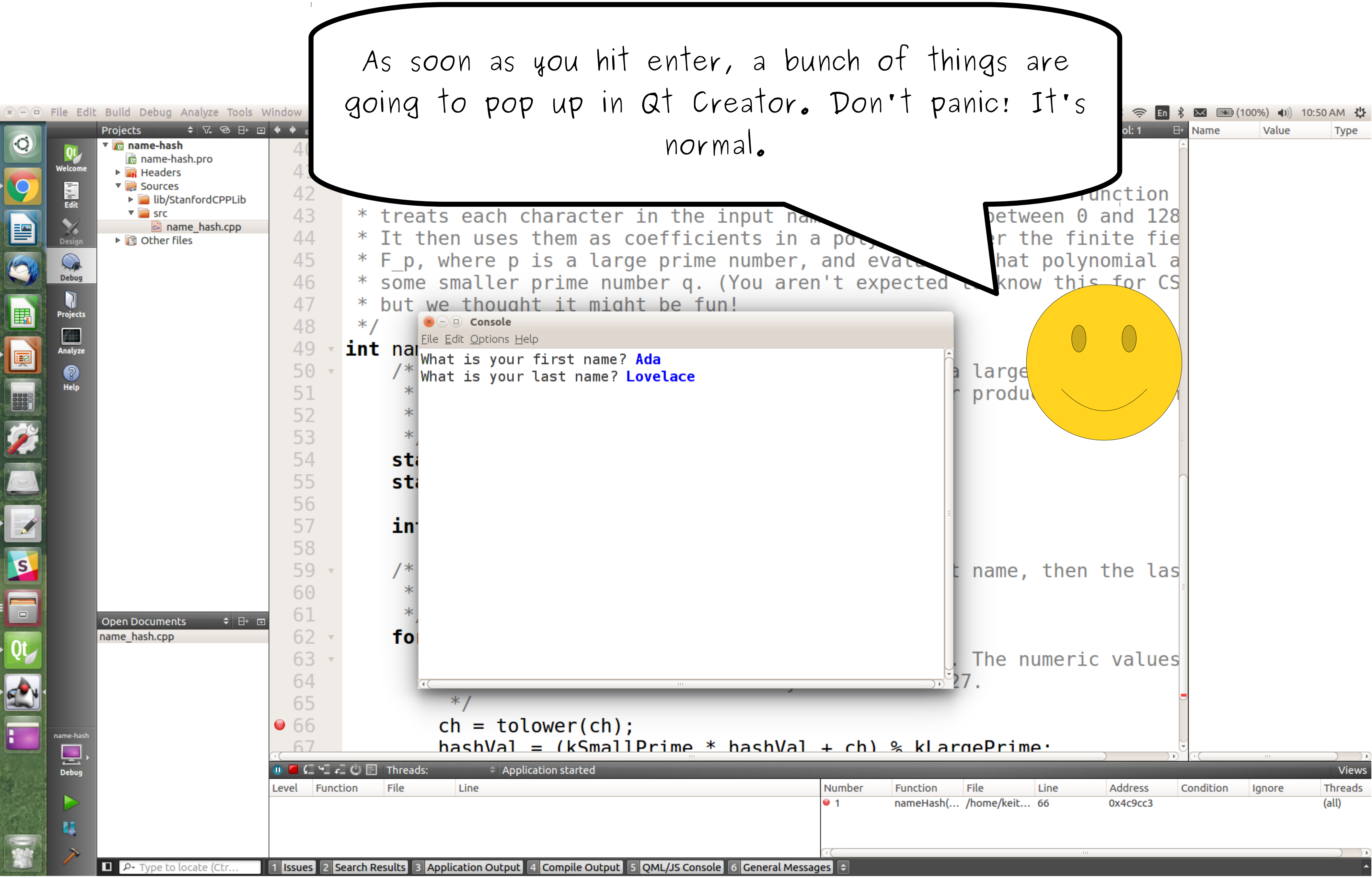


```
                                                          unction
 * treats each character in the input na            between 0 and 128
 * It then uses them as coefficients in a po        r the finite fie
 * F_p, where p is a large prime number, and eva    hat polynomial a
 * some smaller prime number q. (You aren't expected t know this for CS
 * but we thought it might be fun!
 */
int na
    /*                                              a large
     *                                              r produ
     *
     *
    st
    st

    in

    /*                                              t name, then the las
     *
     *
    fo
                                                    . The numeric values
                                                    27.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
```

Console

```
What is your first name? Ada
What is your last name? Lovelace
```

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



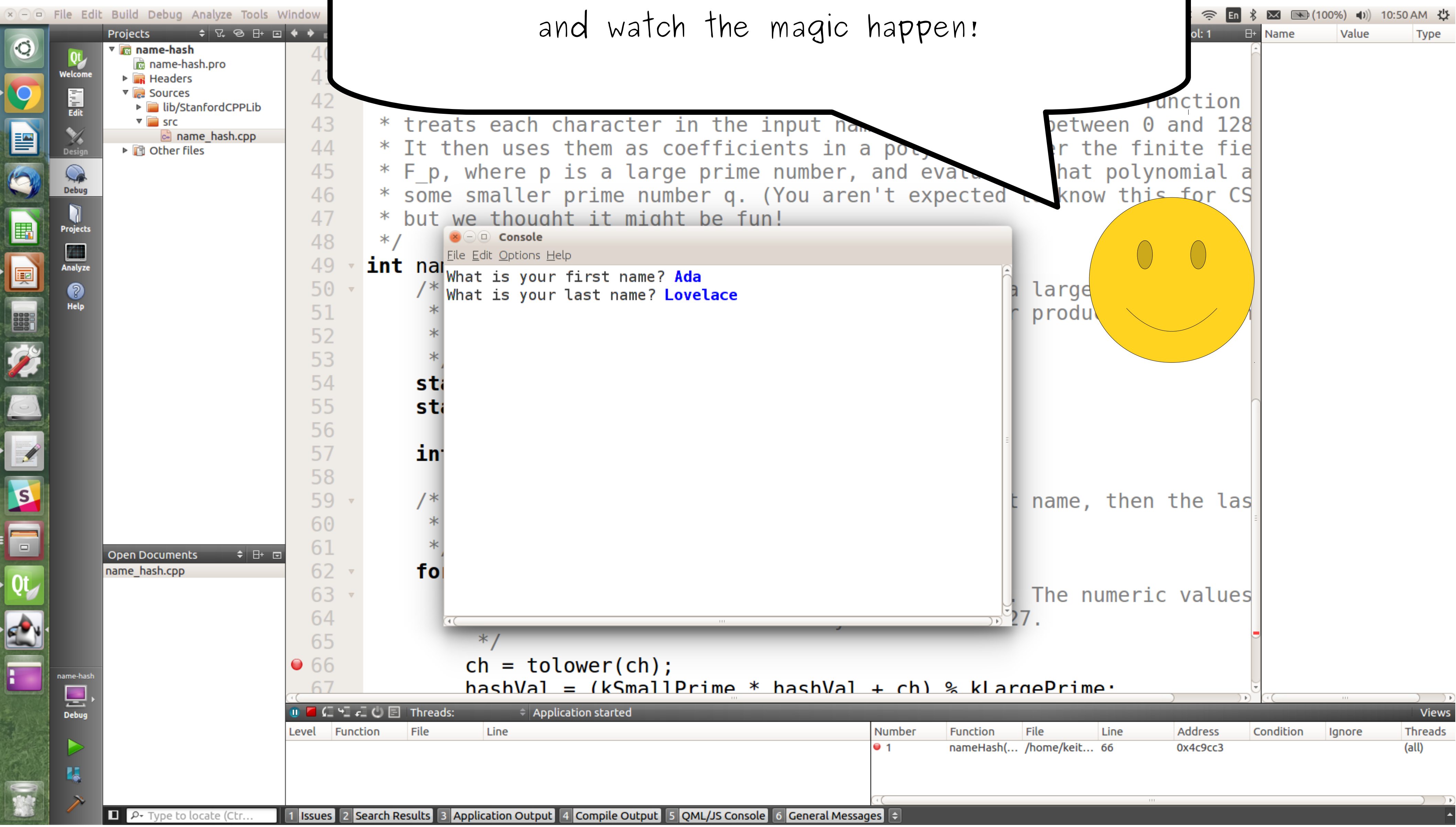
As soon as you hit enter, a bunch of things are going to pop up in Qt Creator. Don't panic! It's normal.

```
What is your first name? Ada
What is your last name? Lovelace
```

```
        ch = tolower(ch);
```

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

With that said, hit enter,
and watch the magic happen!



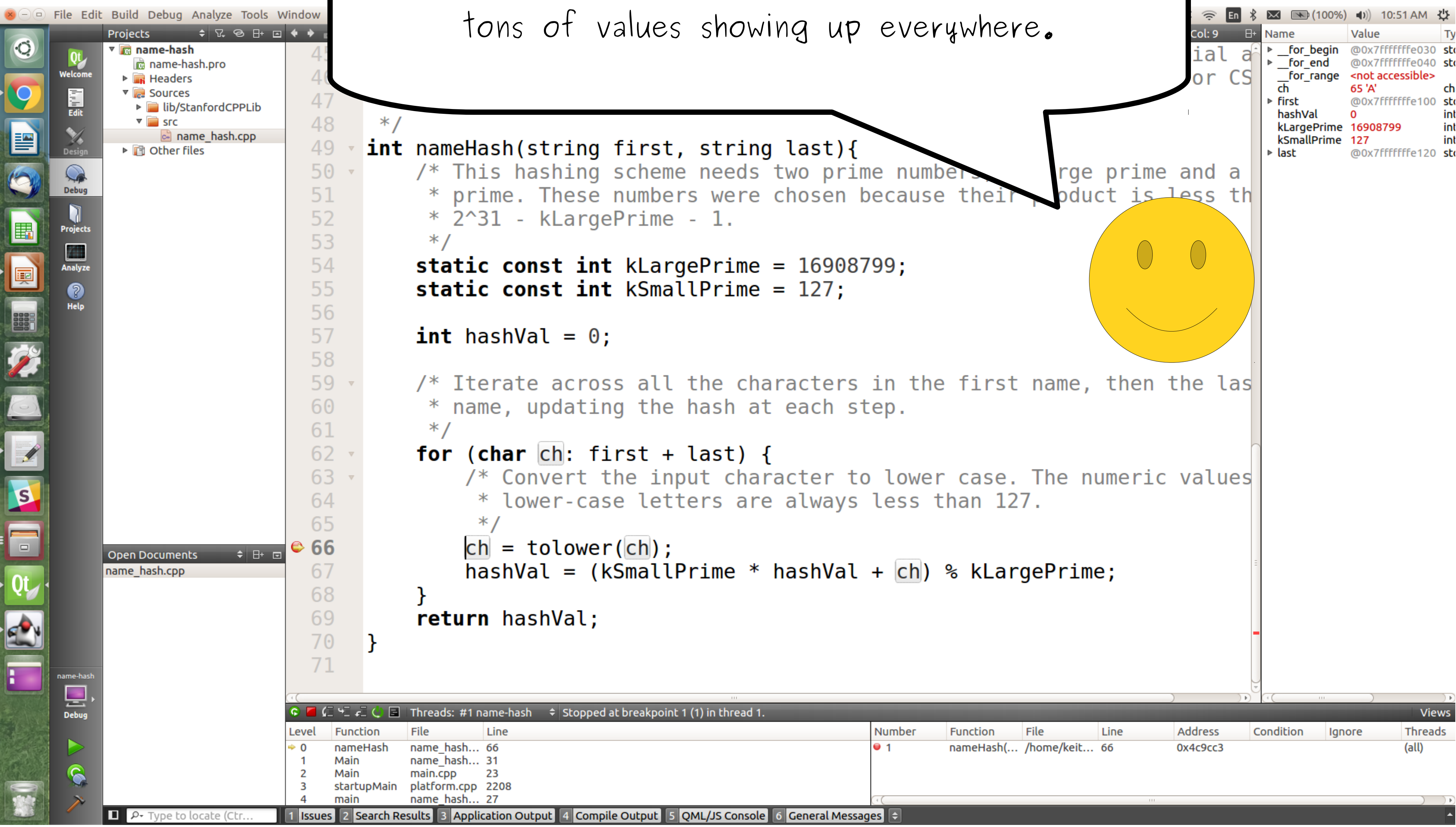
```
 * treats each character in the input na                      between 0 and 128
 * It then uses them as coefficients in a po                  r the finite fie
 * F_p, where p is a large prime number, and eva               hat polynomial a
 * some smaller prime number q. (You aren't expected           know this for CS
 * but we thought it might be fun!
 */
int na
```

```
What is your first name? Ada
What is your last name? Lovelace
```

```
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
```

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Shazam! We're back in Qt Creator, and there's tons of values showing up everywhere.



```
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers                rge prime and a
     * prime. These numbers were chosen because their  oduct is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
▸ __for_begin	@0x7fffffffe030	st
▸ __for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
▸ first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
▸ last	@0x7fffffffe120	st

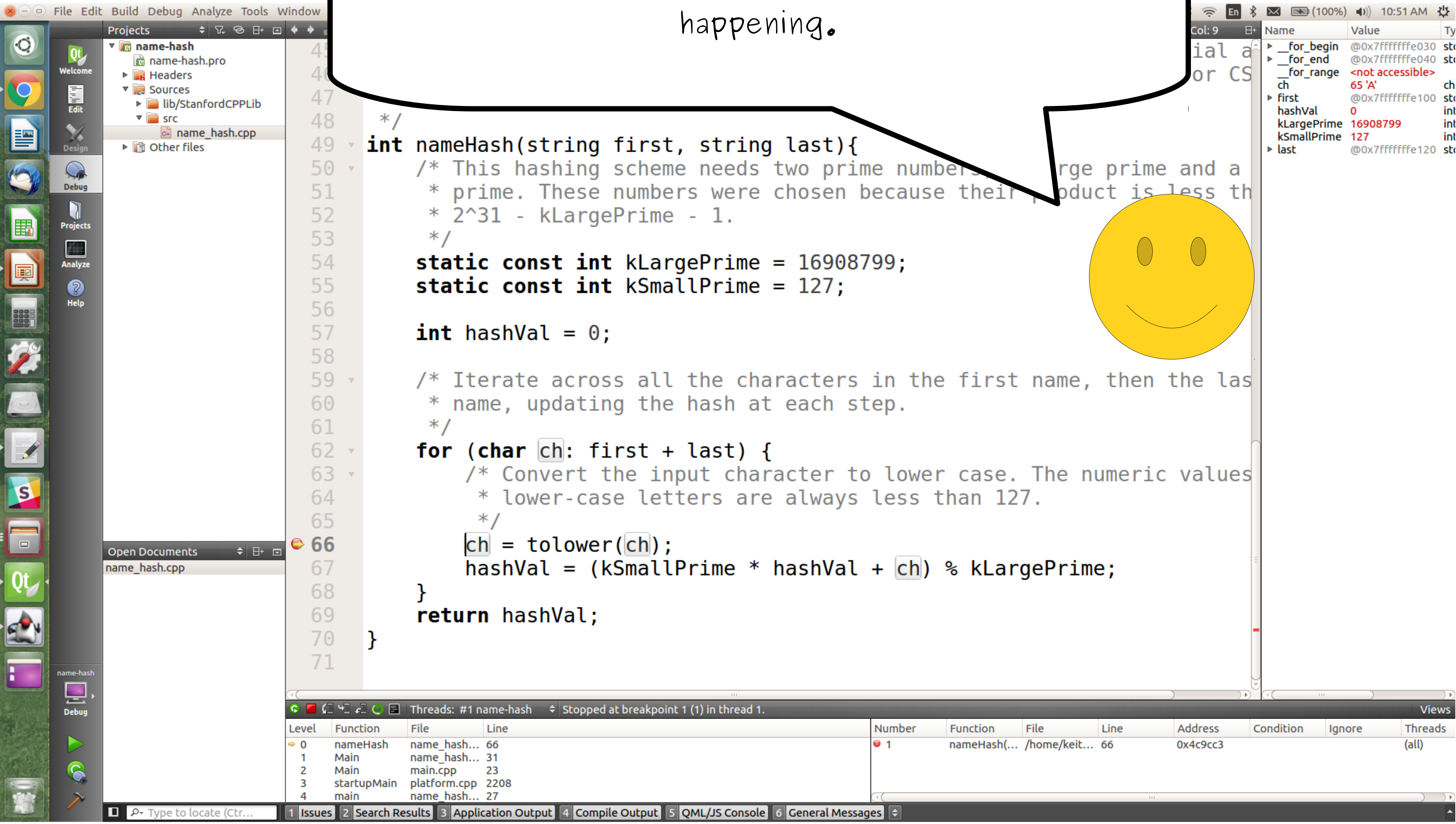
Threads: #1 name-hash

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

There's a lot going on right here. Let's see what's happening.



```
    */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbe          rge prime and a
     * prime. These numbers were chosen because their  oduct is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

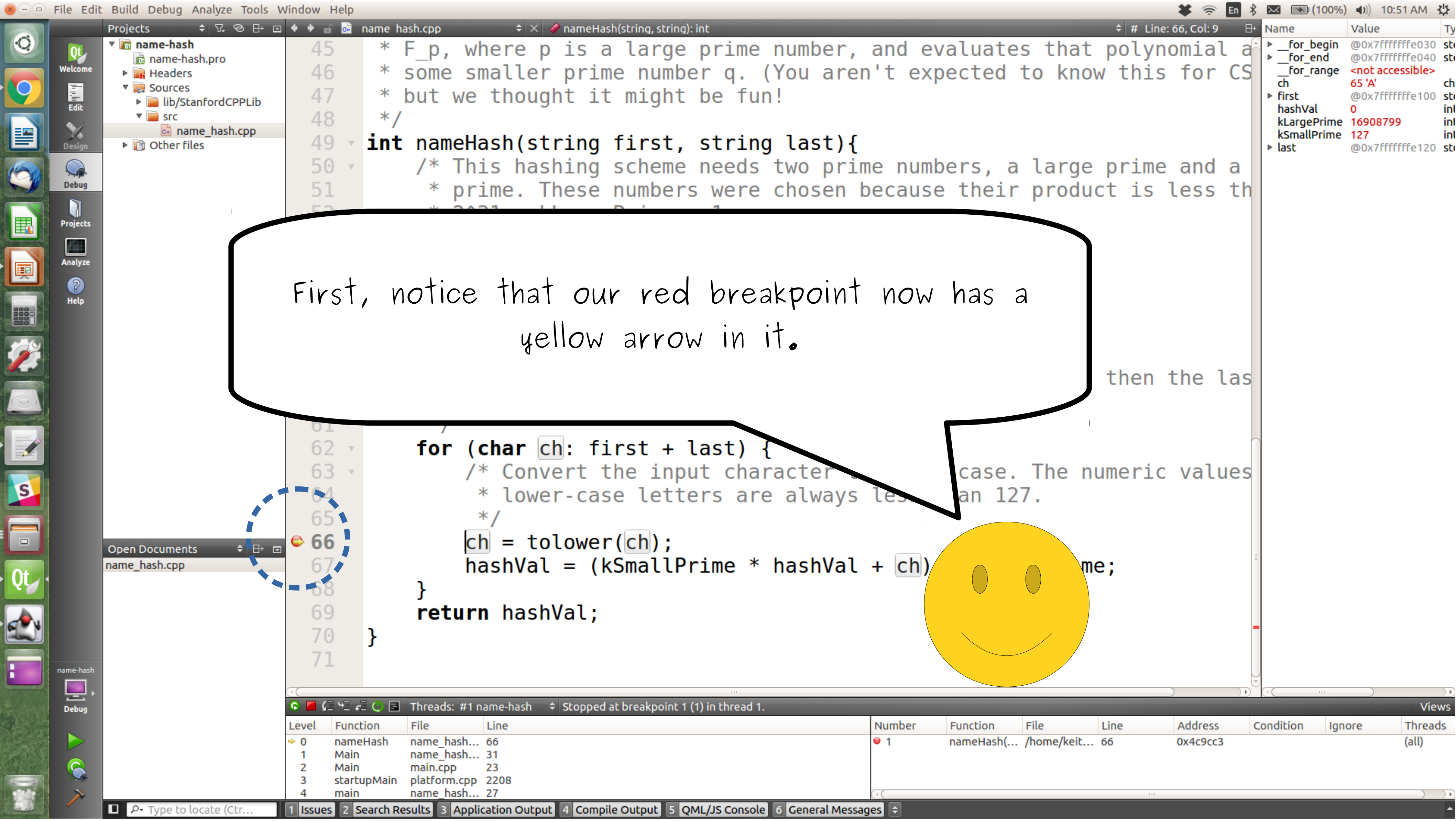
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 QML/JS Console 6 General Messages



First, notice that our red breakpoint now has a yellow arrow in it.

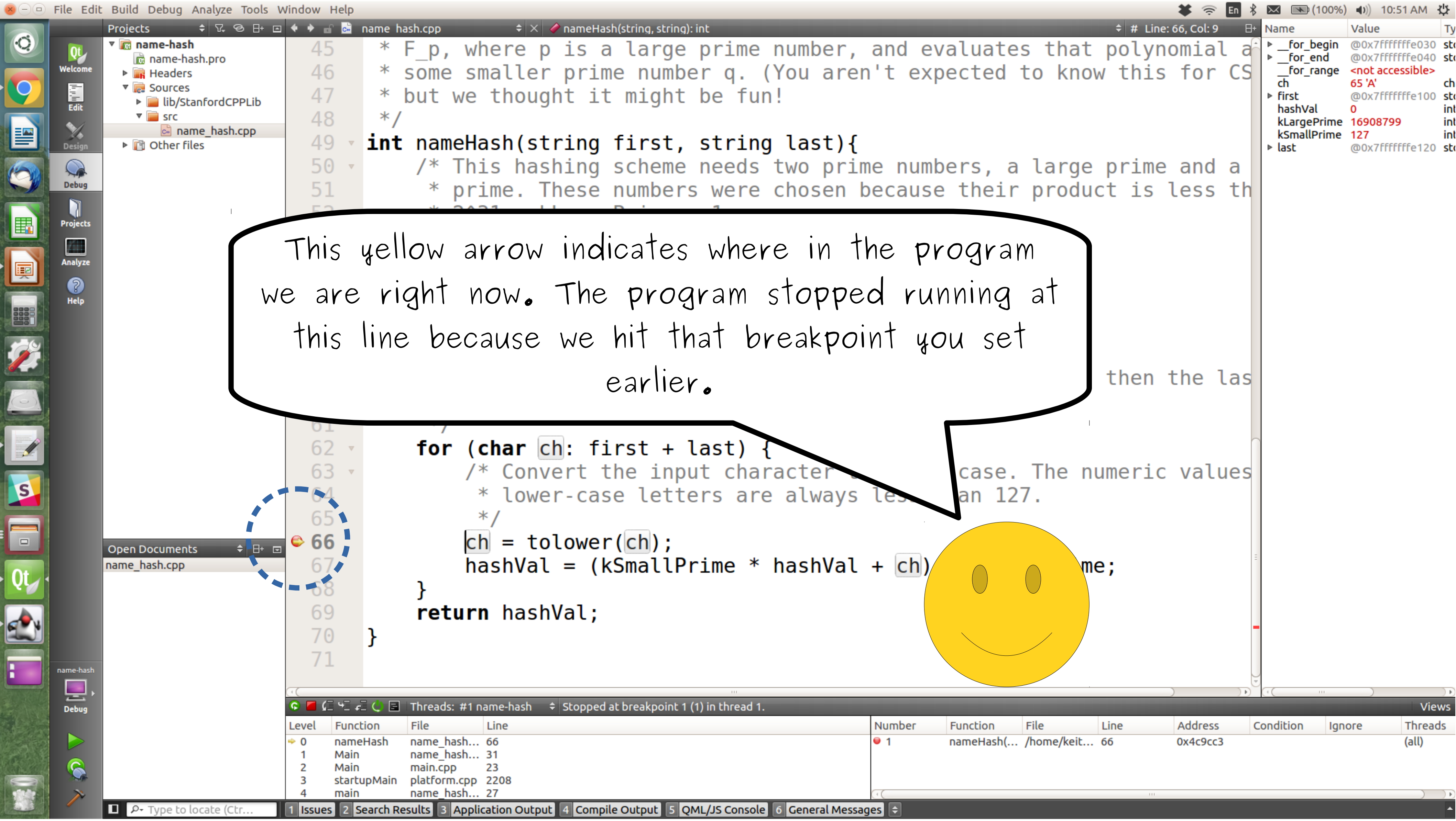
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
                                                            then the las
    for (char ch: first + last) {
        /* Convert the input character                case. The numeric values
         * lower-case letters are always les          an 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch)                me;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
                                                            then the las
    for (char ch: first + last) {
        /* Convert the input character                case. The numeric values
         * lower-case letters are always les          an 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch)              me;
    }
    return hashVal;
}
```

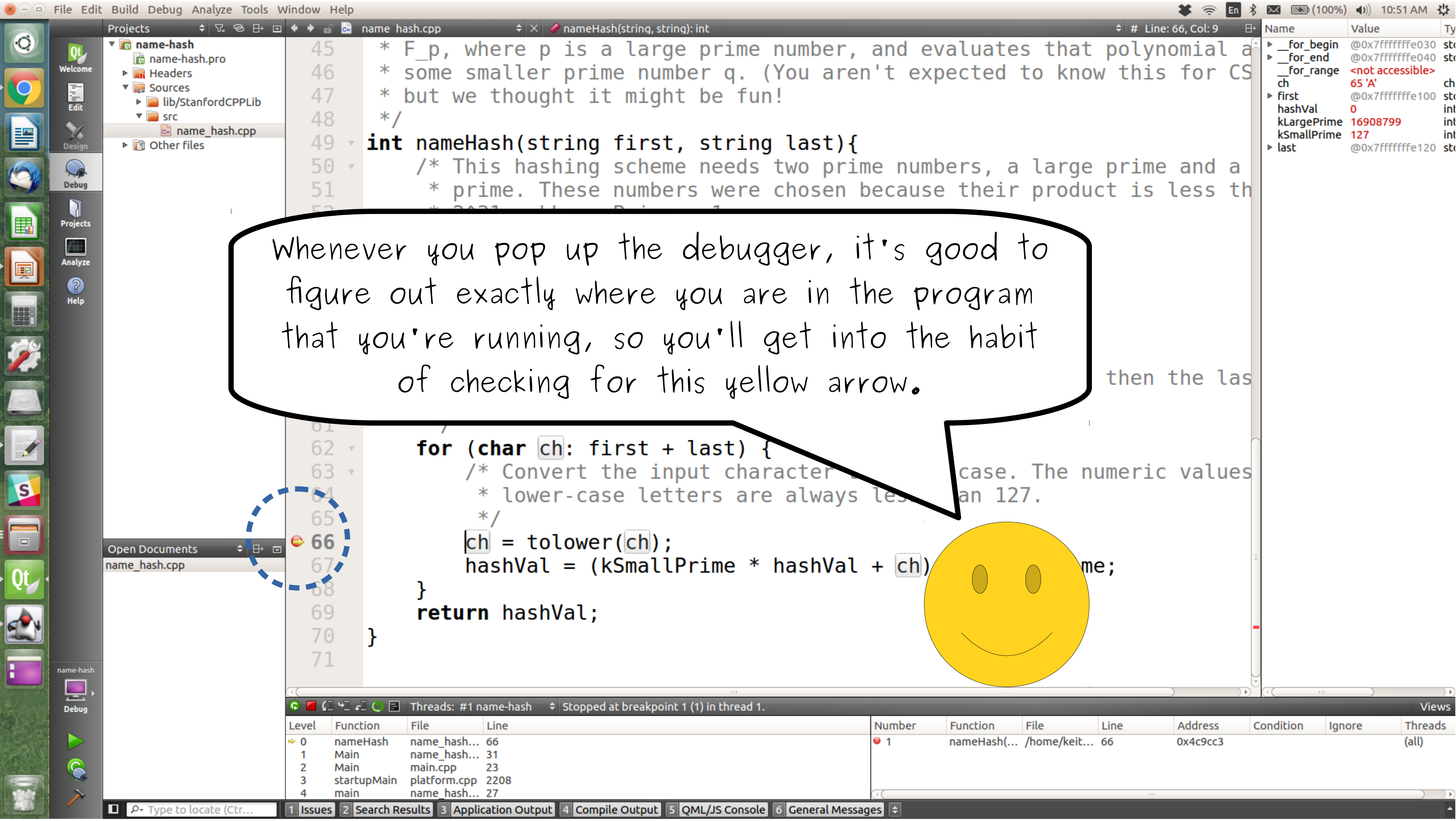
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

This yellow arrow indicates where in the program we are right now. The program stopped running at this line because we hit that breakpoint you set earlier.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th

                                                         then the las

    for (char ch: first + last) {
        /* Convert the input character                case. The numeric values
         * lower-case letters are always les          an 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch)                 me;
    }
    return hashVal;
}
```

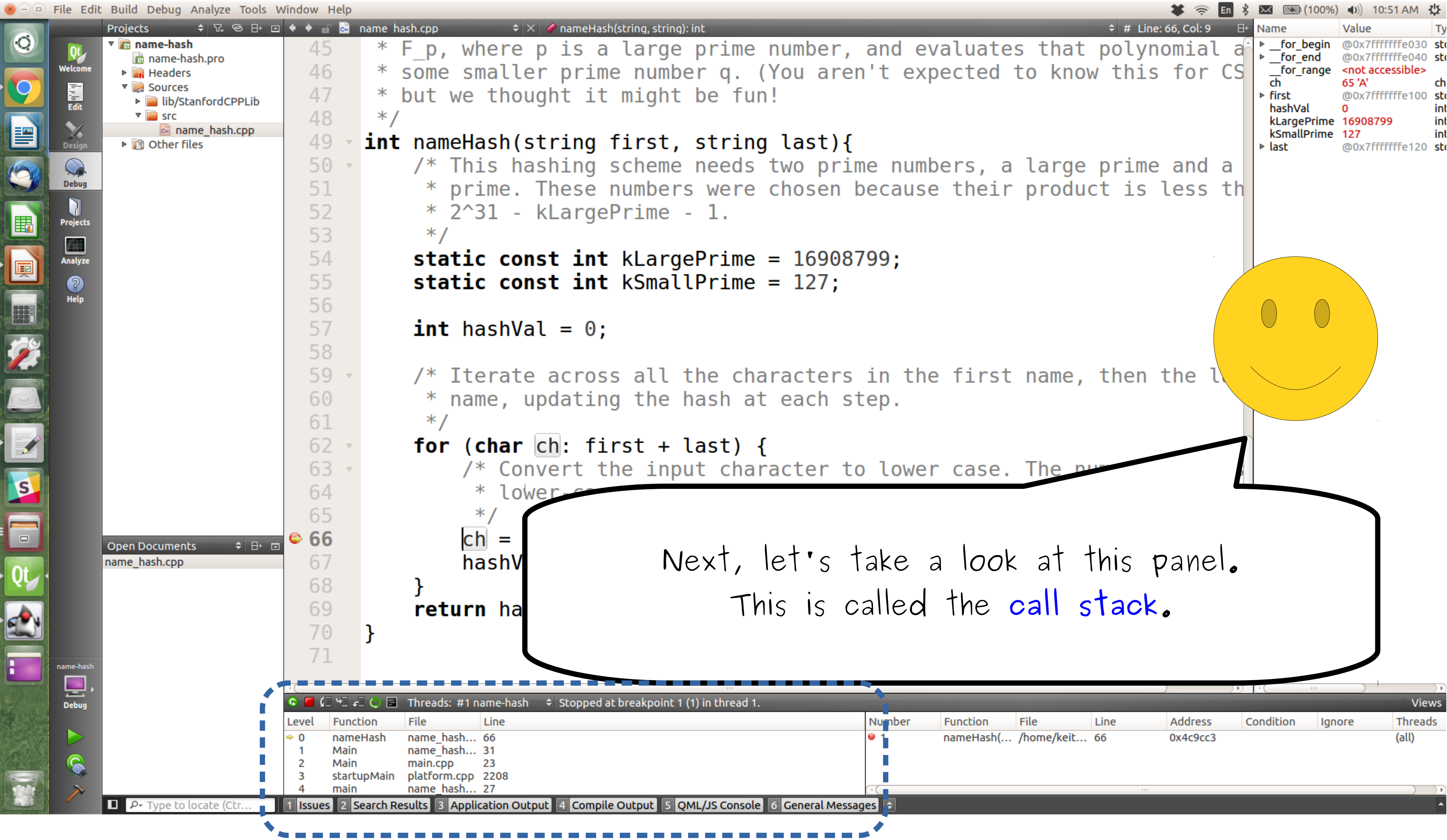
Whenever you pop up the debugger, it's good to figure out exactly where you are in the program that you're running, so you'll get into the habit of checking for this yellow arrow.

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

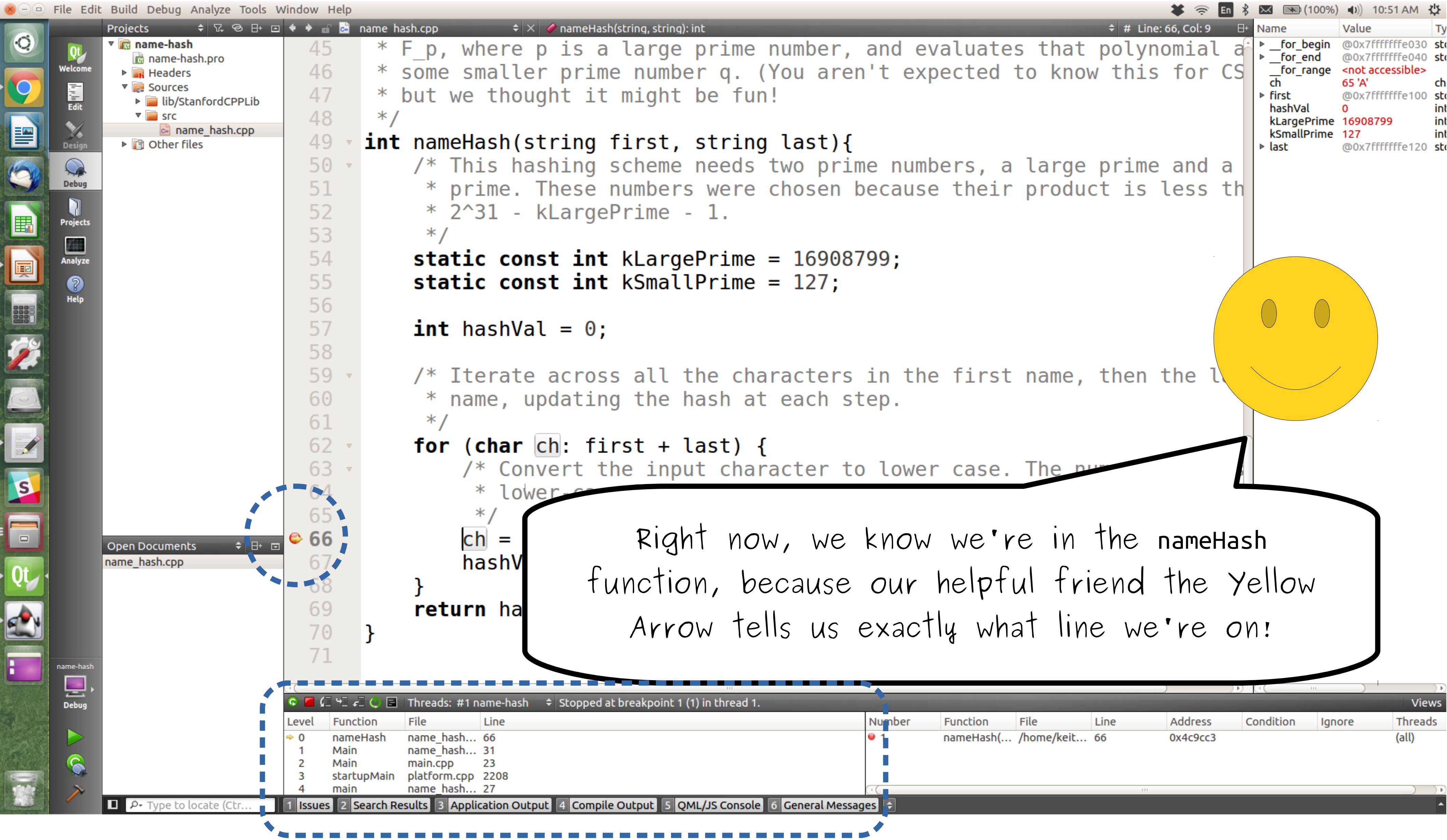
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Next, let's take a look at this panel. This is called the call stack.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

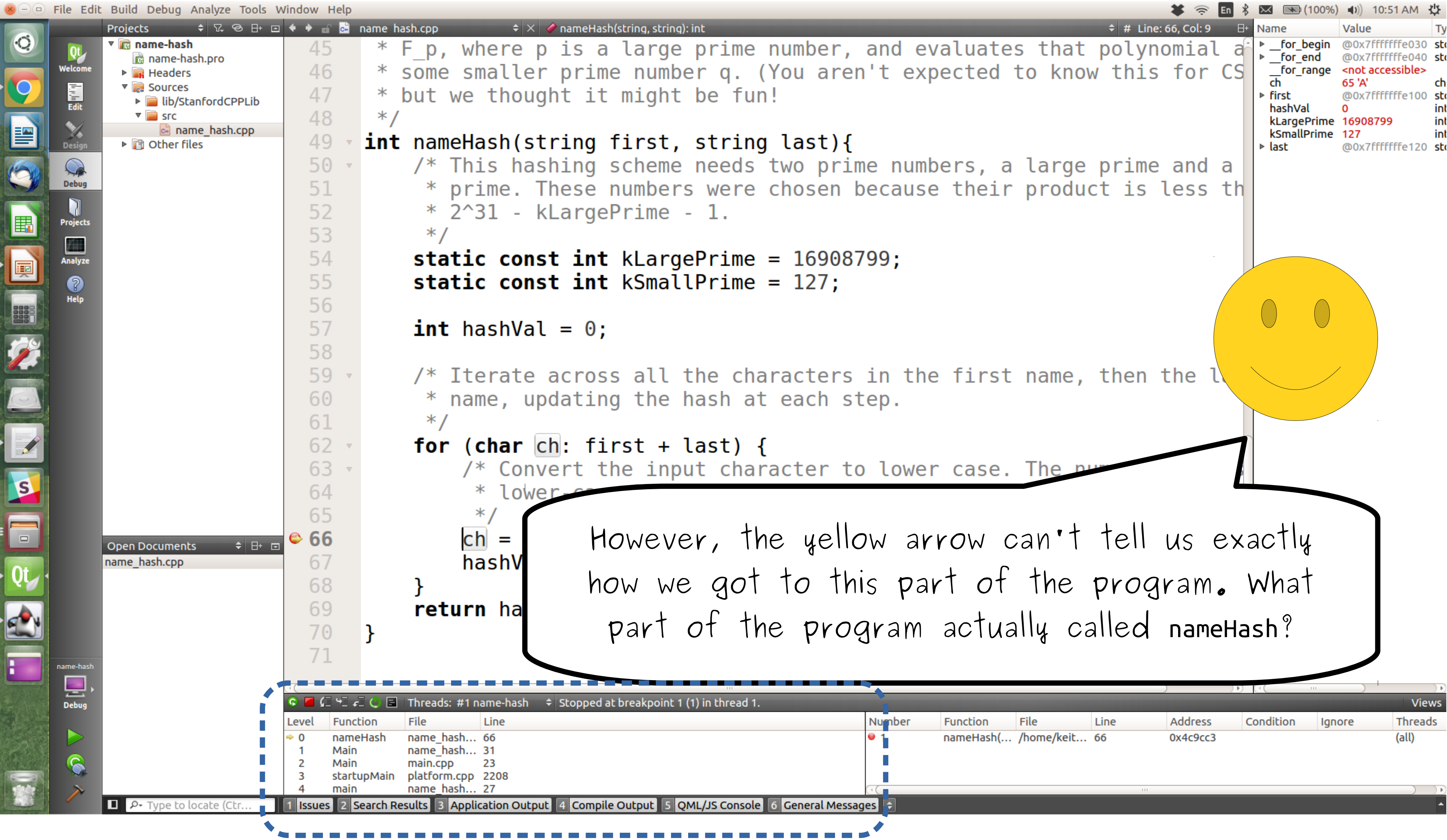
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Right now, we know we're in the nameHash function, because our helpful friend the Yellow Arrow tells us exactly what line we're on!



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

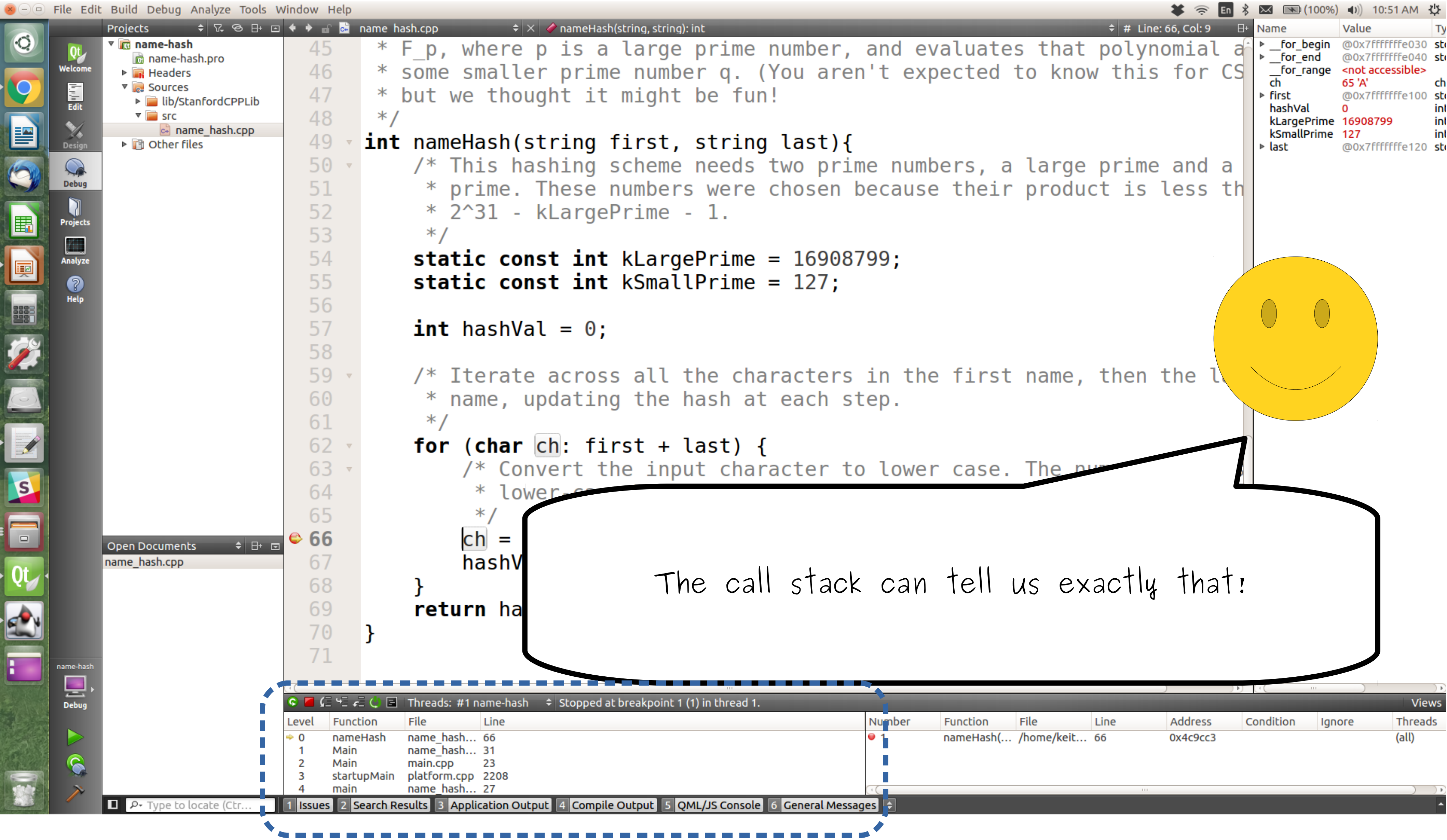
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

However, the yellow arrow can't tell us exactly how we got to this part of the program. What part of the program actually called nameHash?



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

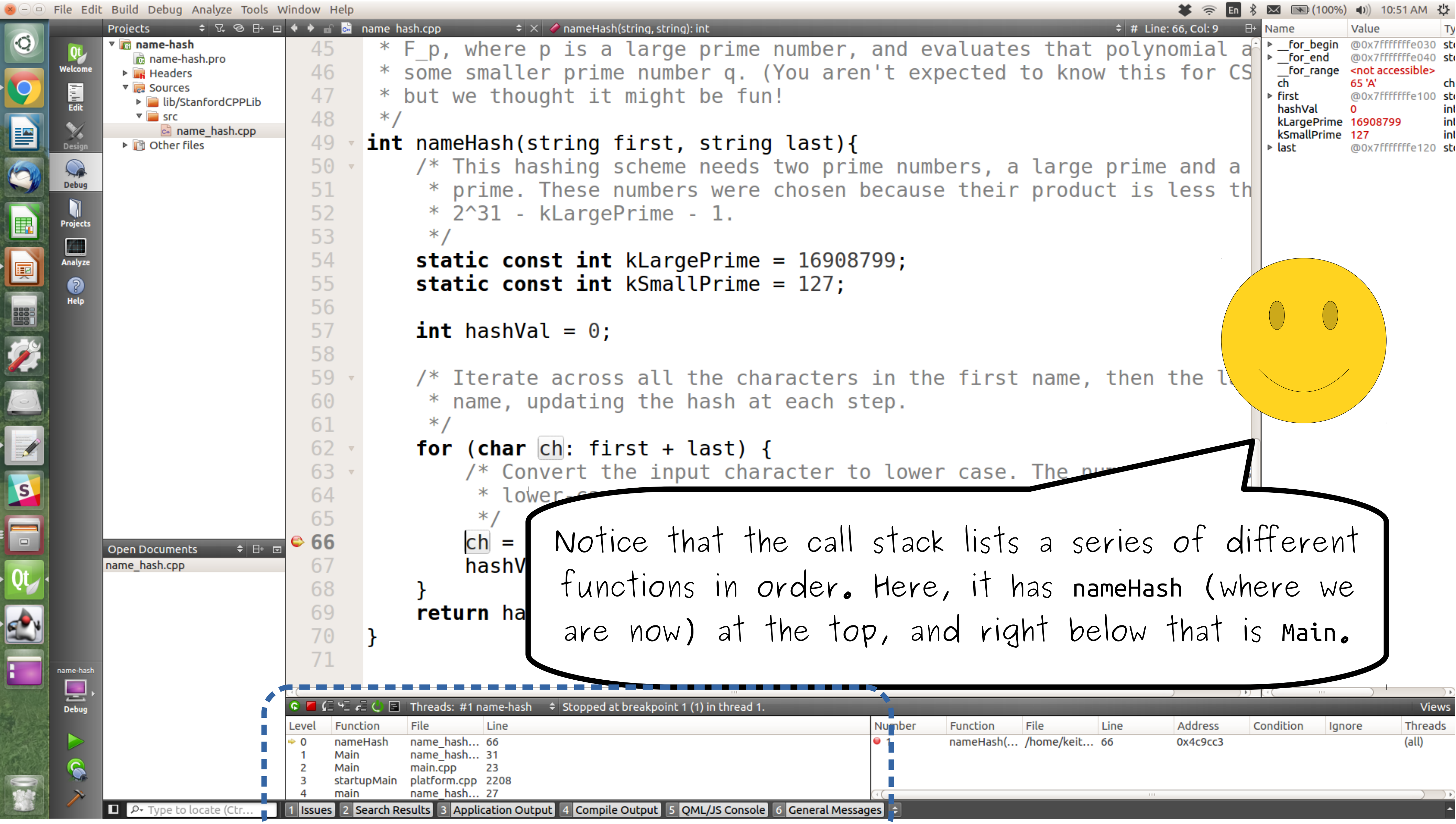
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

The call stack can tell us exactly that!



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

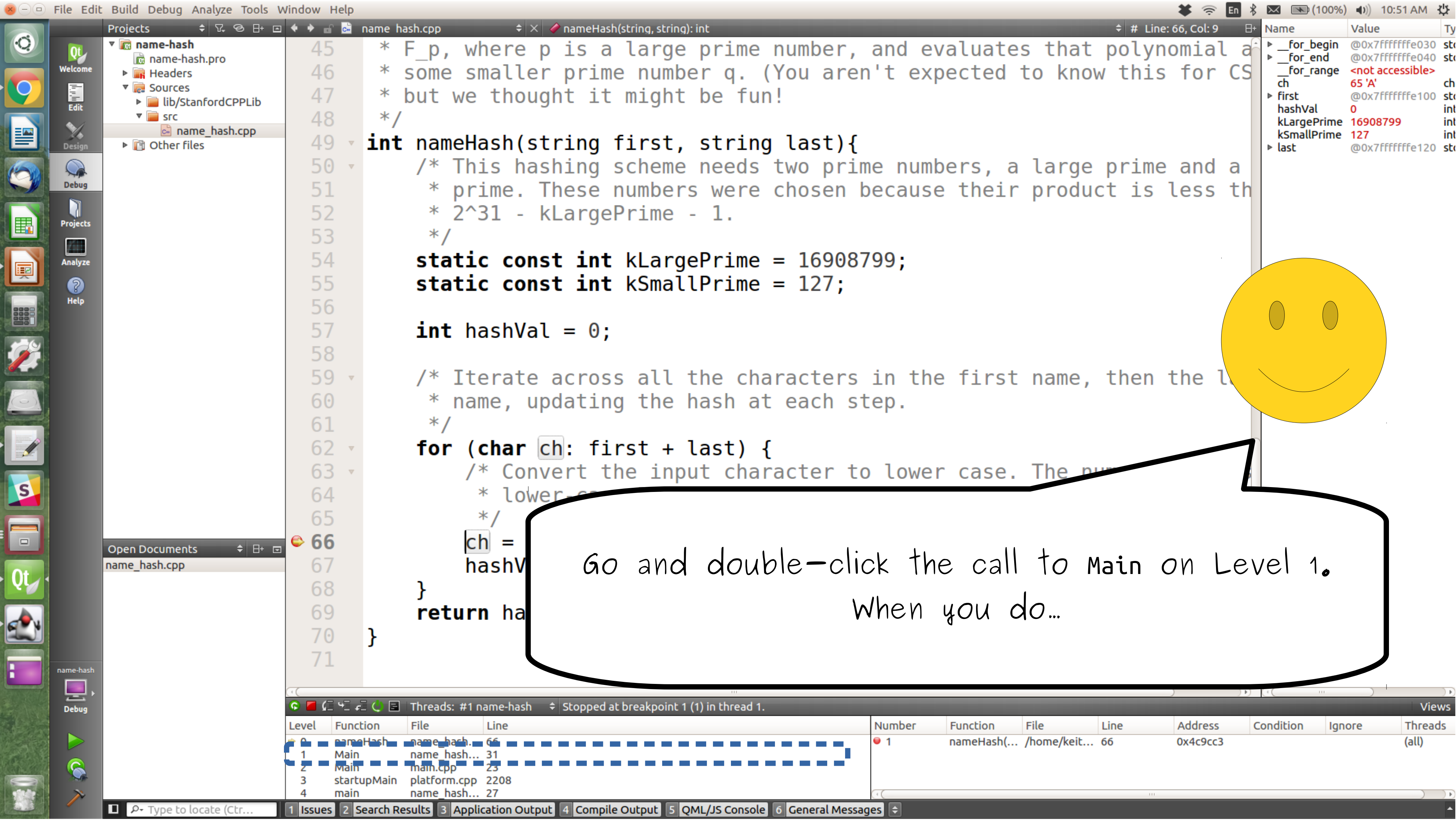
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Notice that the call stack lists a series of different functions in order. Here, it has **nameHash** (where we are now) at the top, and right below that is **Main**.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

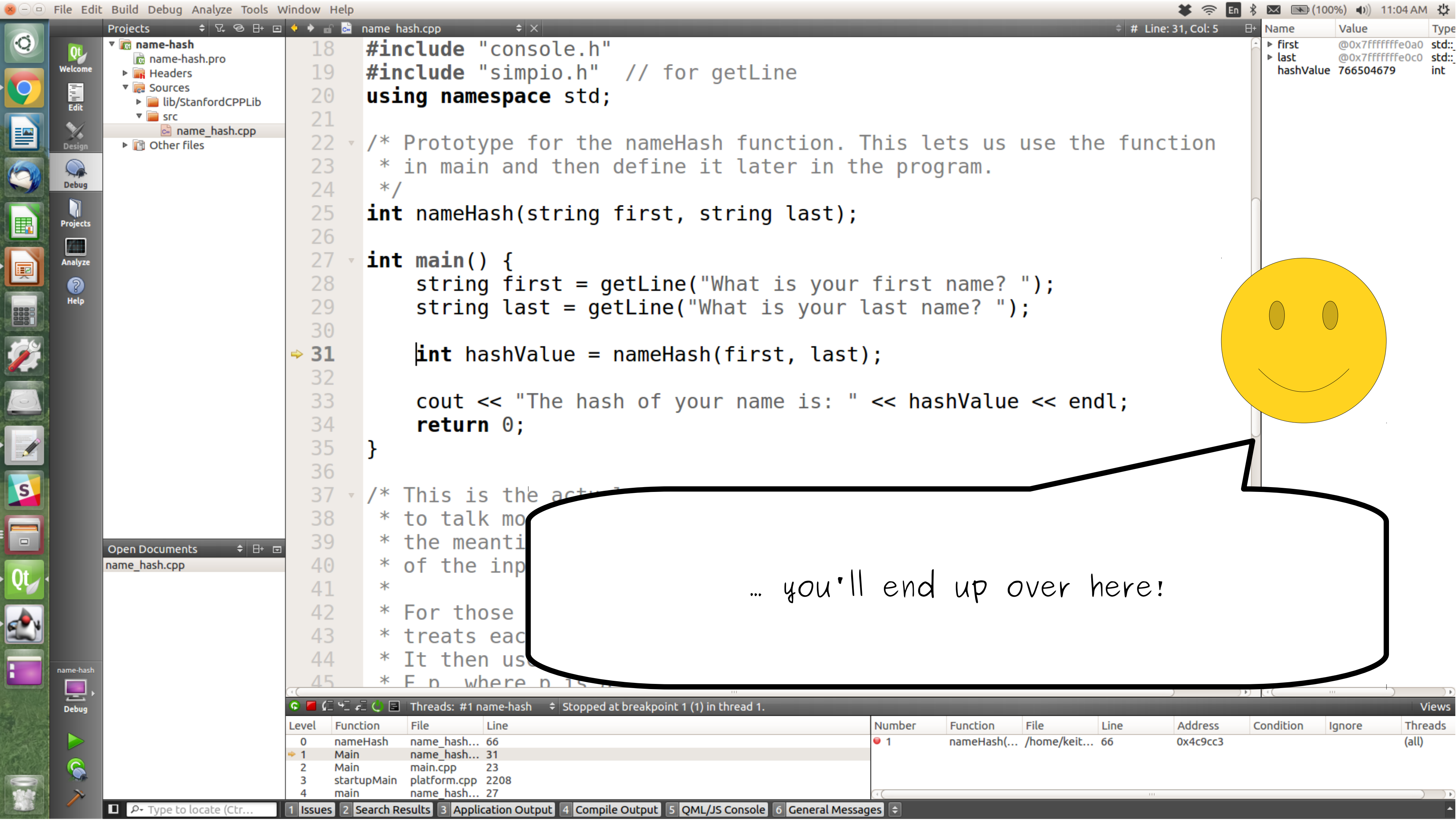
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Go and double-click the call to Main on Level 1. When you do…



```
#include "console.h"
#include "simpio.h"  // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the act
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F_n  where n is
```

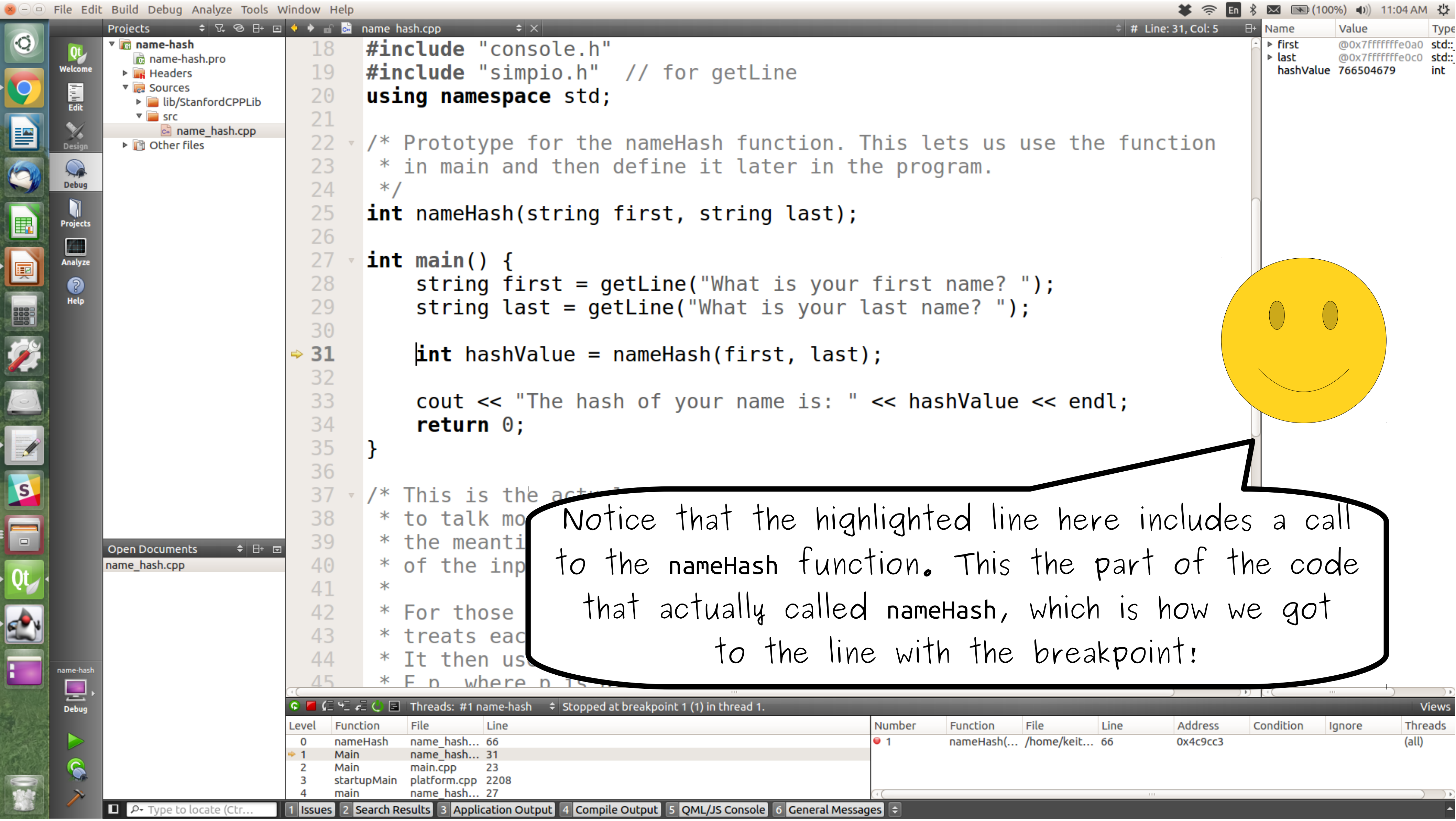
Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	766504679	int

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

… you'll end up over here!



```
#include "console.h"
#include "simpio.h"  // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the act
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F_n where n is
```

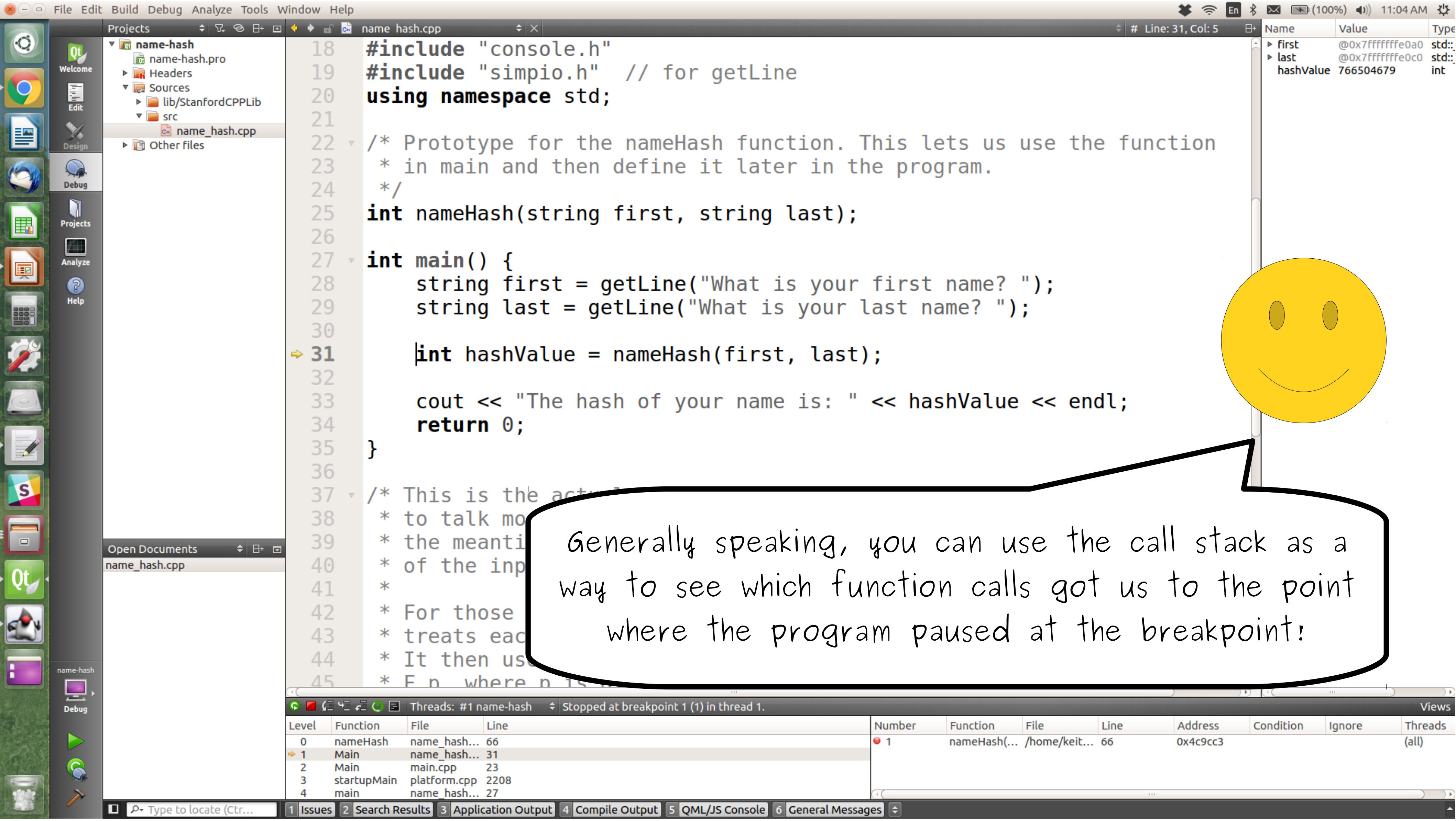
Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	766504679	int

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Notice that the highlighted line here includes a call to the **nameHash** function. This the part of the code that actually called **nameHash**, which is how we got to the line with the breakpoint!



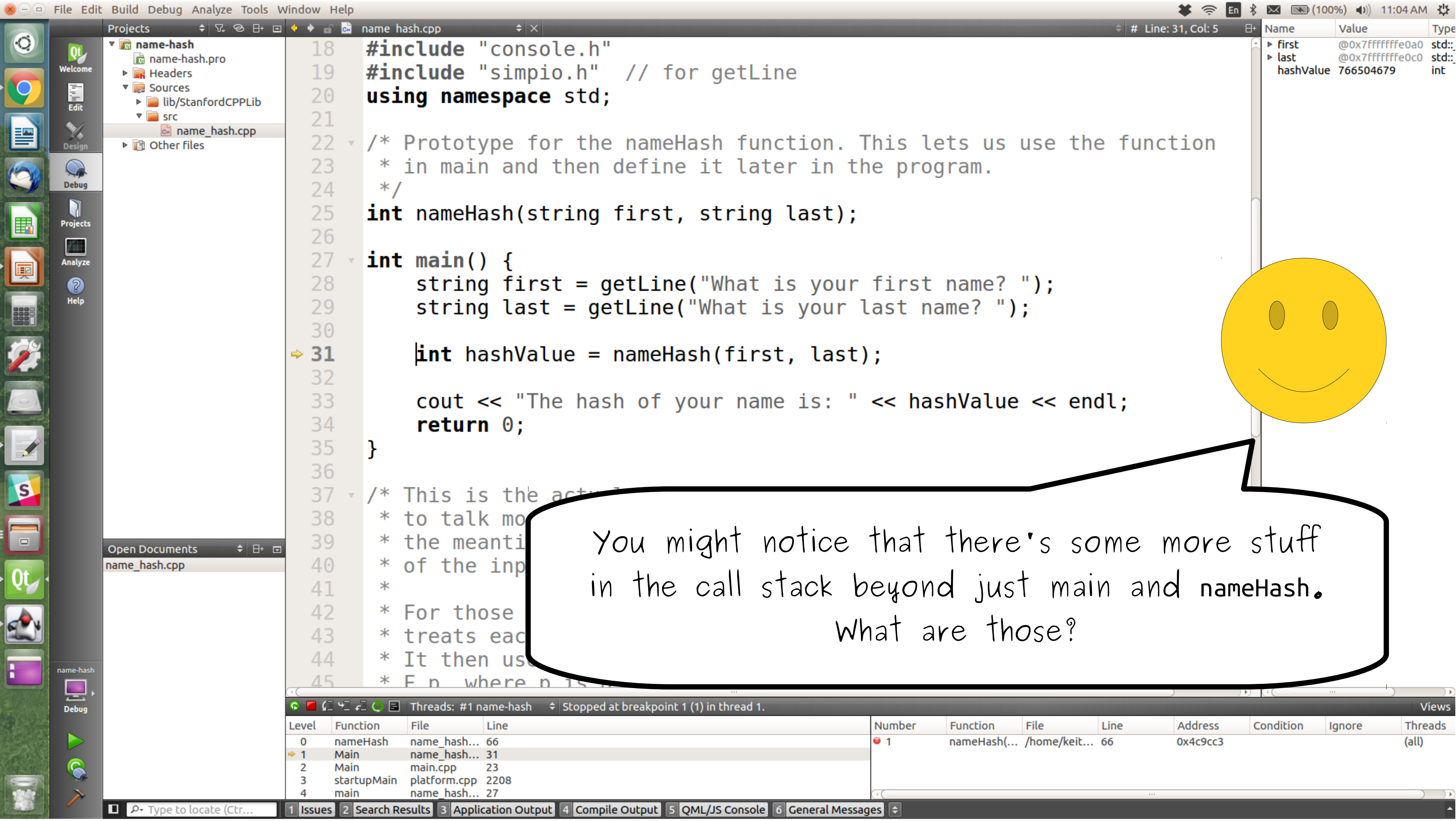
```
18  #include "console.h"
19  #include "simpio.h"  // for getLine
20  using namespace std;
21
22  /* Prototype for the nameHash function. This lets us use the function
23   * in main and then define it later in the program.
24   */
25  int nameHash(string first, string last);
26
27  int main() {
28      string first = getLine("What is your first name? ");
29      string last = getLine("What is your last name? ");
30
31      int hashValue = nameHash(first, last);
32
33      cout << "The hash of your name is: " << hashValue << endl;
34      return 0;
35  }
36
37  /* This is the act
38   * to talk mo
39   * the meanti
40   * of the inp
41   *
42   * For those
43   * treats eac
44   * It then us
45   * F n  where n is
```

Name	Value	Type
first	@0x7fffffffe0a0	std::...
last	@0x7fffffffe0c0	std::...
hashValue	766504679	int

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Generally speaking, you can use the call stack as a way to see which function calls got us to the point where the program paused at the breakpoint!



```
#include "console.h"
#include "simpio.h"  // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the act
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F n  where n is
```

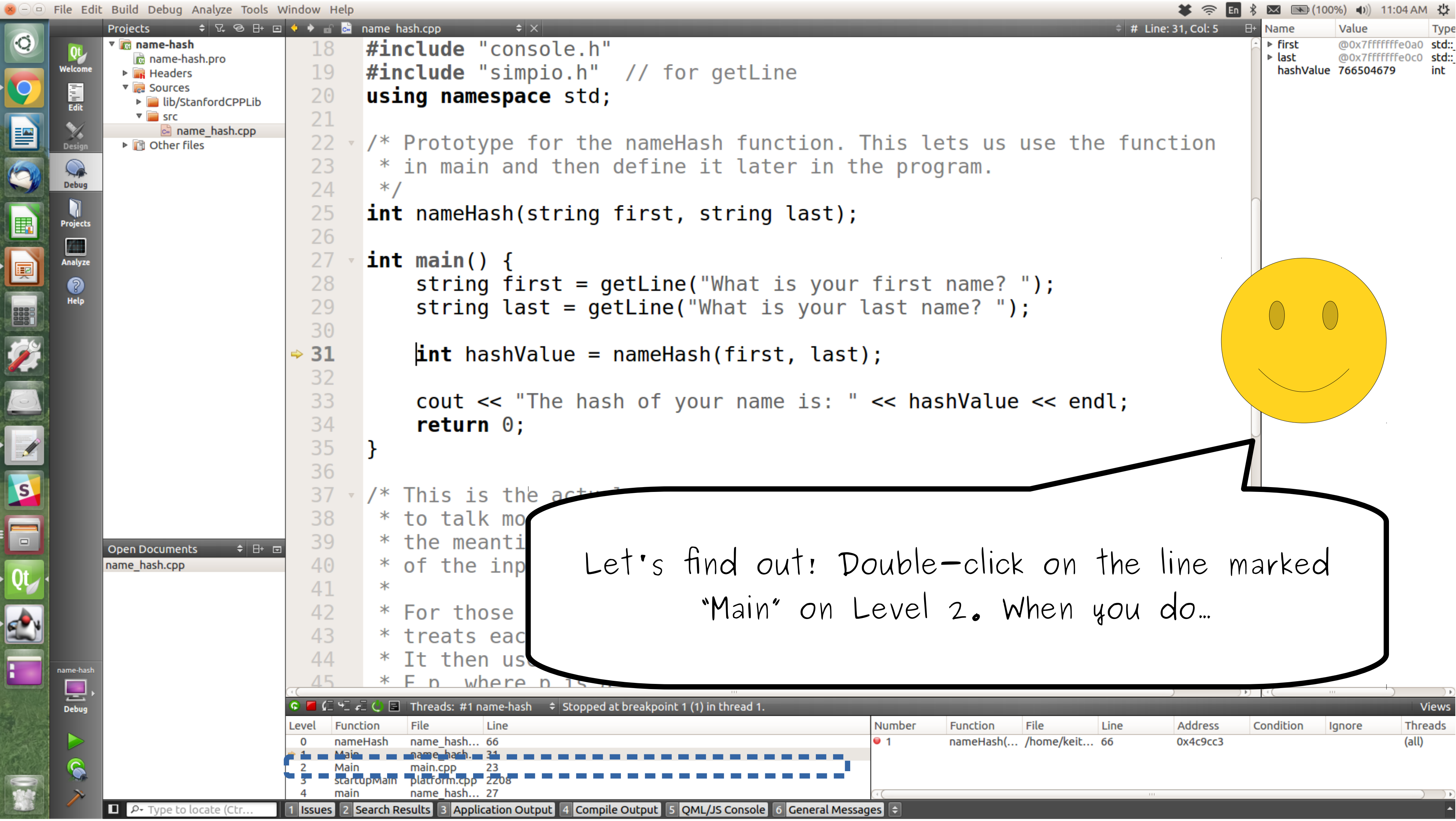
Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	766504679	int

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

You might notice that there's some more stuff in the call stack beyond just main and nameHash. What are those?



```
#include "console.h"
#include "simpio.h"   // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the act
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F n  where n is
```

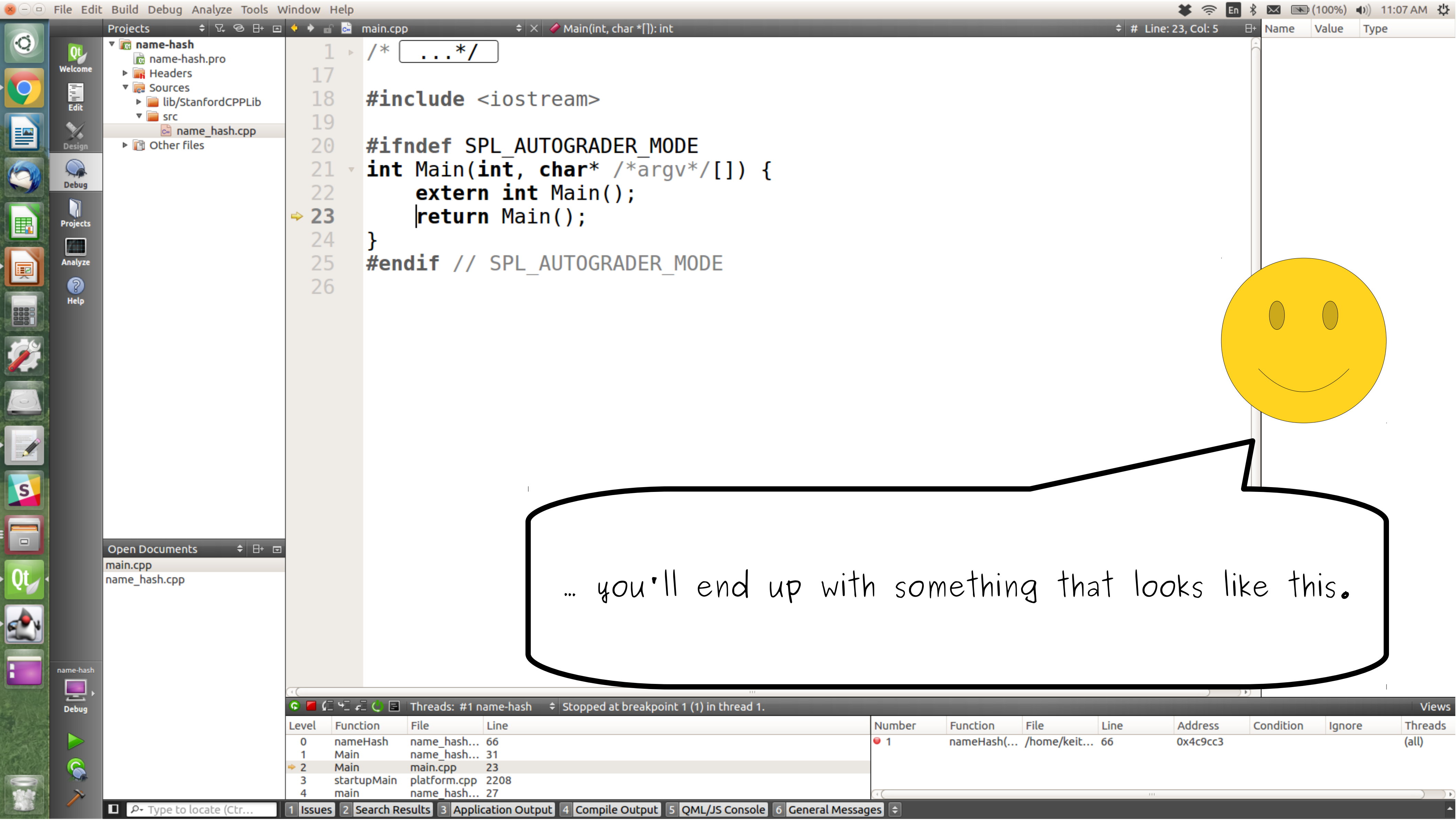
Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	766504679	int

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Let's find out! Double-click on the line marked "Main" on Level 2. When you do…



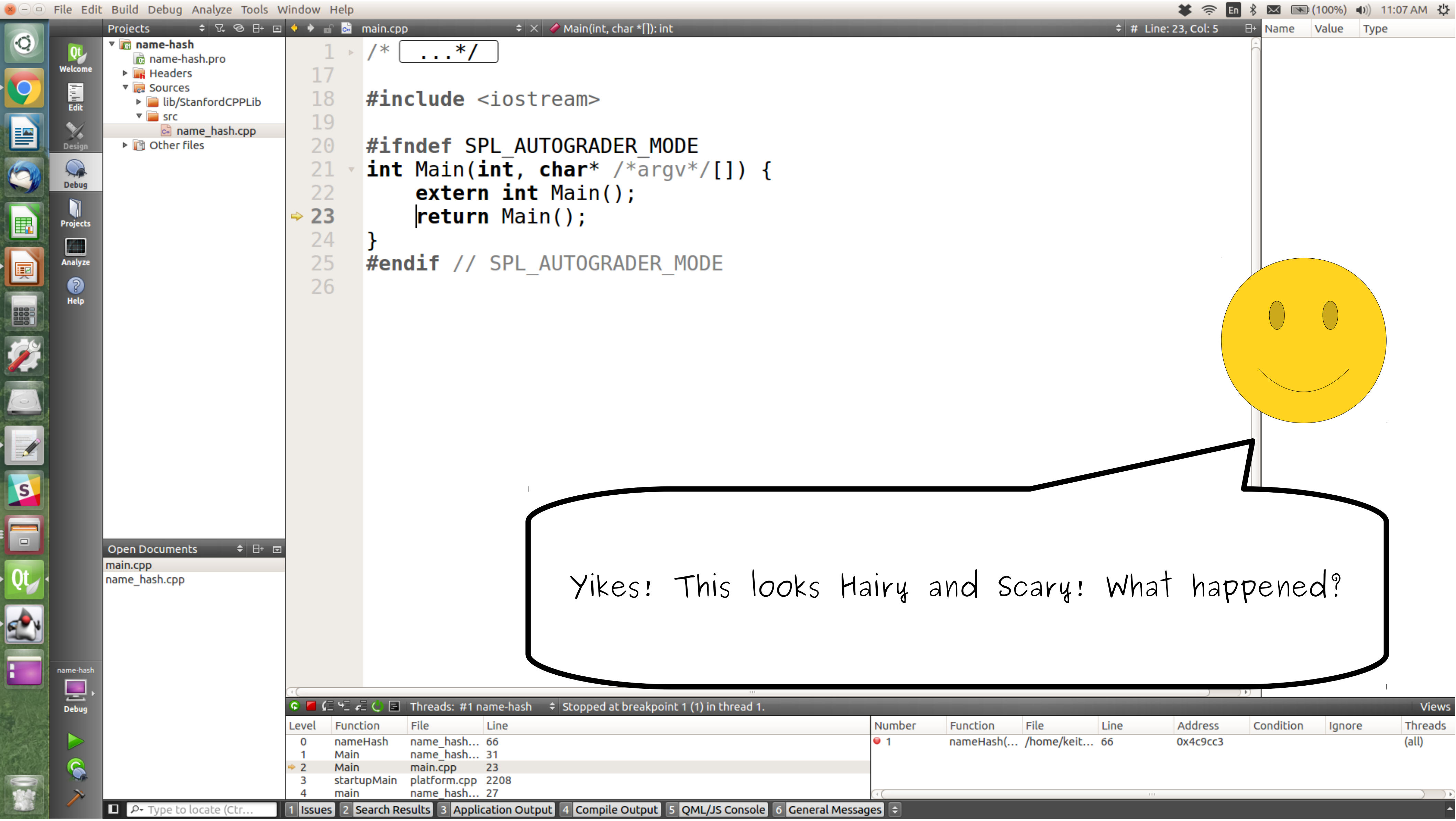
```
/* ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

... you'll end up with something that looks like this.

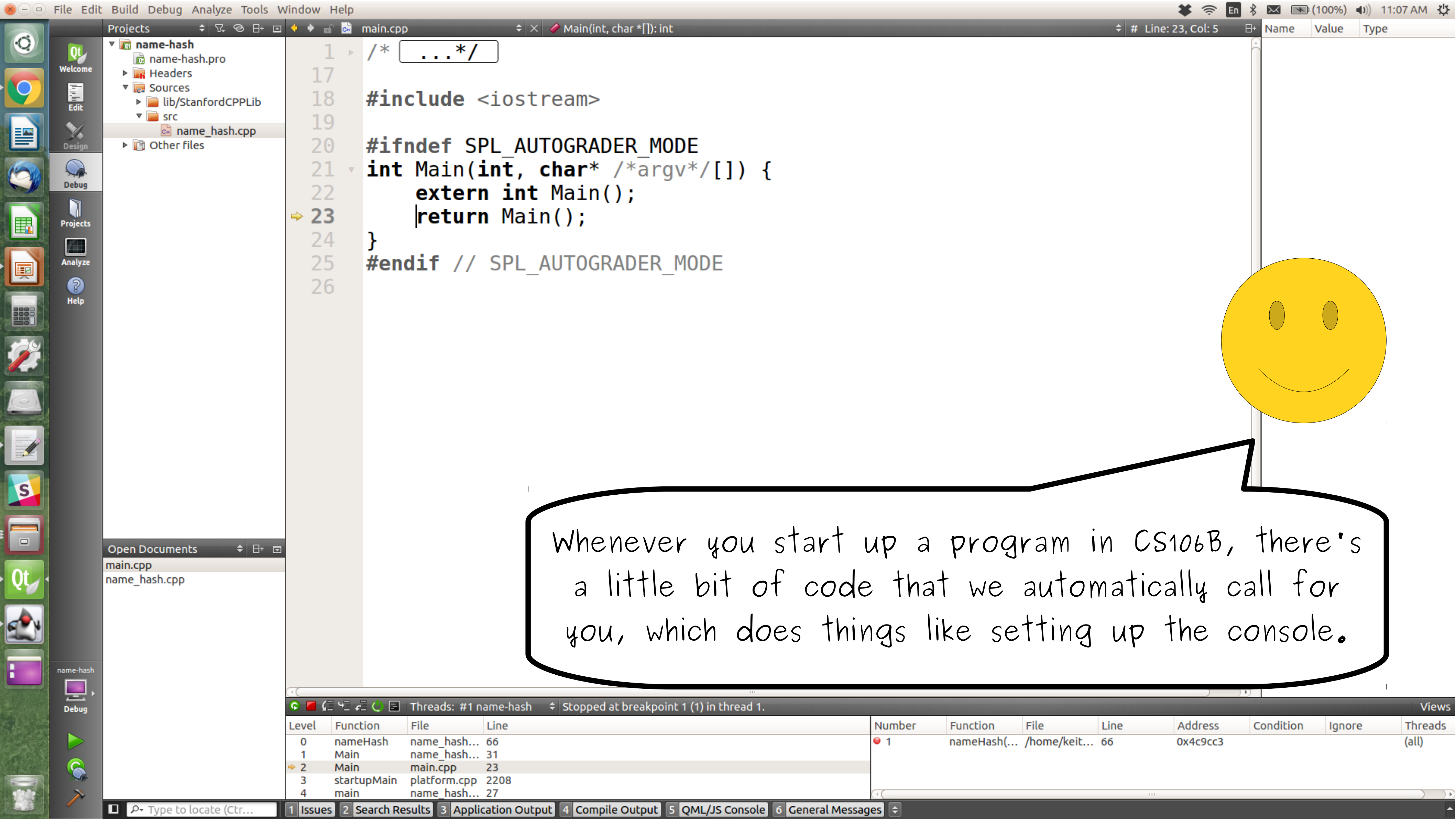


```
/* ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



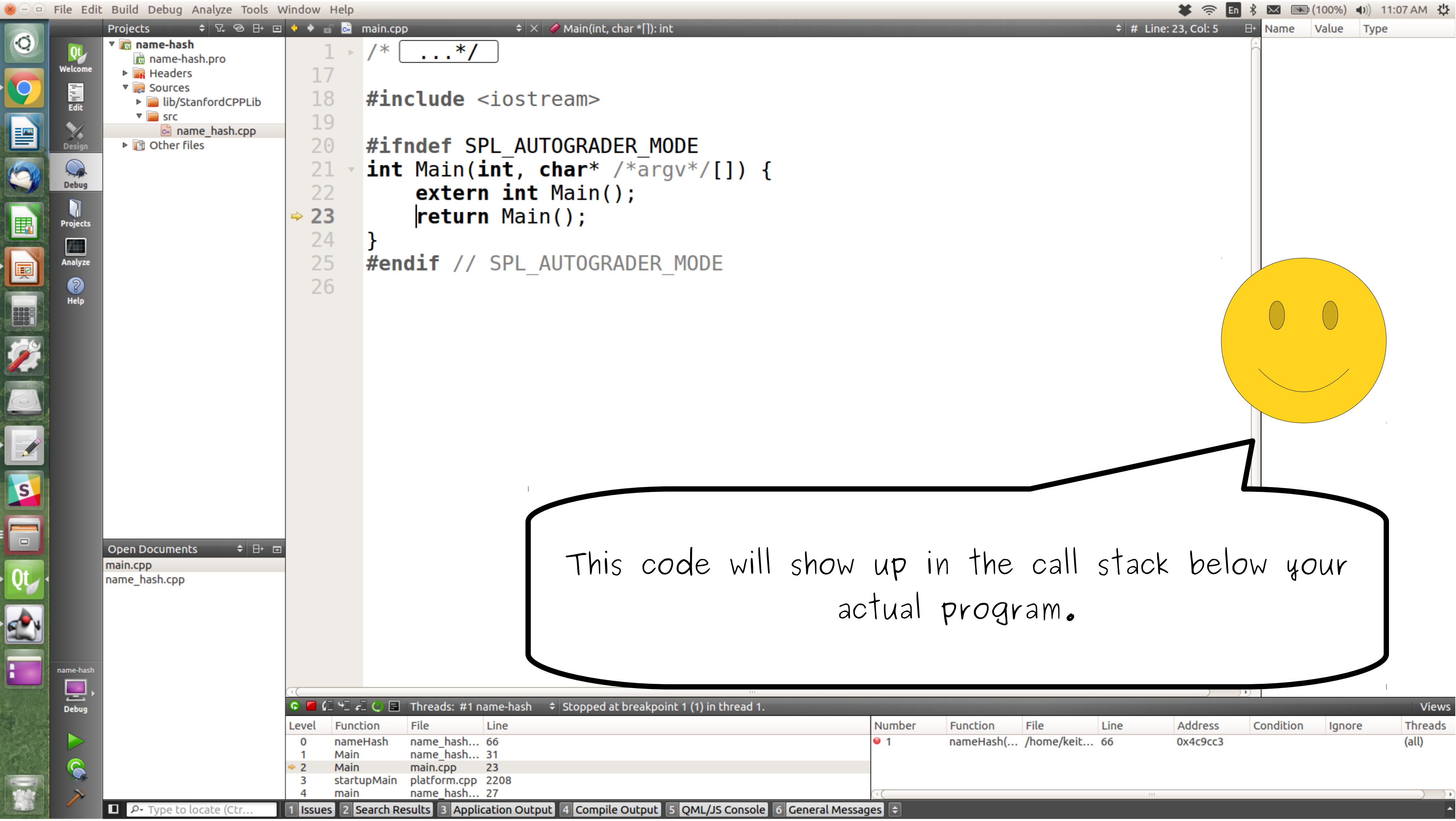
```
/* ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Whenever you start up a program in CS106B, there's a little bit of code that we automatically call for you, which does things like setting up the console.



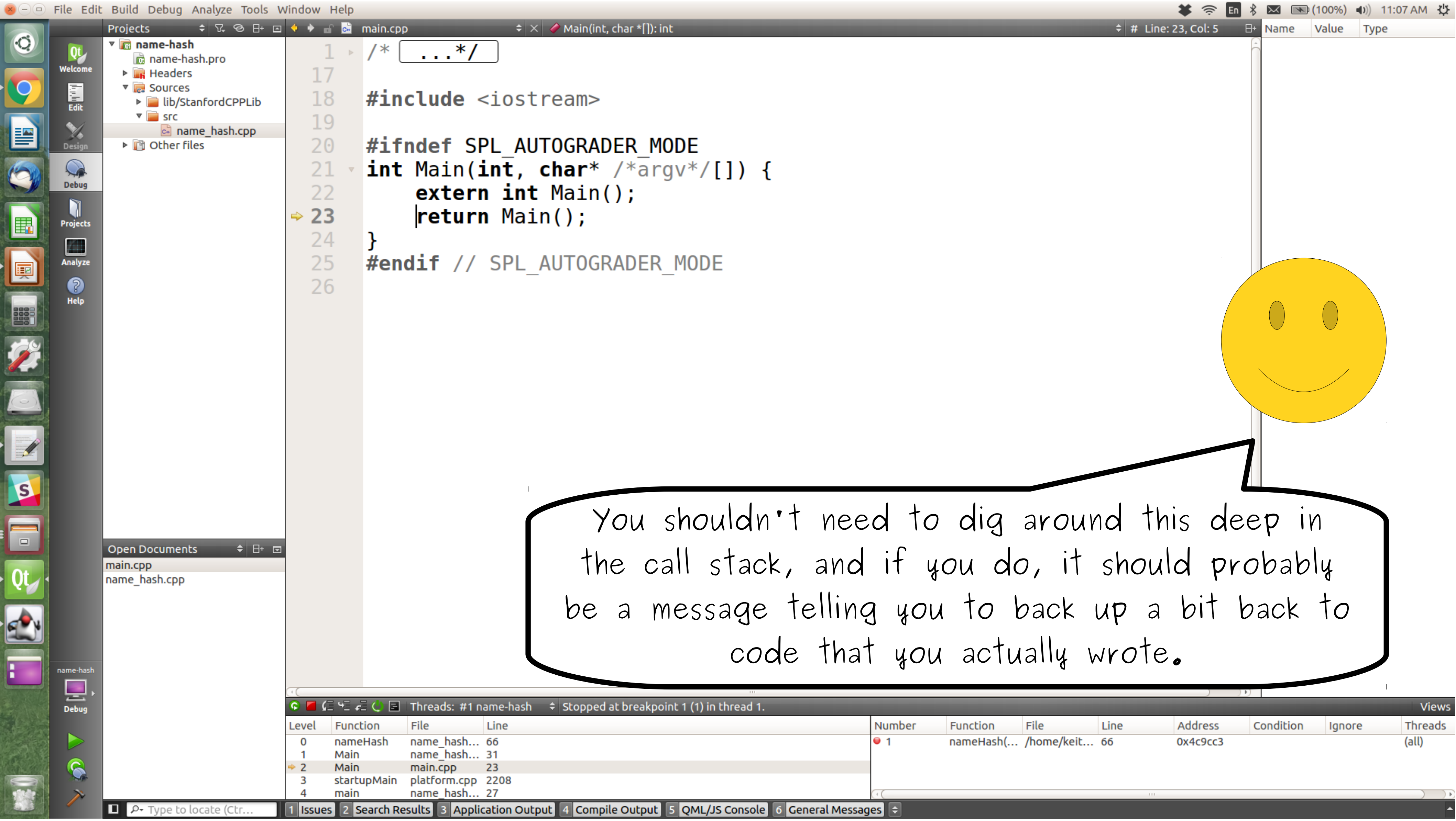
```
/*  ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

This code will show up in the call stack below your actual program.



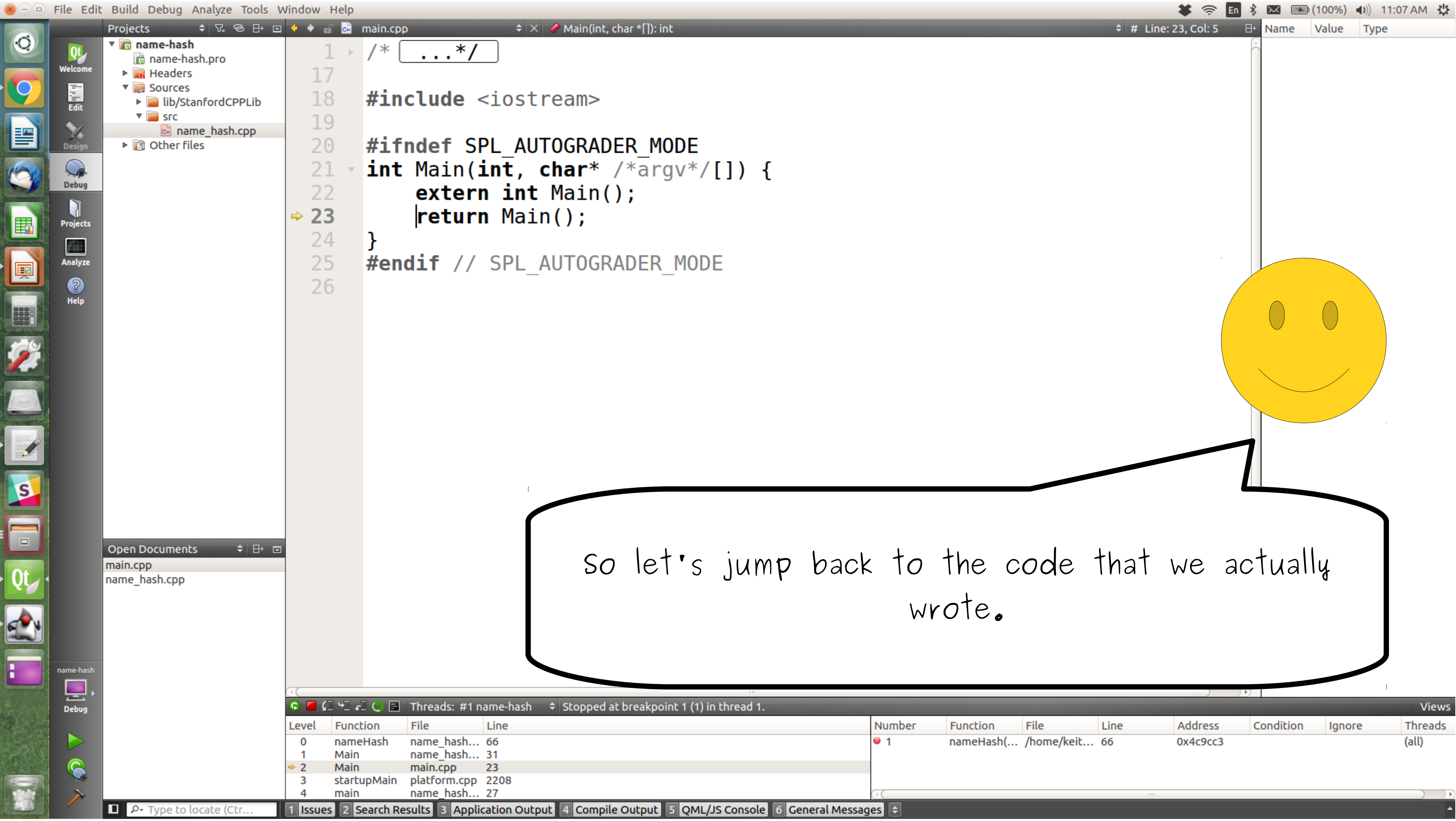
```
/* ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

You shouldn't need to dig around this deep in the call stack, and if you do, it should probably be a message telling you to back up a bit back to code that you actually wrote.

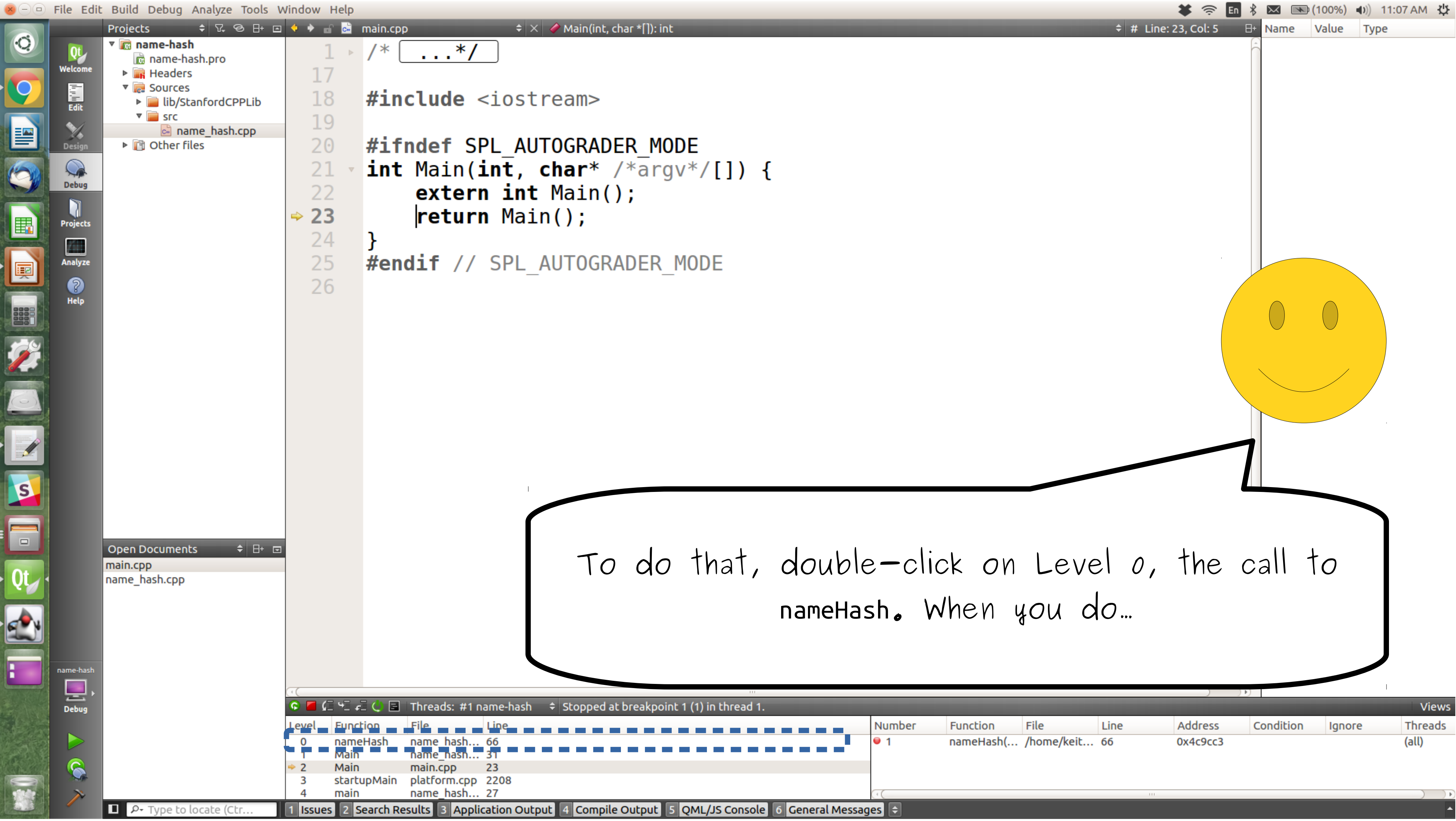


```
/* ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

So let's jump back to the code that we actually wrote.



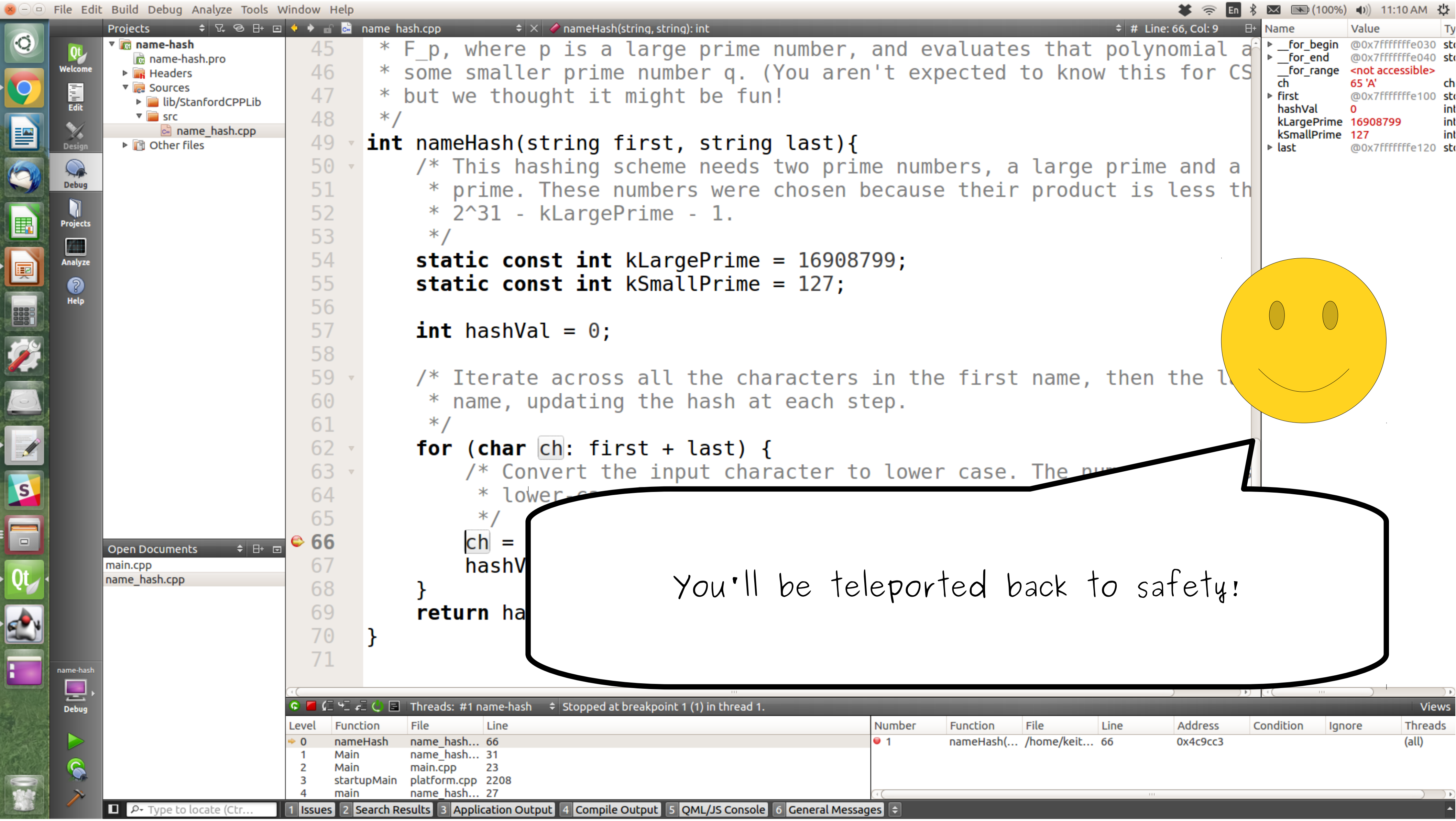
```
/* ...*/

#include <iostream>

#ifndef SPL_AUTOGRADER_MODE
int Main(int, char* /*argv*/[]) {
    extern int Main();
    return Main();
}
#endif // SPL_AUTOGRADER_MODE
```

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

To do that, double-click on Level 0, the call to nameHash. When you do…



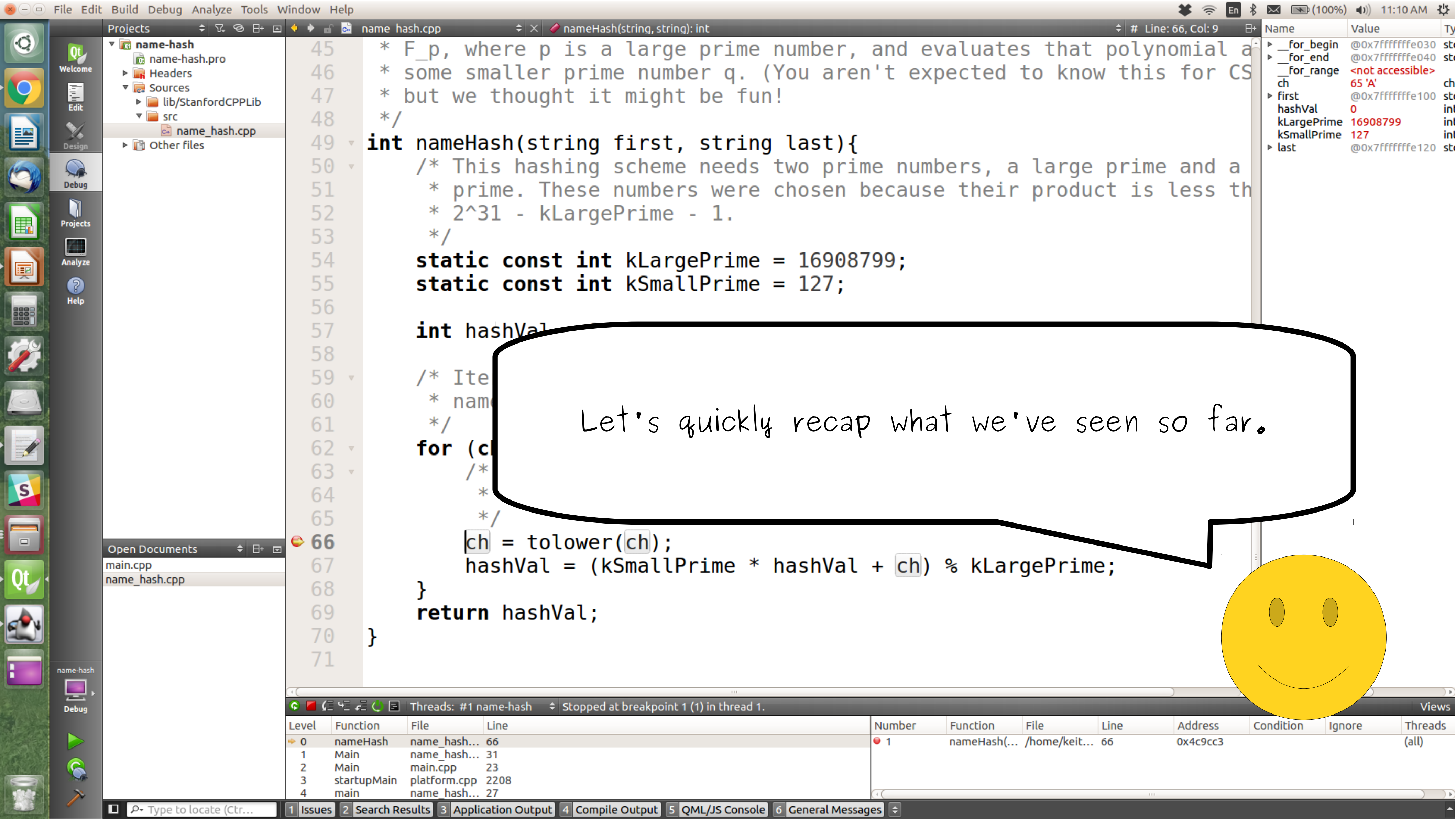
```
     * F_p, where p is a large prime number, and evaluates that polynomial a
     * some smaller prime number q. (You aren't expected to know this for CS
     * but we thought it might be fun!
     */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the l
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The nu
         * lower-c
         */
        ch =
        hashV
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal

    /* Ite
     * name
     */
    for (c
        /*
         *
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

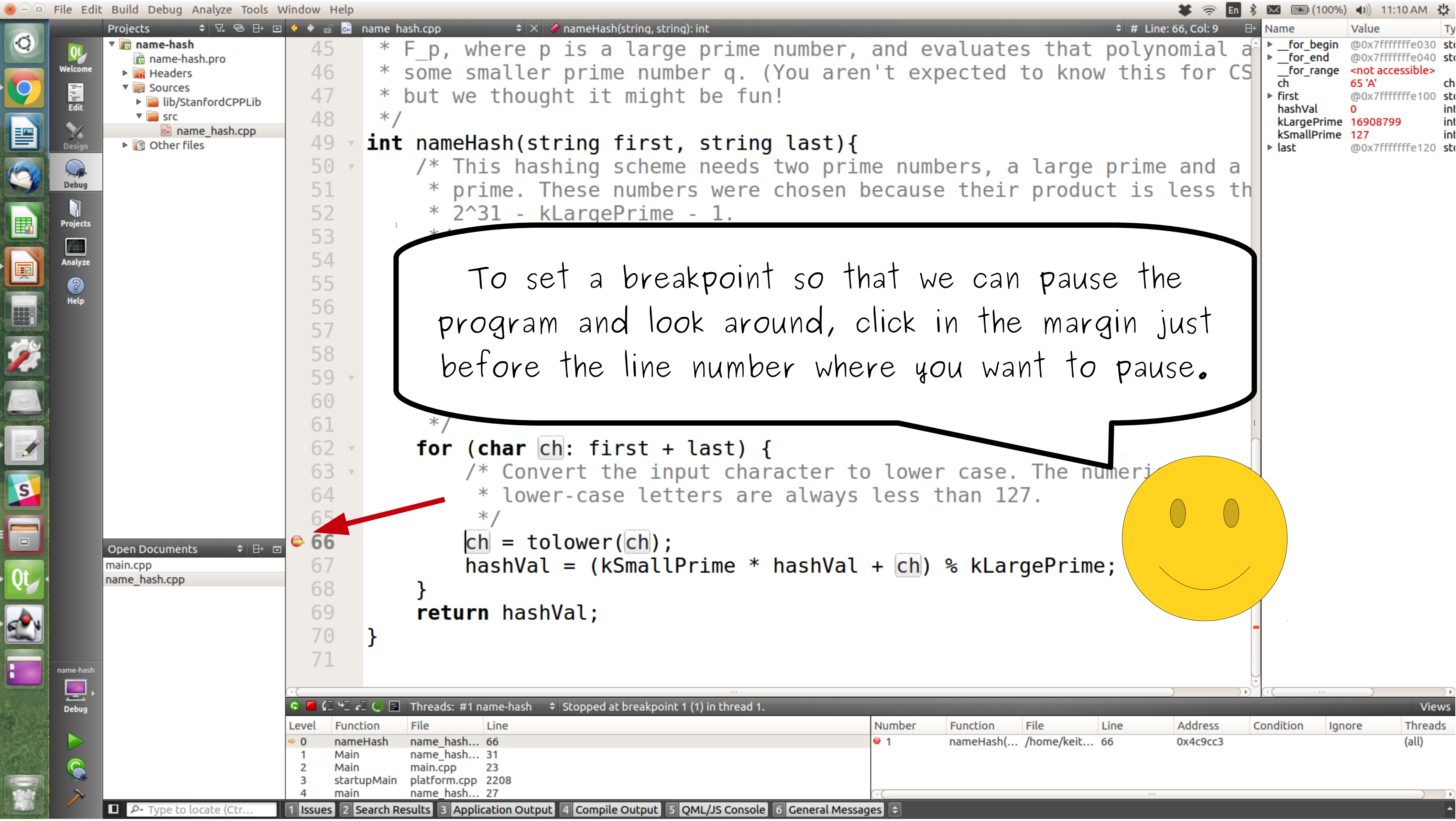
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Let's quickly recap what we've seen so far.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     *
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeri
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

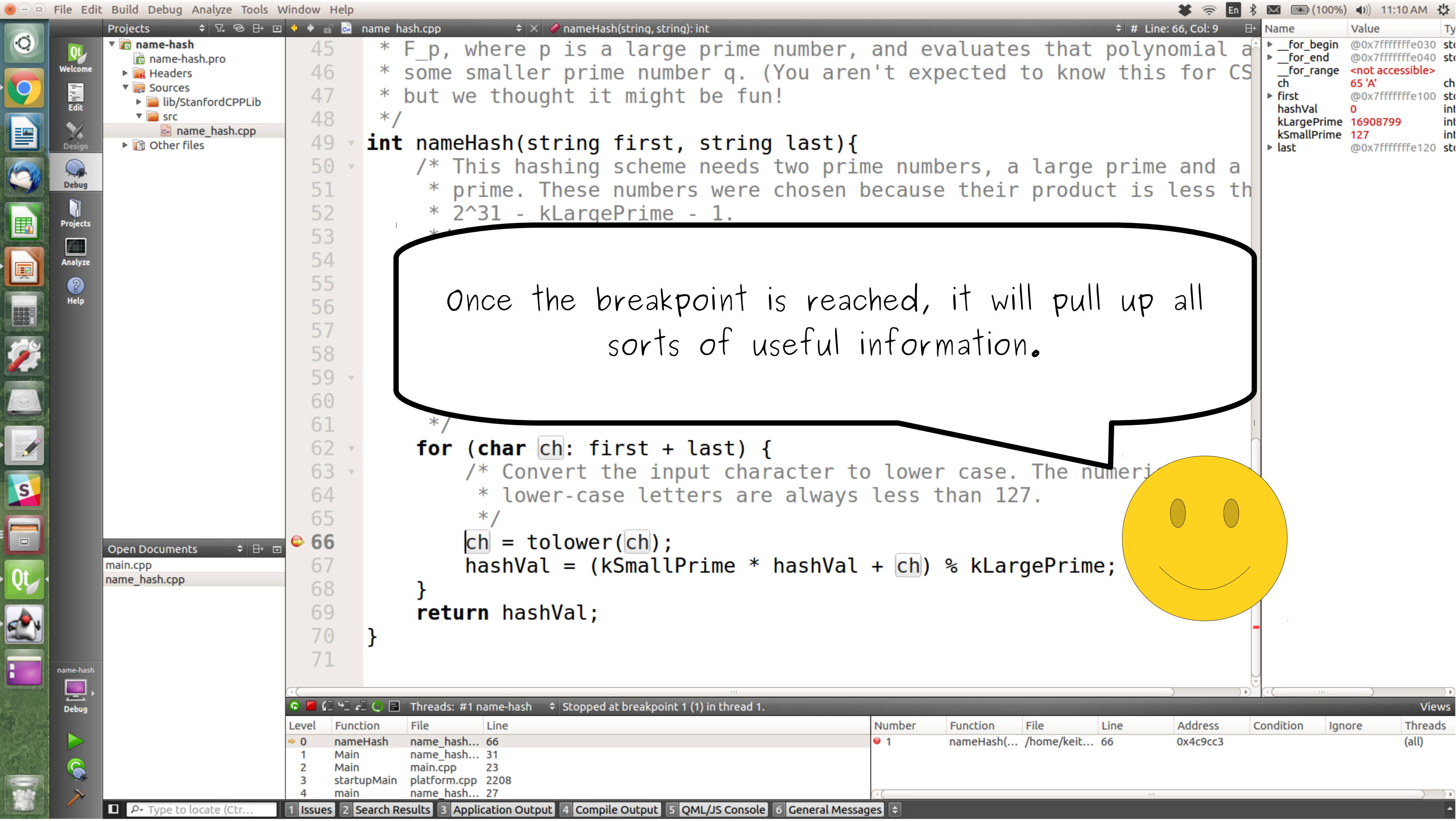
Threads: #1 name-hash

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

To set a breakpoint so that we can pause the program and look around, click in the margin just before the line number where you want to pause.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     *

     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeri
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

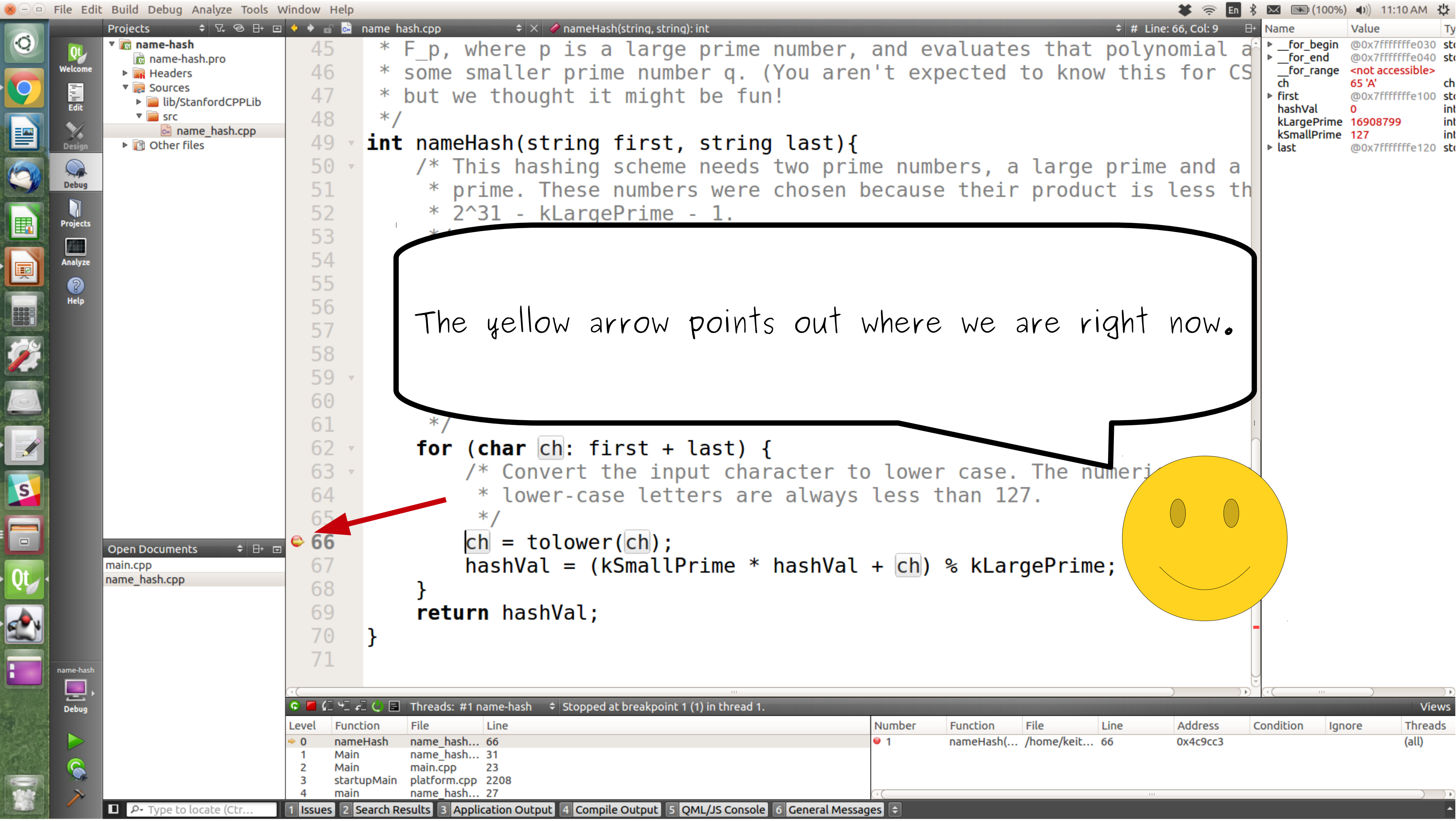
Once the breakpoint is reached, it will pull up all sorts of useful information.

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeri
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

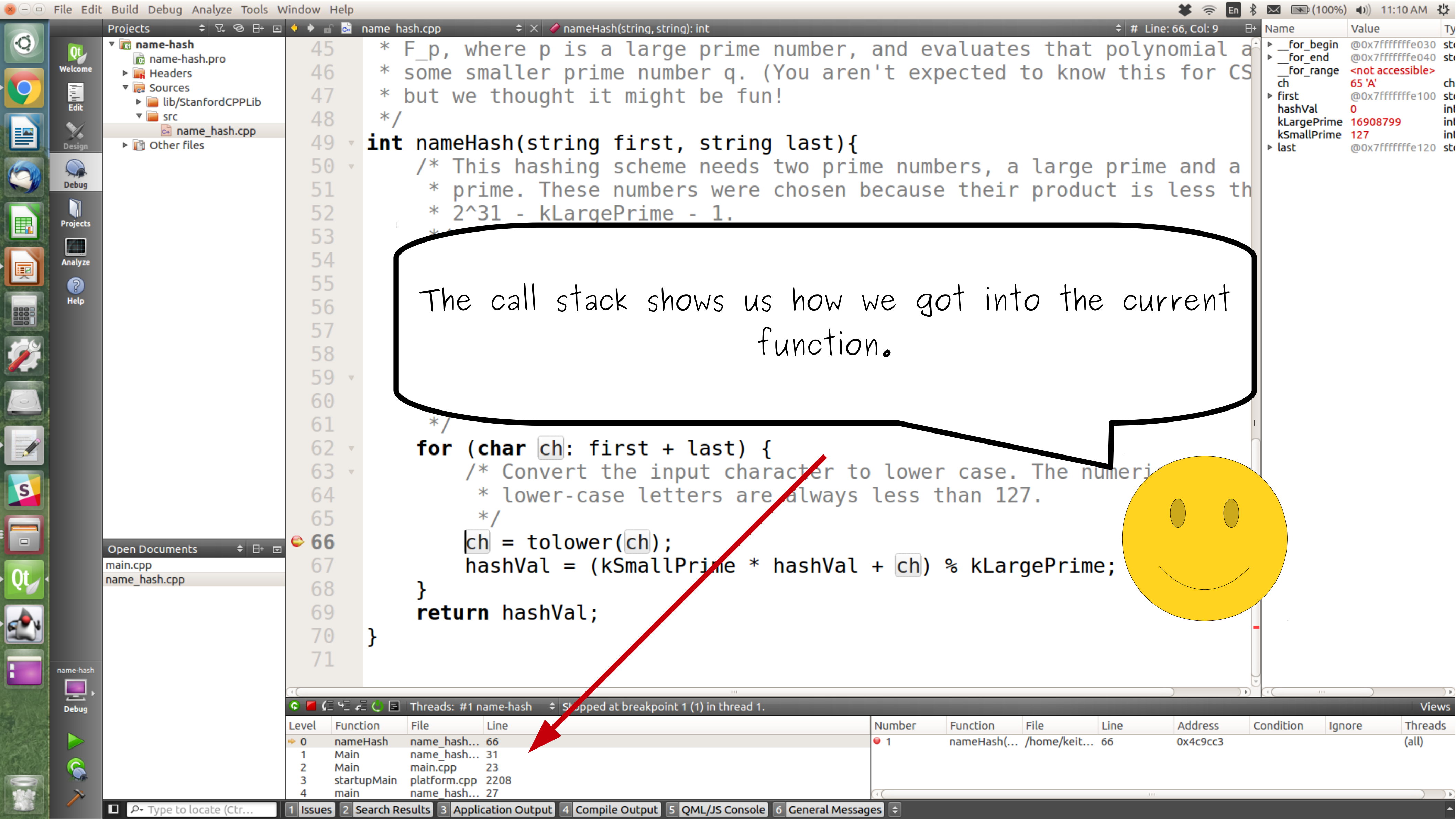
The yellow arrow points out where we are right now.

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



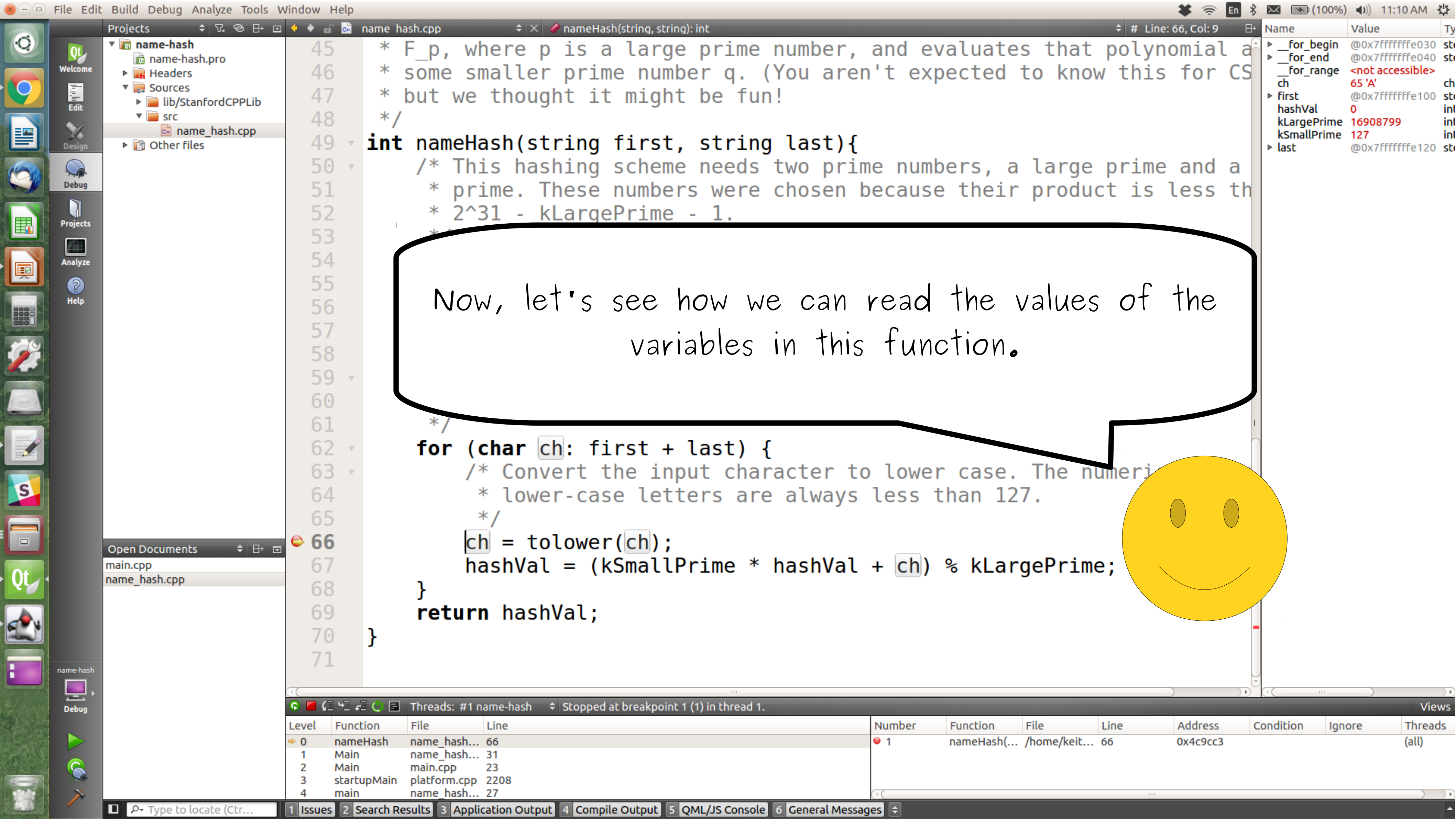
The call stack shows us how we got into the current function.

```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     *
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeri
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Now, let's see how we can read the values of the variables in this function.

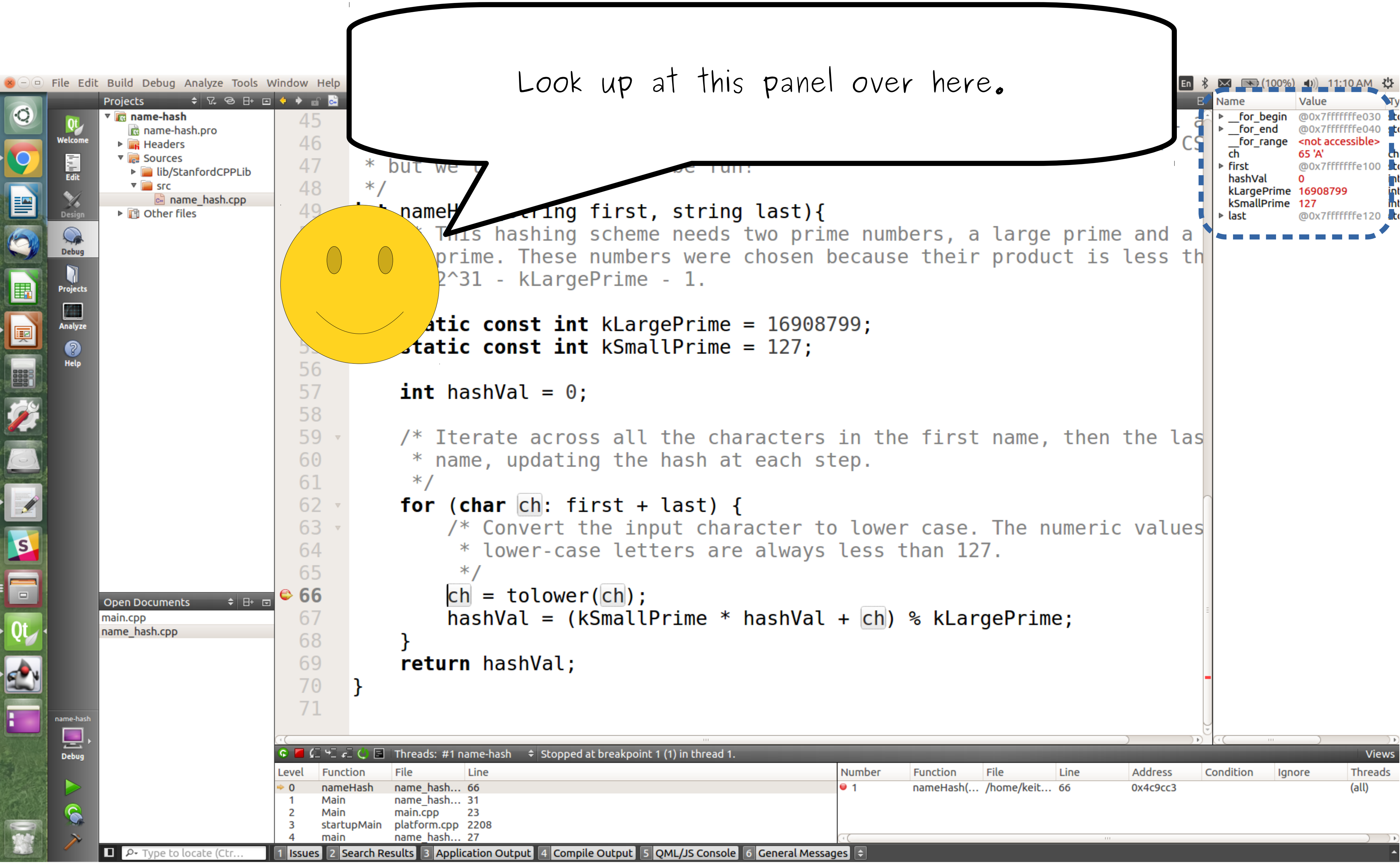
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeri
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Look up at this panel over here.

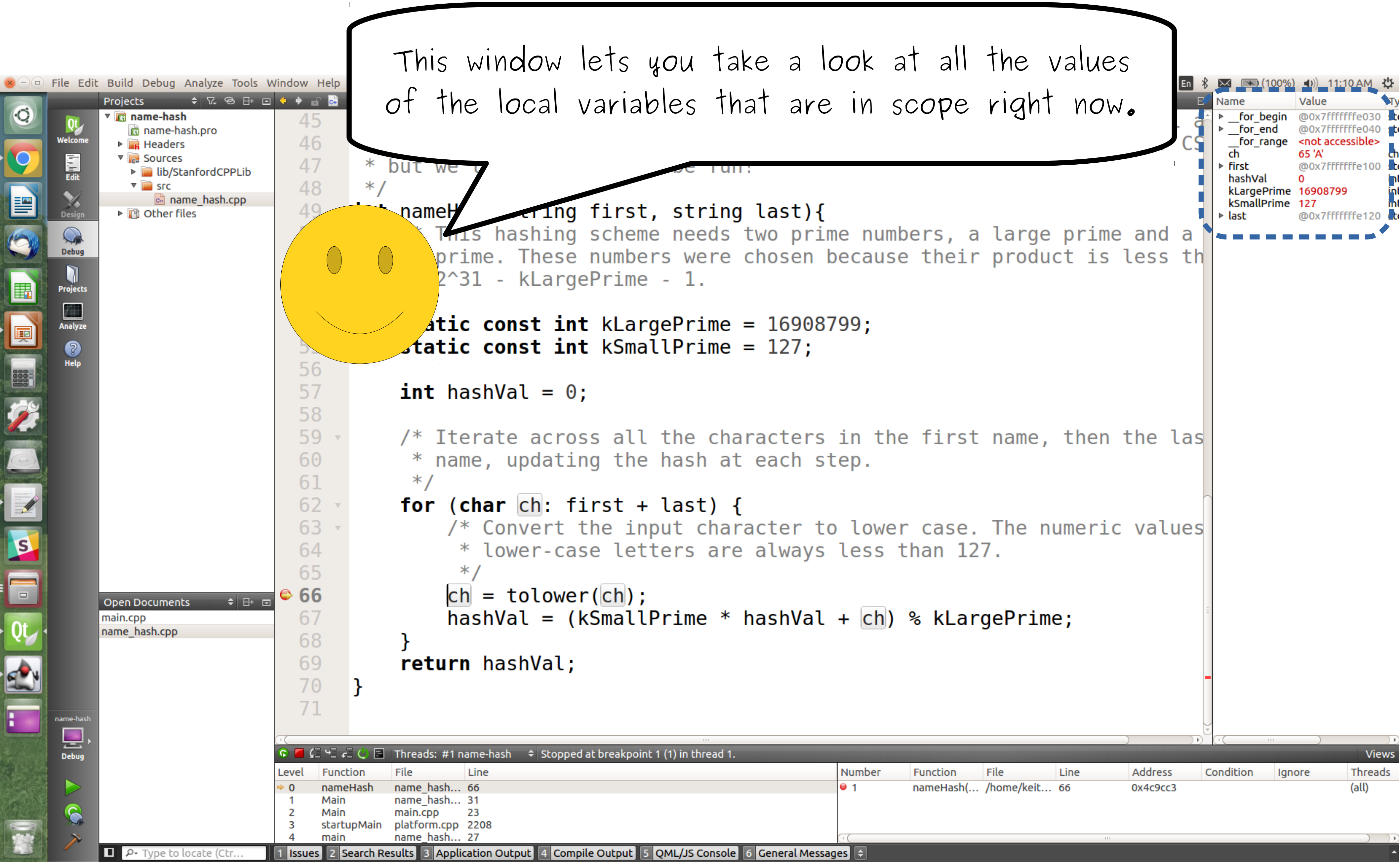
```
 * but we c                    be run.
 */
     nameH     ring first, string last){
   *  This hashing scheme needs two prime numbers, a large prime and a
      prime. These numbers were chosen because their product is less th
      2^31 - kLargePrime - 1.

     atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

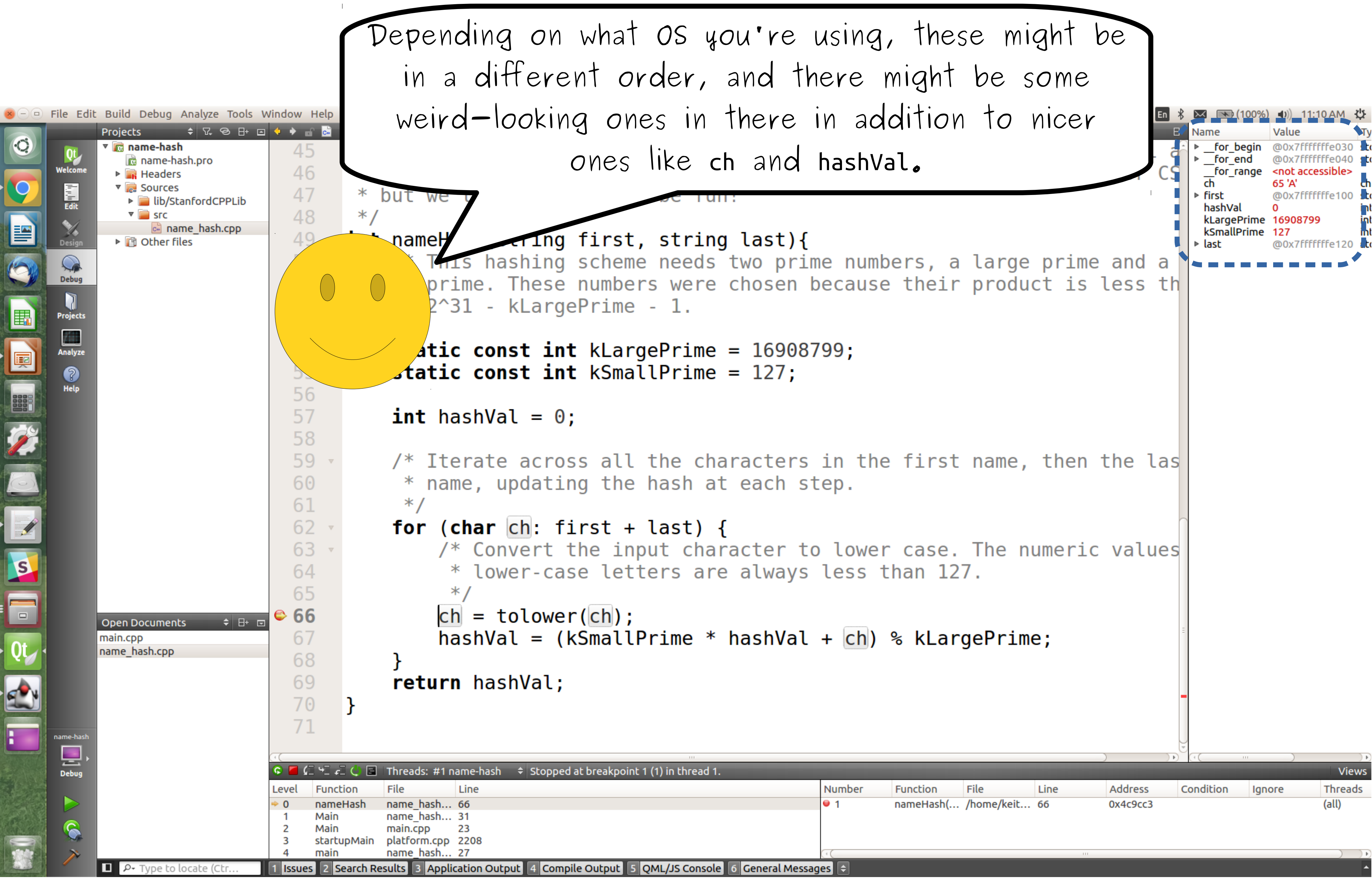
Name	Value
__for_begin	@0x7fffffffe030
__for_end	@0x7fffffffe040
__for_range	<not accessible>
ch	65 'A'
first	@0x7fffffffe100
hashVal	0
kLargePrime	16908799
kSmallPrime	127
last	@0x7fffffffe120

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)





Depending on what OS you're using, these might be in a different order, and there might be some weird-looking ones in there in addition to nicer ones like **ch** and **hashVal**.

```
 * but we c                  be run:
 */
      nameH      ing first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

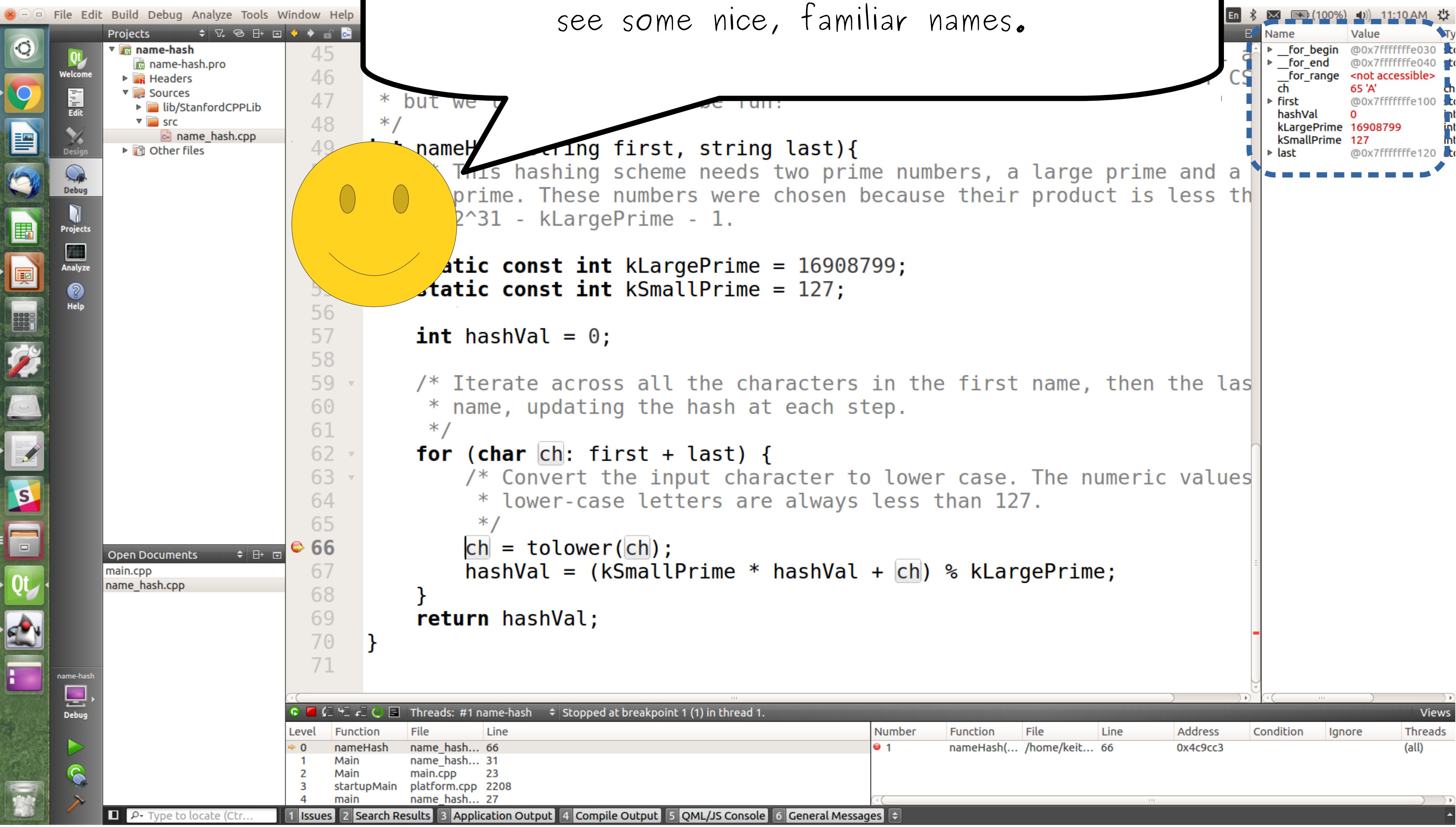
Name	Value
__for_begin	@0x7fffffffe030
__for_end	@0x7fffffffe040
__for_range	<not accessible>
ch	65 'A'
first	@0x7fffffffe100
hashVal	0
kLargePrime	16908799
kSmallPrime	127
last	@0x7fffffffe120

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

If we ignore the weird-looking ones, we can see some nice, familiar names.



```
 * but we c                    be run.
 */
    nameH      ring first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.

     atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

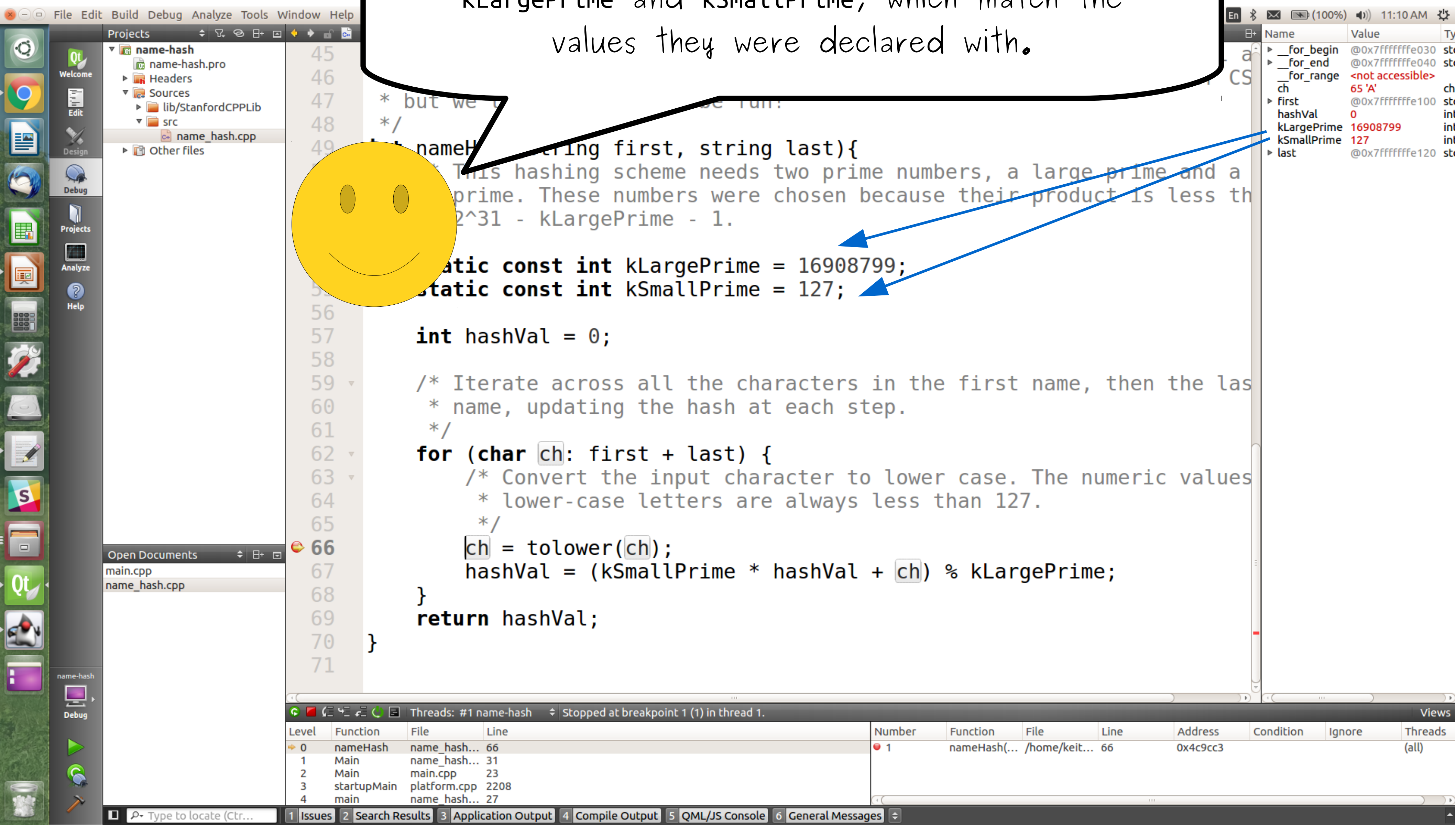
Name	Value
__for_begin	@0x7fffffffe030
__for_end	@0x7fffffffe040
__for_range	<not accessible>
ch	65 'A'
first	@0x7fffffffe100
hashVal	0
kLargePrime	16908799
kSmallPrime	127
last	@0x7fffffffe120

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

For example, here you can see the values of kLargePrime and kSmallPrime, which match the values they were declared with.



```
 * but we c                be run!
 */
   nameH     ring first, string last){
      This hashing scheme needs two prime numbers, a large prime and a
     prime. These numbers were chosen because their product is less th
     2^31 - kLargePrime - 1.

     atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

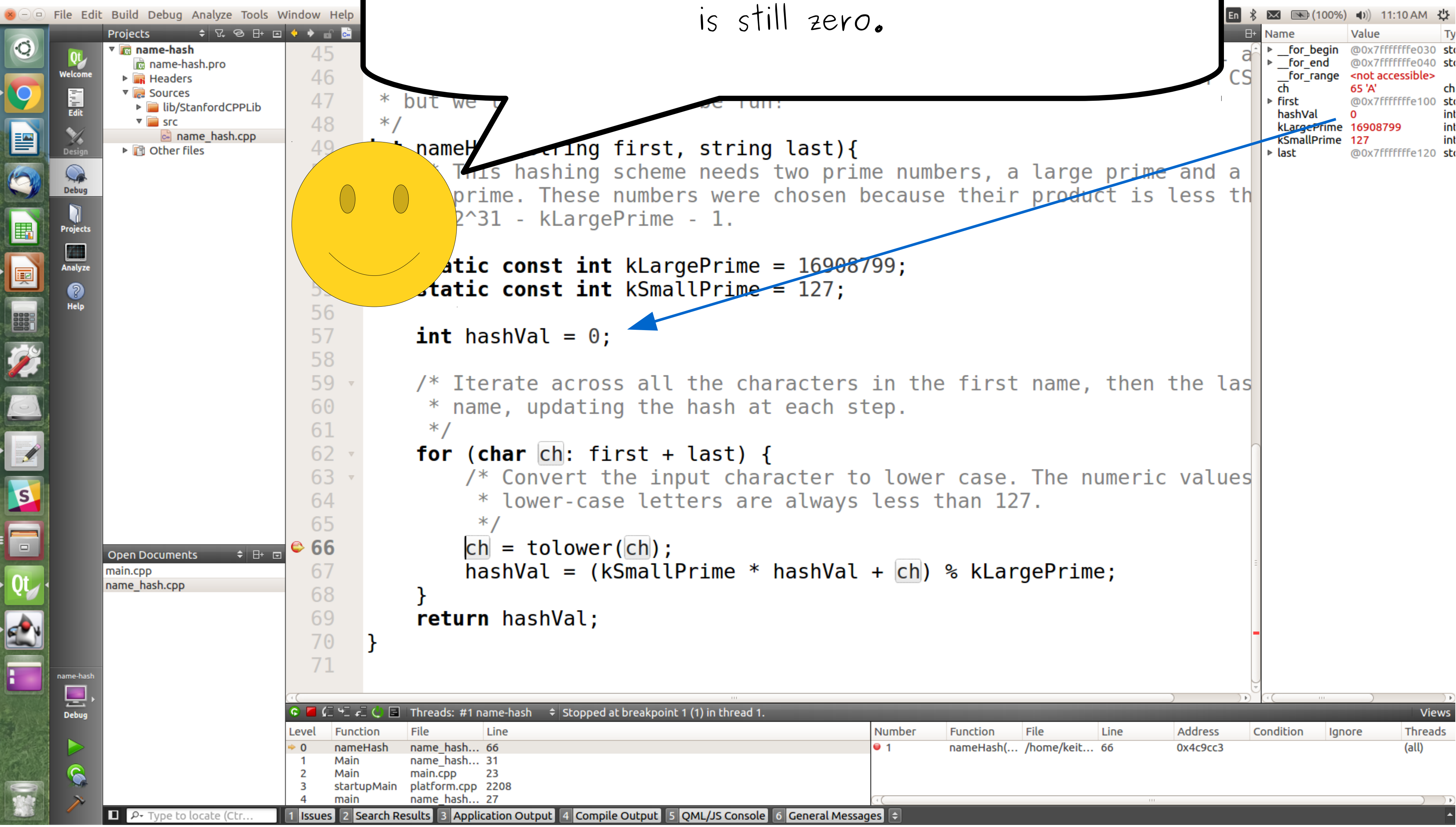
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

We can also see that, at this point, hashVal is still zero.



```
                                                                      a
                                                                    CS
 * but we c                          be run.
 */
   nameH     tring first, string last){
      This hashing scheme needs two prime numbers, a large prime and a
      prime. These numbers were chosen because their product is less th
      2^31 - kLargePrime - 1.

    atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

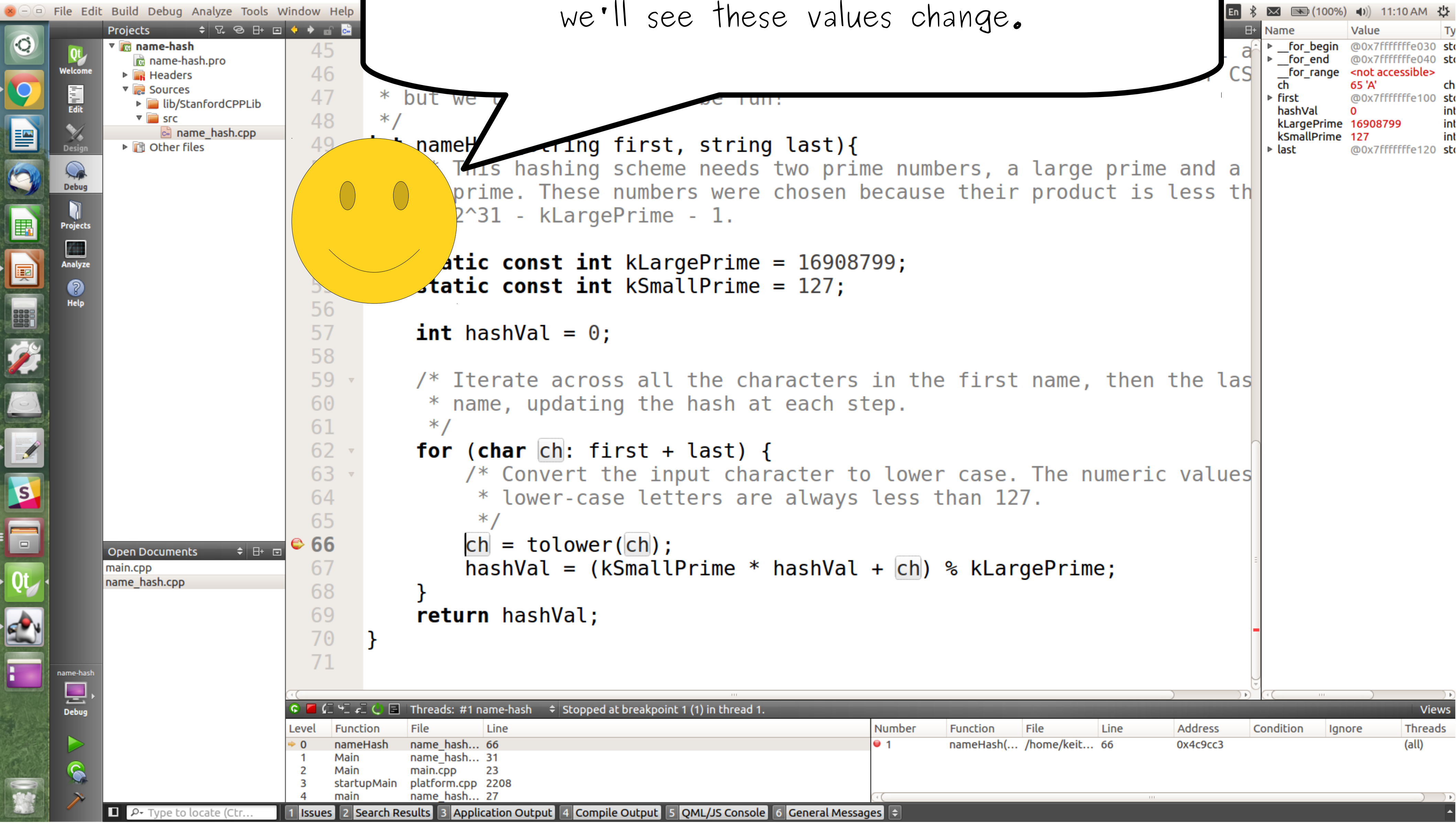
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

As we walk through the program one step at a time, we'll see these values change.



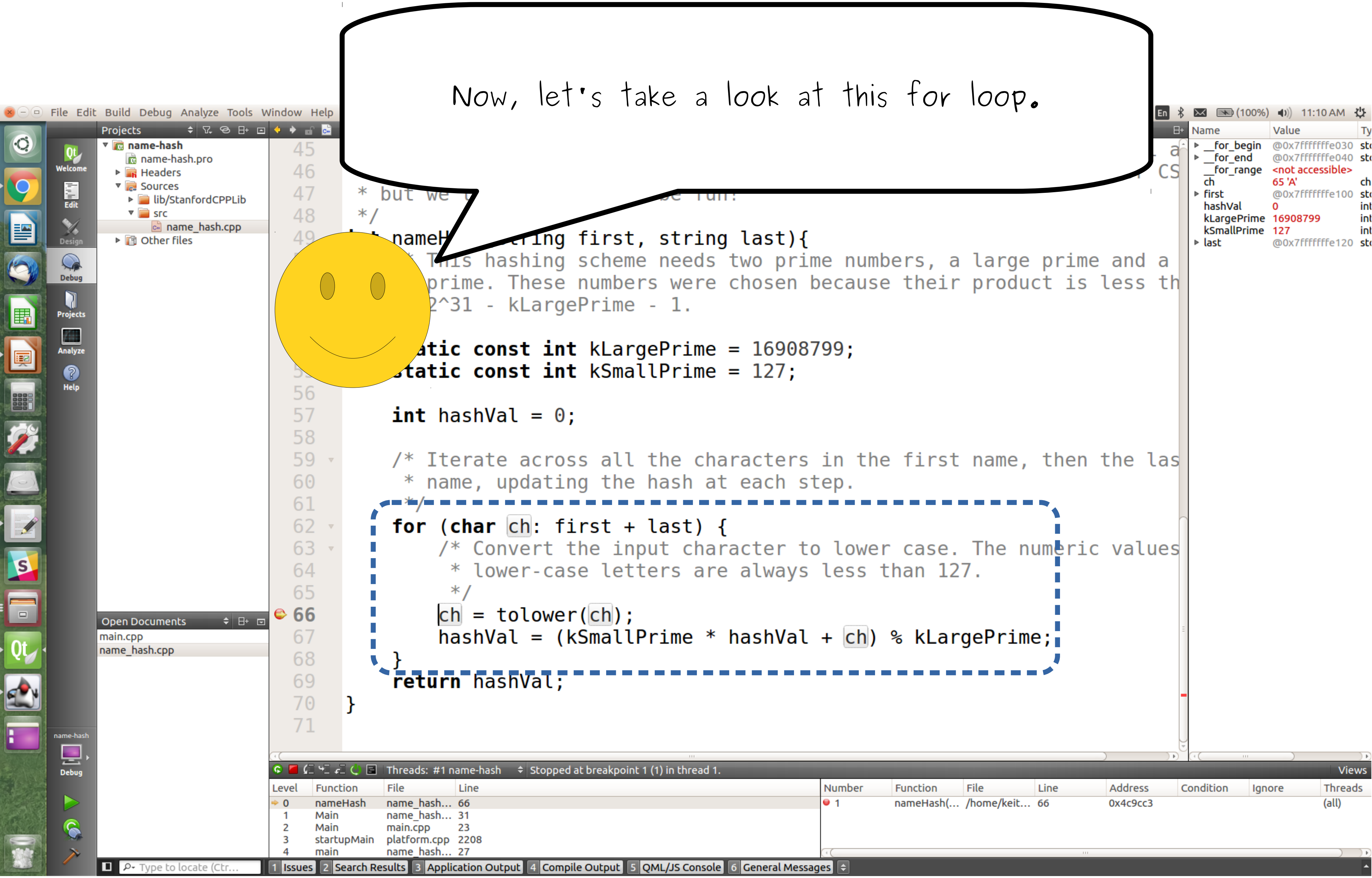
```
 * but we                       be run!
 */
      nameH    ring first, string last){
           is hashing scheme needs two prime numbers, a large prime and a
           prime. These numbers were chosen because their product is less th
           2^31 - kLargePrime - 1.

          tic const int kLargePrime = 16908799;
         static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Now, let's take a look at this for loop.

```
                                                                  a
                                                                 CS
 * but we c                       be run!
 */
int nameH      ring first, string last){
       is hashing scheme needs two prime numbers, a large prime and a
       prime. These numbers were chosen because their product is less th
       2^31 - kLargePrime - 1.

      atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

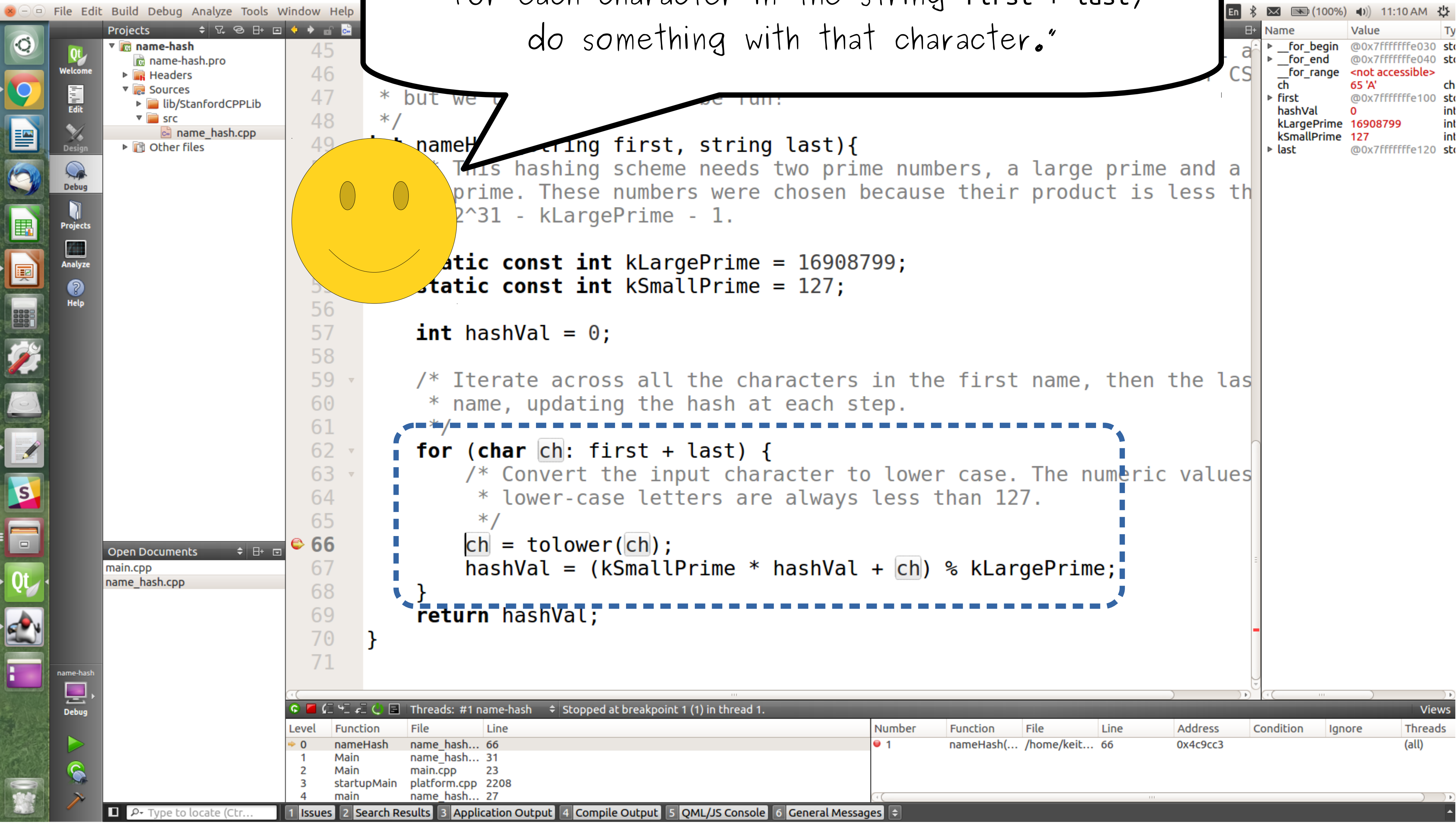
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

This loop is a range-based for loop. It says "for each character in the string **first + last**, do something with that character."



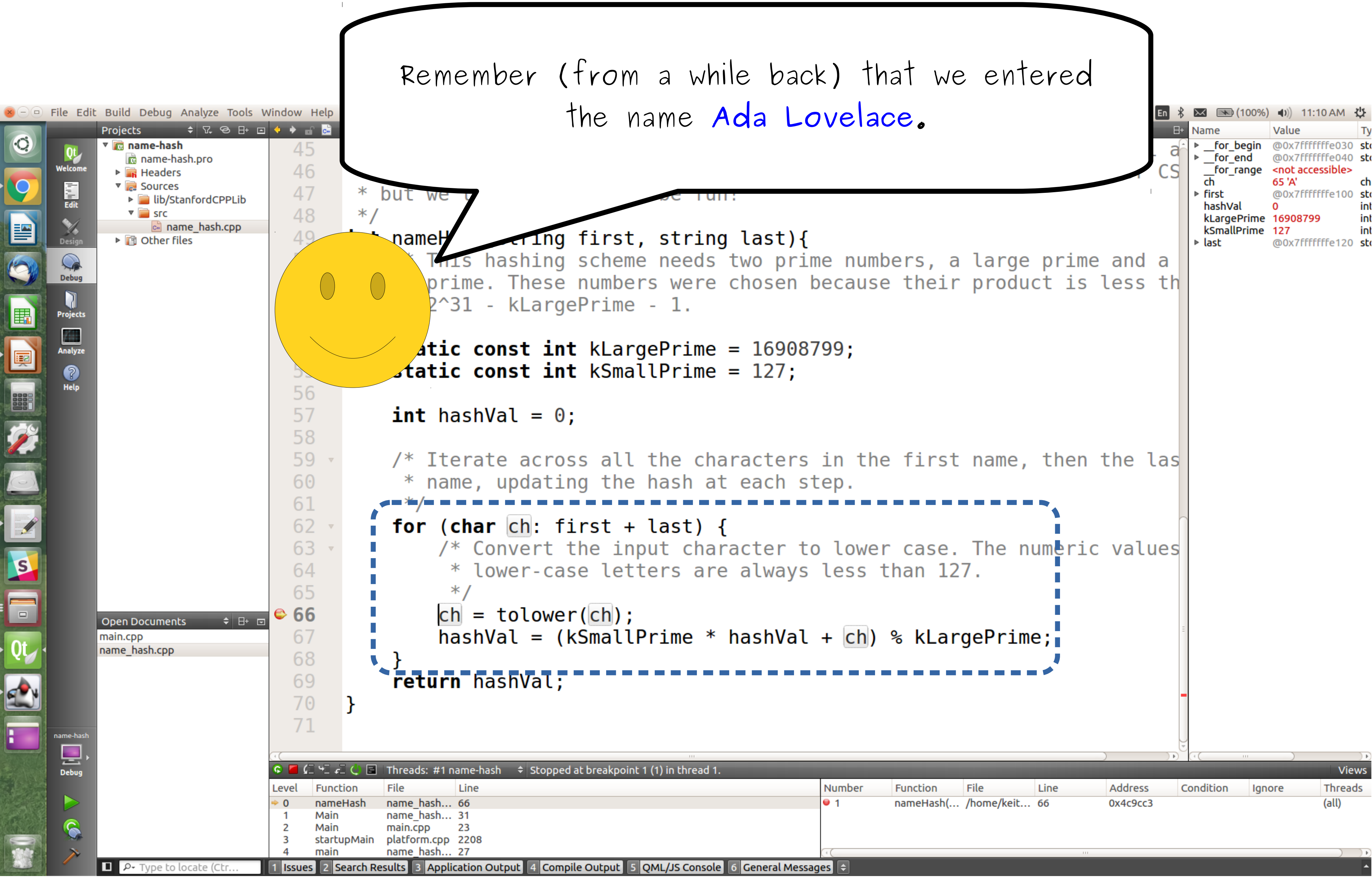
```
            a
            CS
 * but we       be run!
 */
    nameH    ring first, string last){
     * This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.

    atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Remember (from a while back) that we entered the name Ada Lovelace.

```
                                                                        a
                                                                       CS
 * but we                    be run.
 */
int nameH      tring first, string last){
      This hashing scheme needs two prime numbers, a large prime and a
     prime. These numbers were chosen because their product is less th
     2^31 - kLargePrime - 1.

     atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

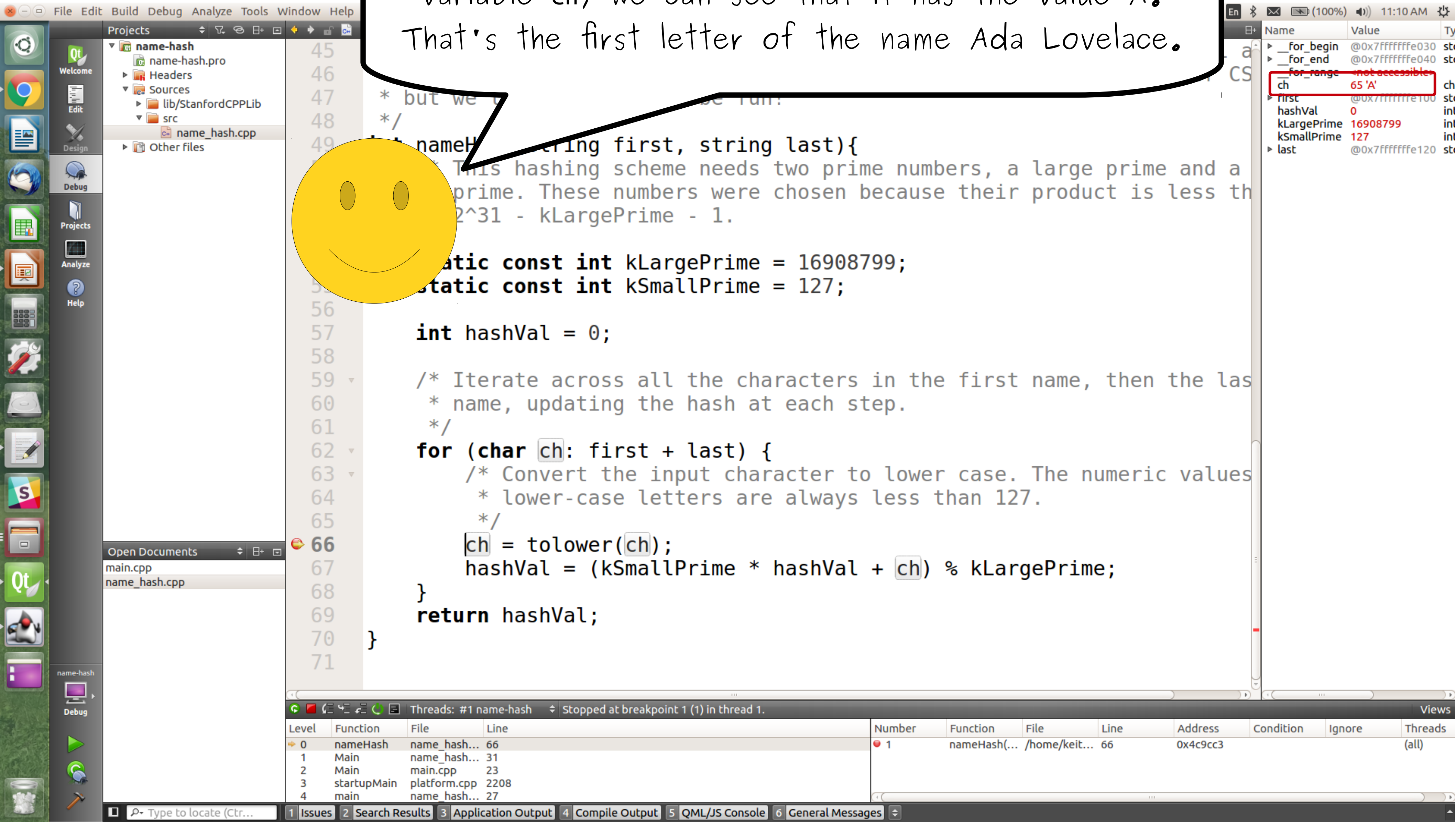
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

If we take a look at the current value of the variable **ch**, we can see that it has the value A. That's the first letter of the name Ada Lovelace.



```
 * but we c                    be run!
 */
    nameH      ring first, string last){
     * This hashing scheme needs two prime numbers, a large prime and a
     prime. These numbers were chosen because their product is less th
     2^31 - kLargePrime - 1.

     atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

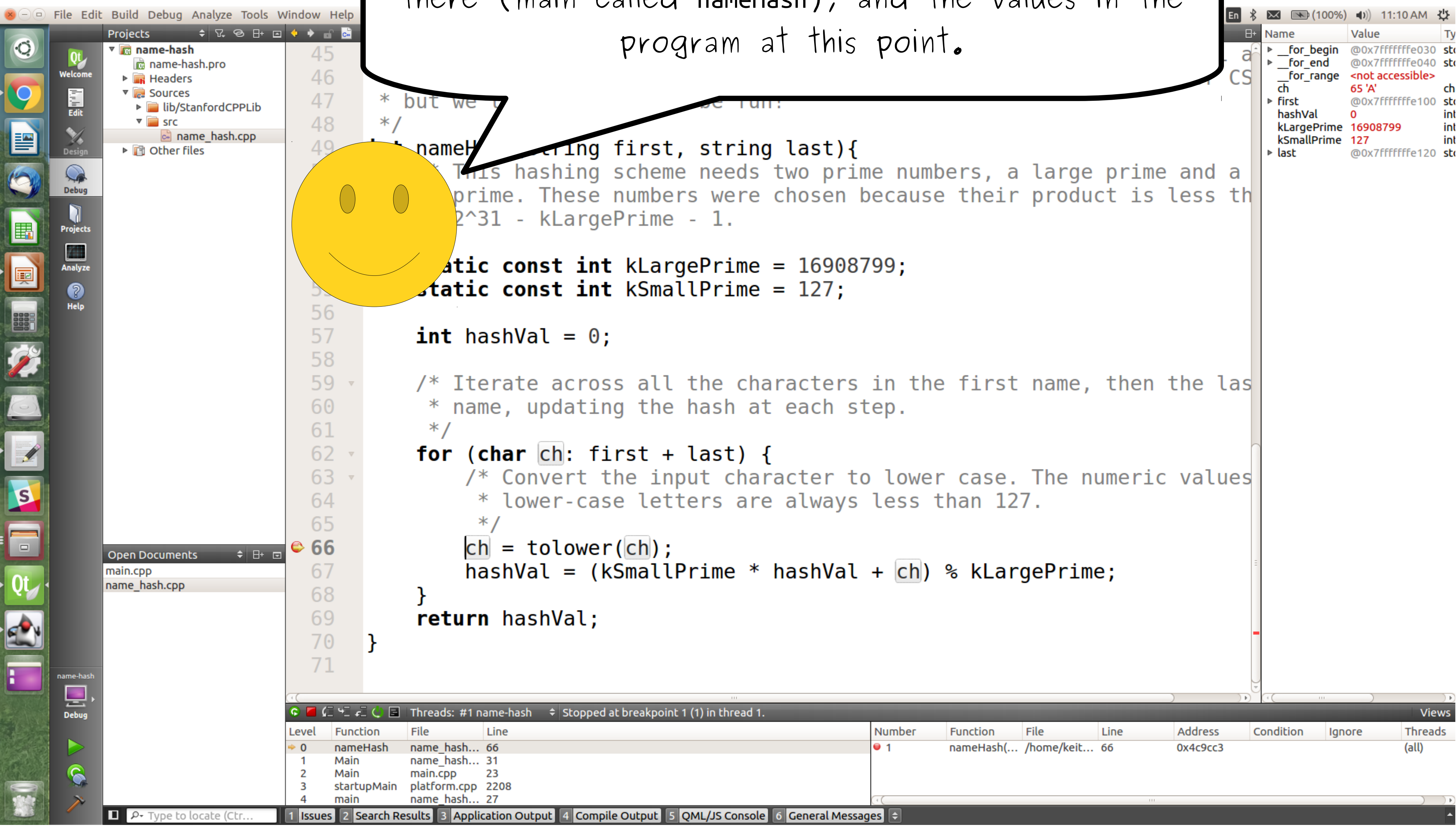
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

So now we know where we are (line 66), how we got there (main called **nameHash**), and the values in the program at this point.



```
45                                                                  a
46                                                                  CS
47  * but we                          be run!
48  */
49   t nameH          ring first, string last){
50          * This hashing scheme needs two prime numbers, a large prime and a
51          prime. These numbers were chosen because their product is less th
52          2^31 - kLargePrime - 1.
53
54        atic const int kLargePrime = 16908799;
55      static const int kSmallPrime = 127;
56
57      int hashVal = 0;
58
59      /* Iterate across all the characters in the first name, then the las
60       * name, updating the hash at each step.
61       */
62      for (char ch: first + last) {
63          /* Convert the input character to lower case. The numeric values
64           * lower-case letters are always less than 127.
65           */
66          ch = tolower(ch);
67          hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
68      }
69      return hashVal;
70  }
71
```

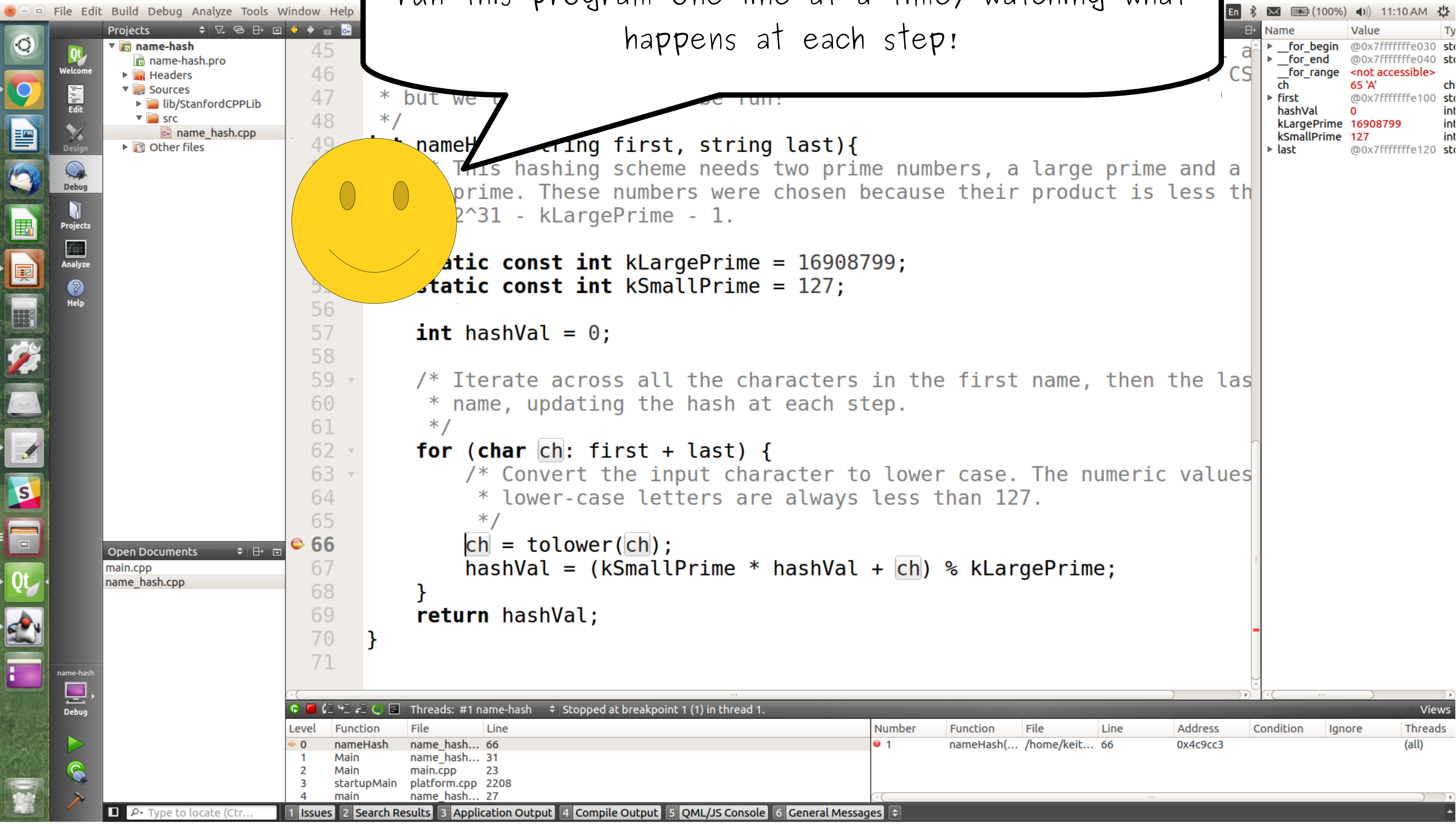
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Now, let's do something really cool – we're going to run this program one line at a time, watching what happens at each step!



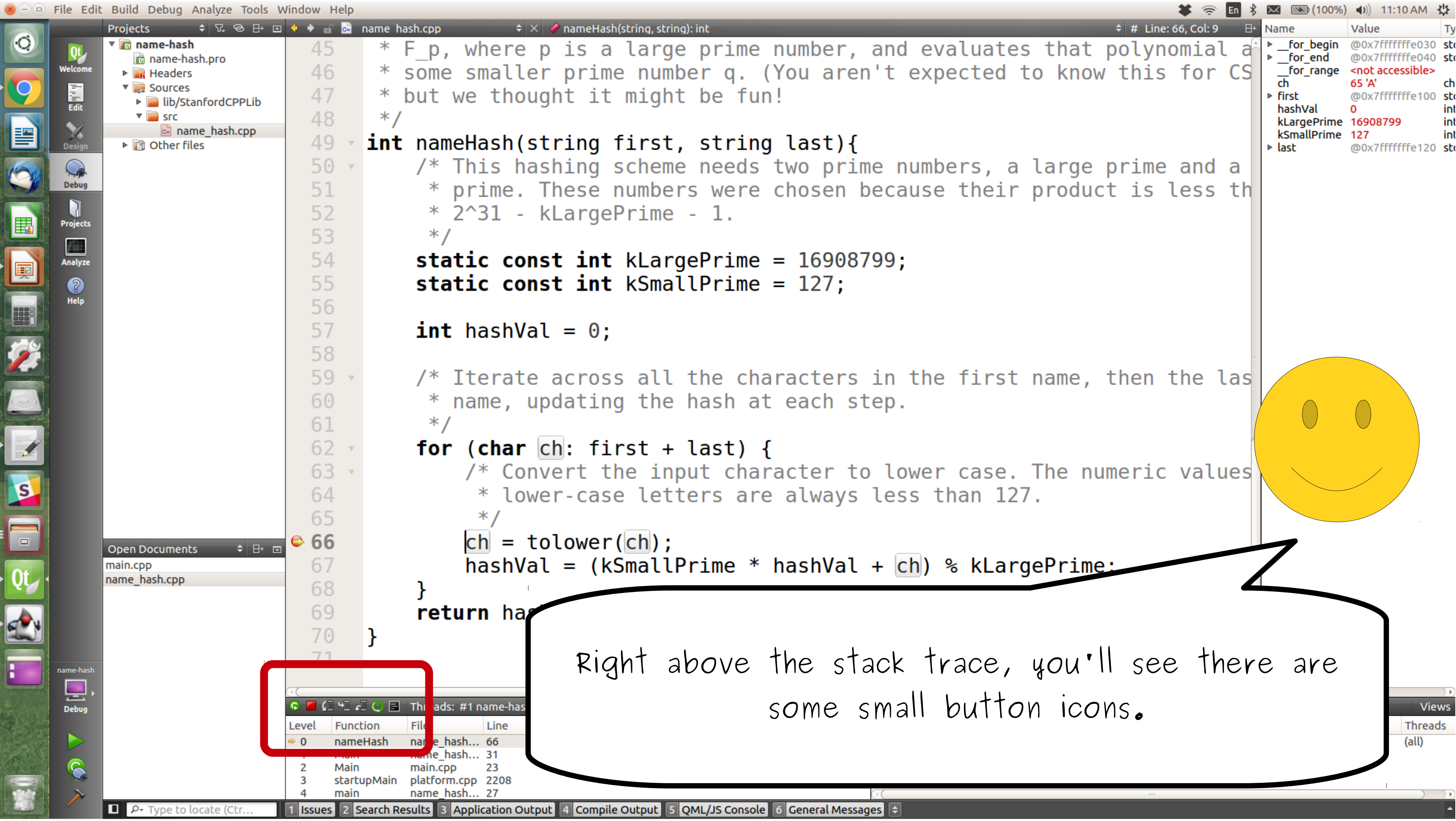
```
 * but we c                        be run.
 */
       nameH        ring first, string last){
     This hashing scheme needs two prime numbers, a large prime and a
     prime. These numbers were chosen because their product is less th
     2^31 - kLargePrime - 1.

    atic const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

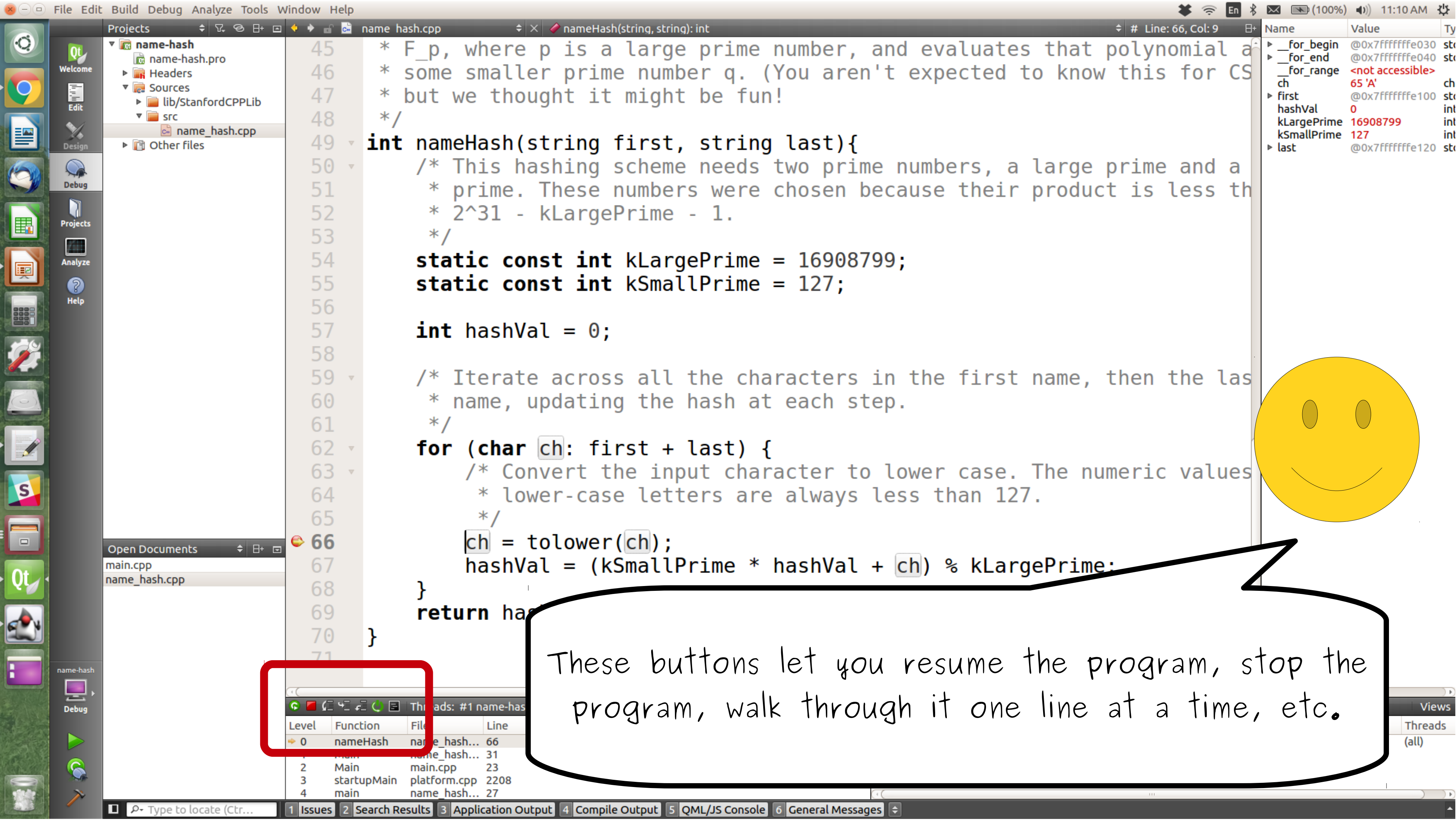


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Right above the stack trace, you'll see there are some small button icons.

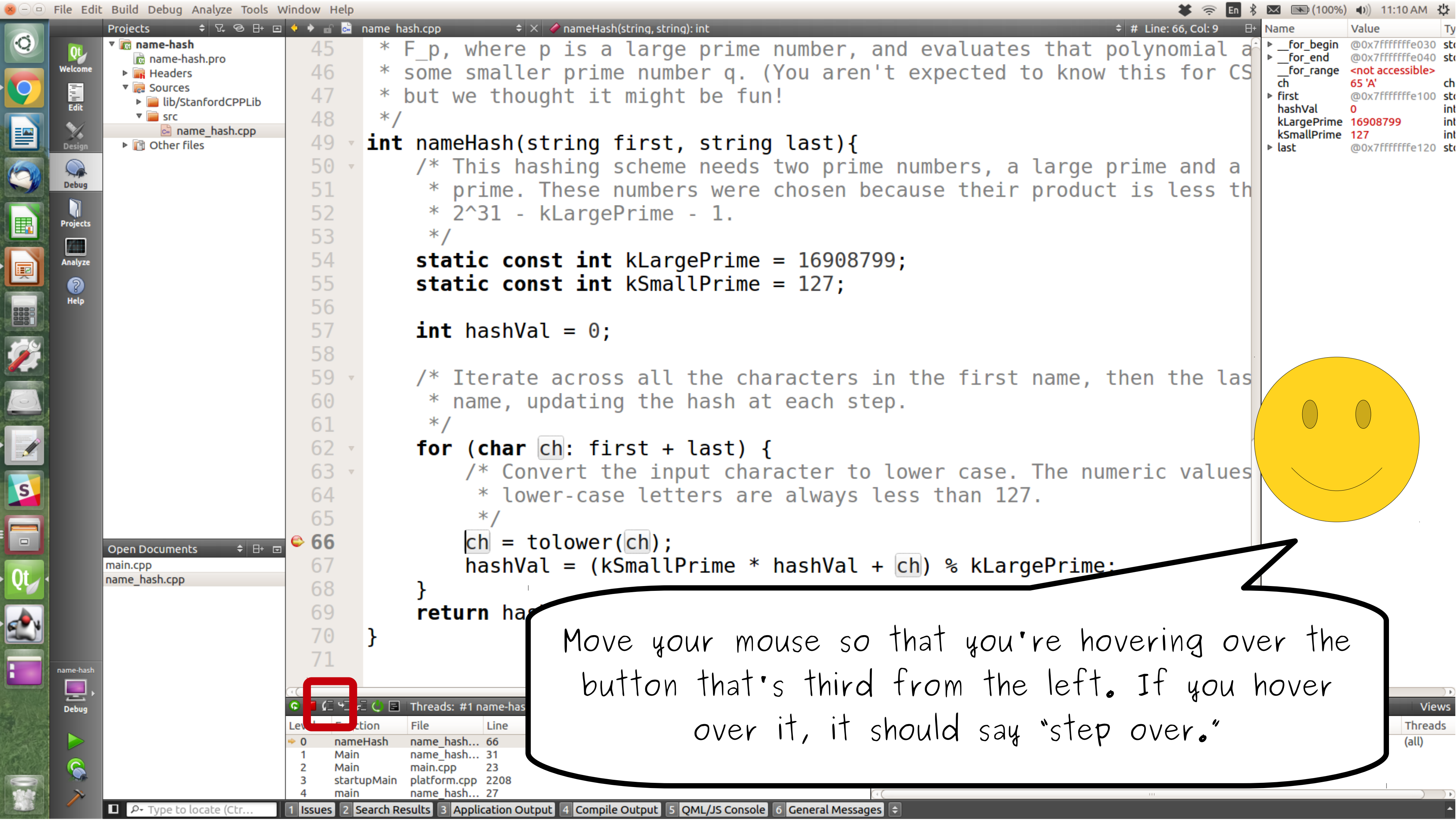


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	na_e_hash...	66
		_e_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

These buttons let you resume the program, stop the program, walk through it one line at a time, etc.

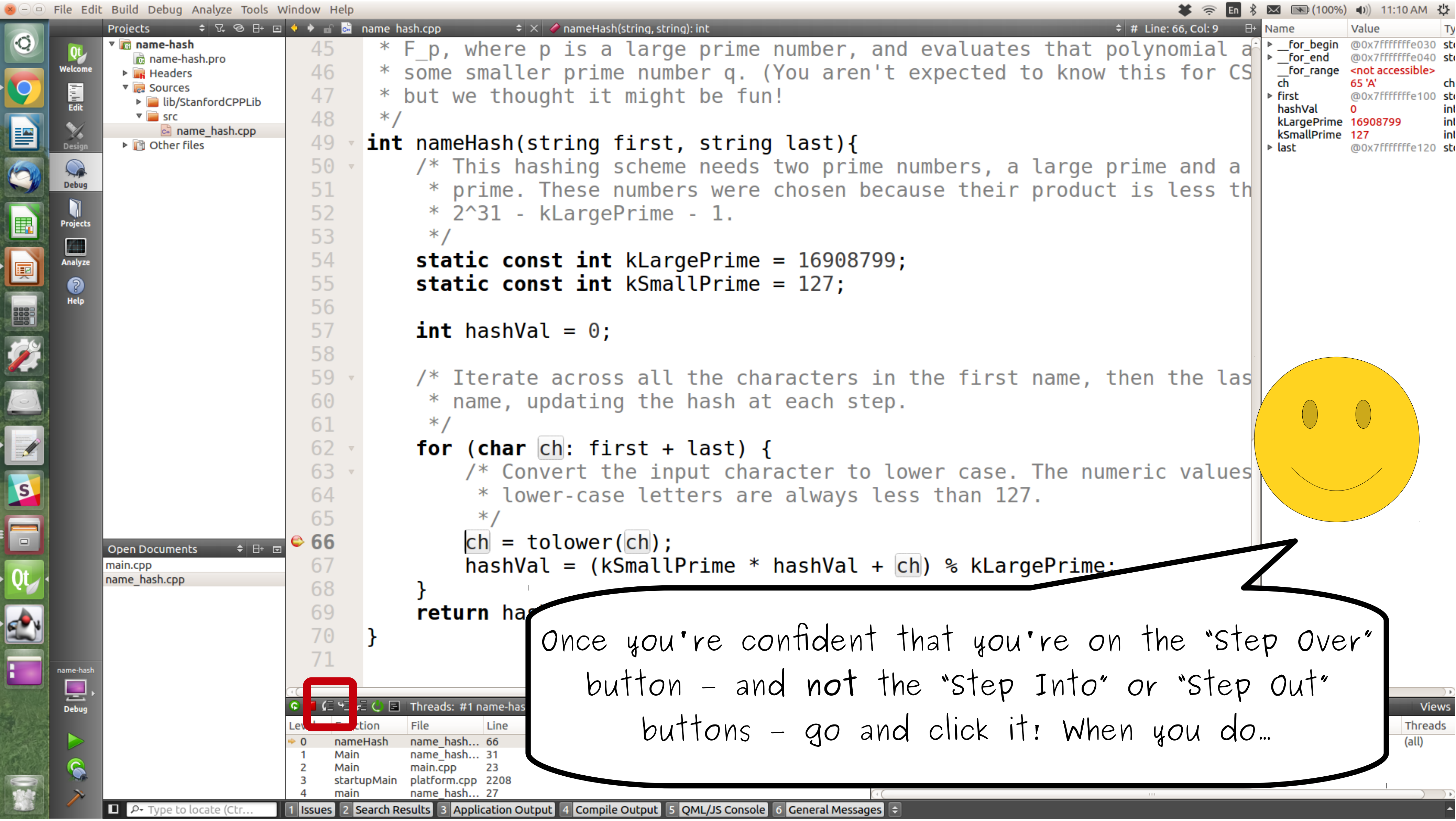


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

```
Name          Value                 Ty
__for_begin   @0x7fffffffe030       st
__for_end     @0x7fffffffe040       st
__for_range   <not accessible>
ch            65 'A'                ch
first         @0x7fffffffe100       st
hashVal       0                     int
kLargePrime   16908799              int
kSmallPrime   127                   int
last          @0x7fffffffe120       st
```

```
Lev  Function     File          Line
0    nameHash     name_hash...  66
1    Main         name_hash...  31
2    Main         main.cpp      23
3    startupMain  platform.cpp  2208
4    main         name_hash...  27
```

Move your mouse so that you're hovering over the button that's third from the left. If you hover over it, it should say "step over."

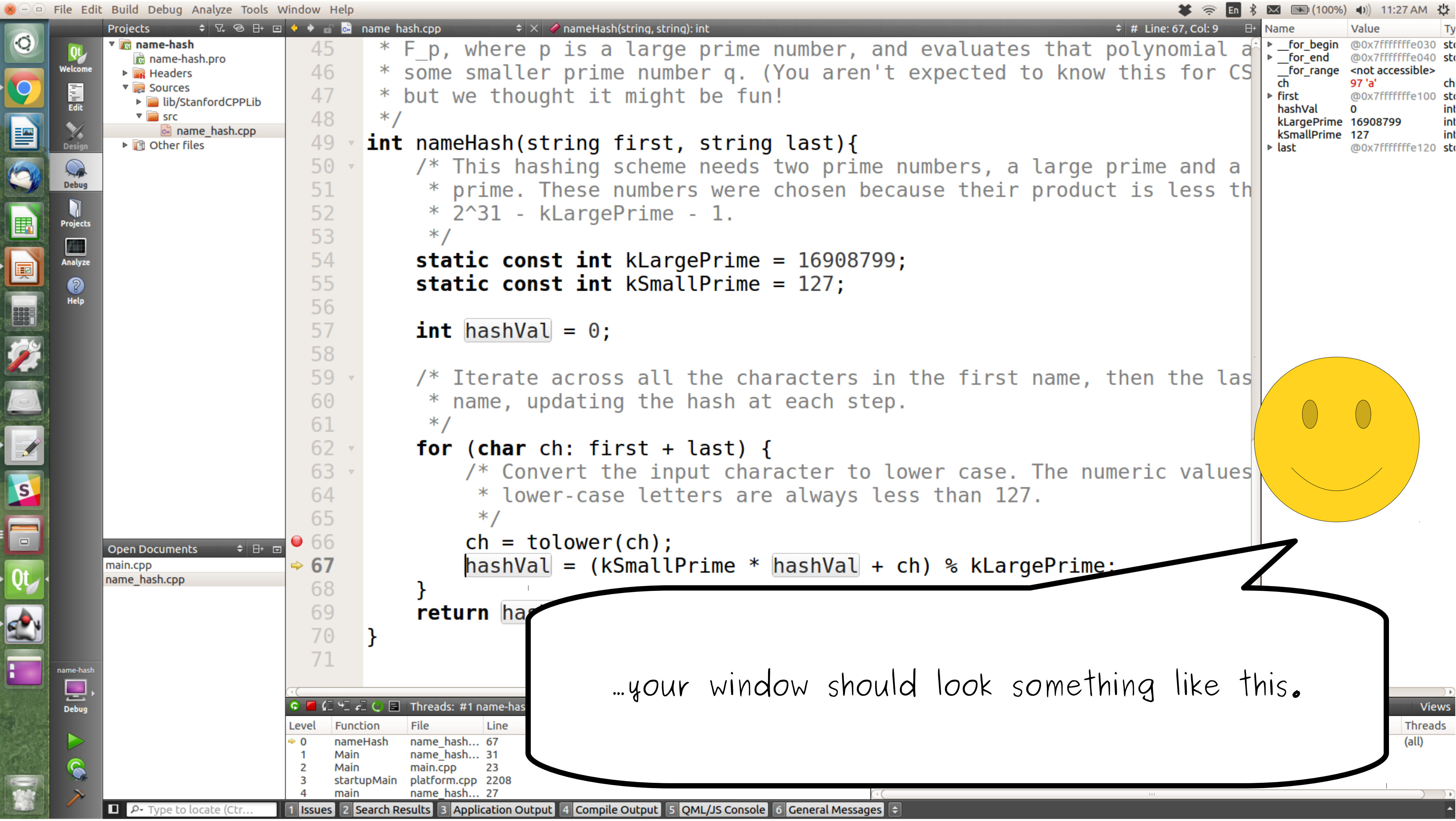


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	65 'A'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Once you're confident that you're on the "Step Over" button – and **not** the "Step Into" or "Step Out" buttons – go and click it! When you do…

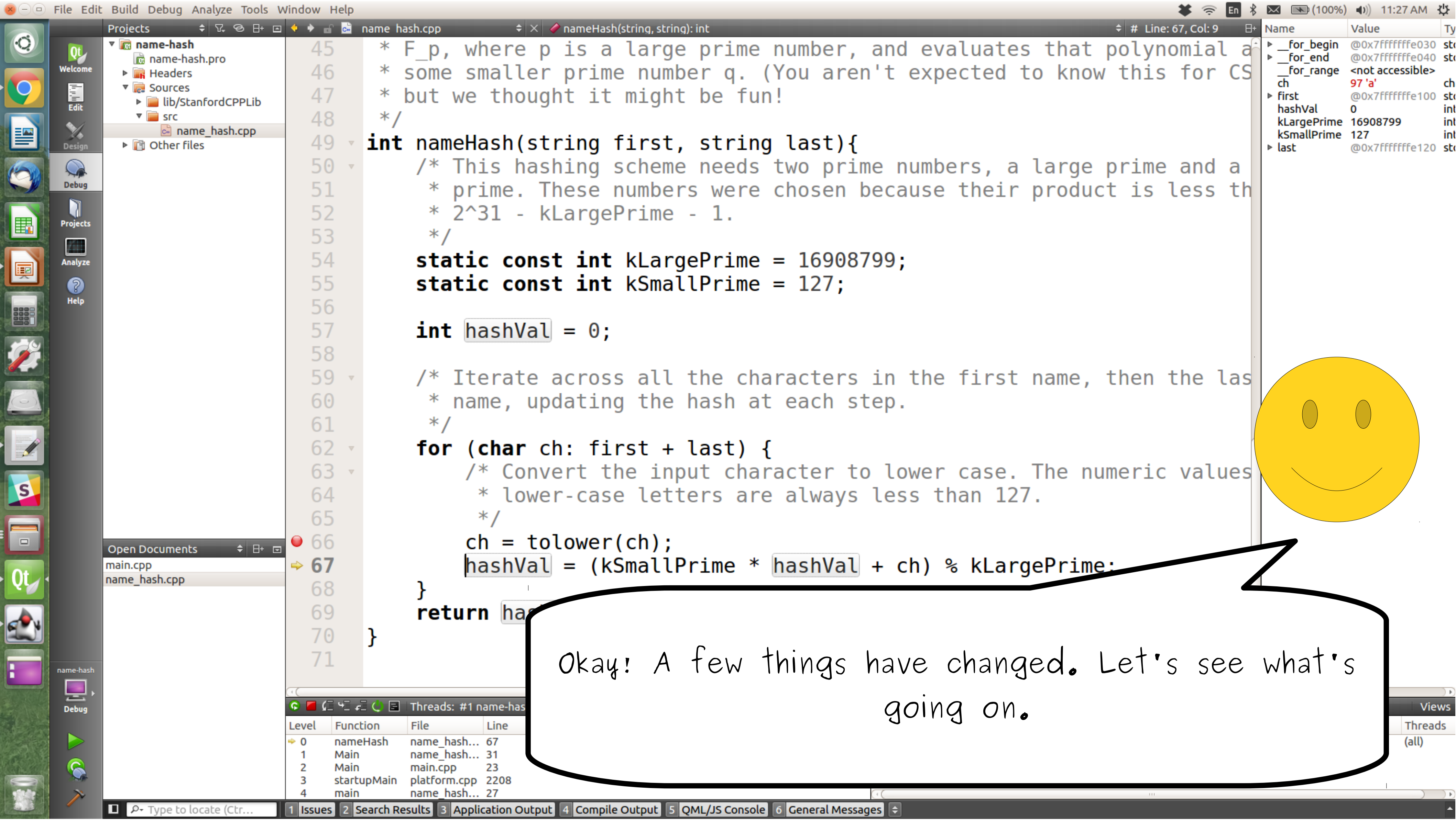


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

...your window should look something like this.

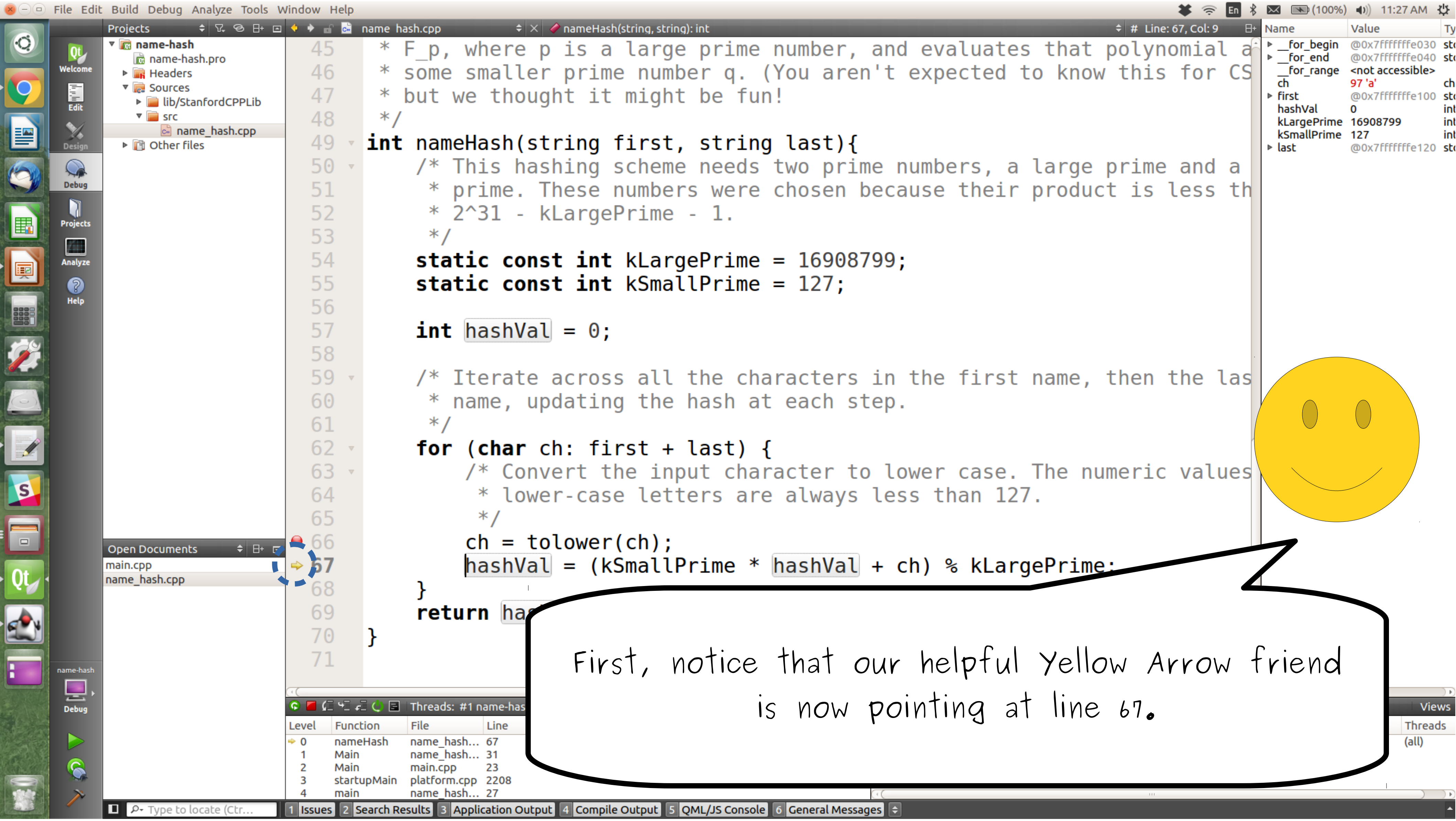


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
▸ __for_begin	@0x7fffffffe030	st
▸ __for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
▸ first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
▸ last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Okay! A few things have changed. Let's see what's going on.

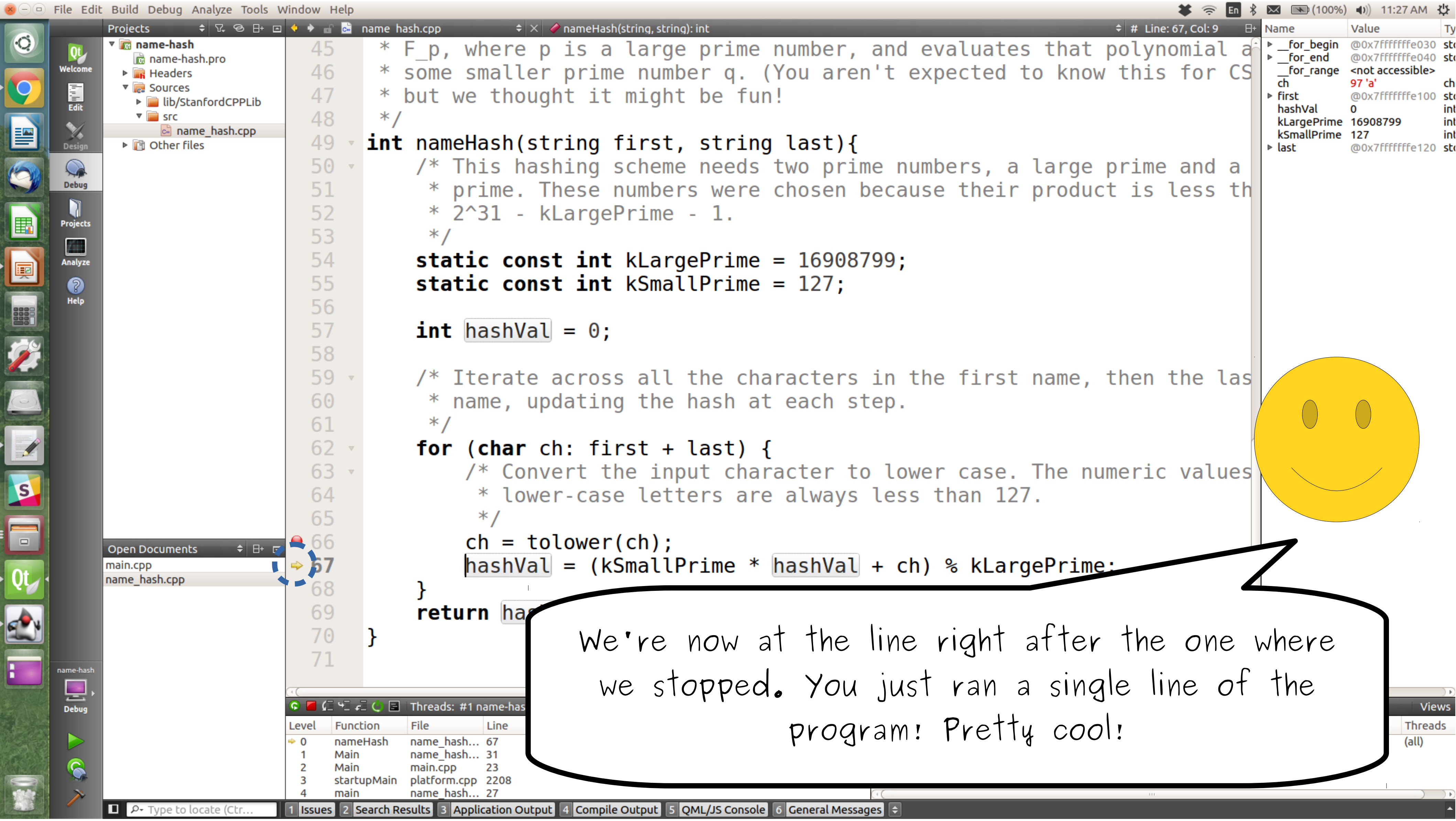


```
45  * F_p, where p is a large prime number, and evaluates that polynomial a
46  * some smaller prime number q. (You aren't expected to know this for CS
47  * but we thought it might be fun!
48  */
49 int nameHash(string first, string last){
50     /* This hashing scheme needs two prime numbers, a large prime and a
51      * prime. These numbers were chosen because their product is less th
52      * 2^31 - kLargePrime - 1.
53      */
54     static const int kLargePrime = 16908799;
55     static const int kSmallPrime = 127;
56
57     int hashVal = 0;
58
59     /* Iterate across all the characters in the first name, then the las
60      * name, updating the hash at each step.
61      */
62     for (char ch: first + last) {
63         /* Convert the input character to lower case. The numeric values
64          * lower-case letters are always less than 127.
65          */
66         ch = tolower(ch);
67         hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
68     }
69     return ha
70 }
71
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

First, notice that our helpful Yellow Arrow friend is now pointing at line 67.

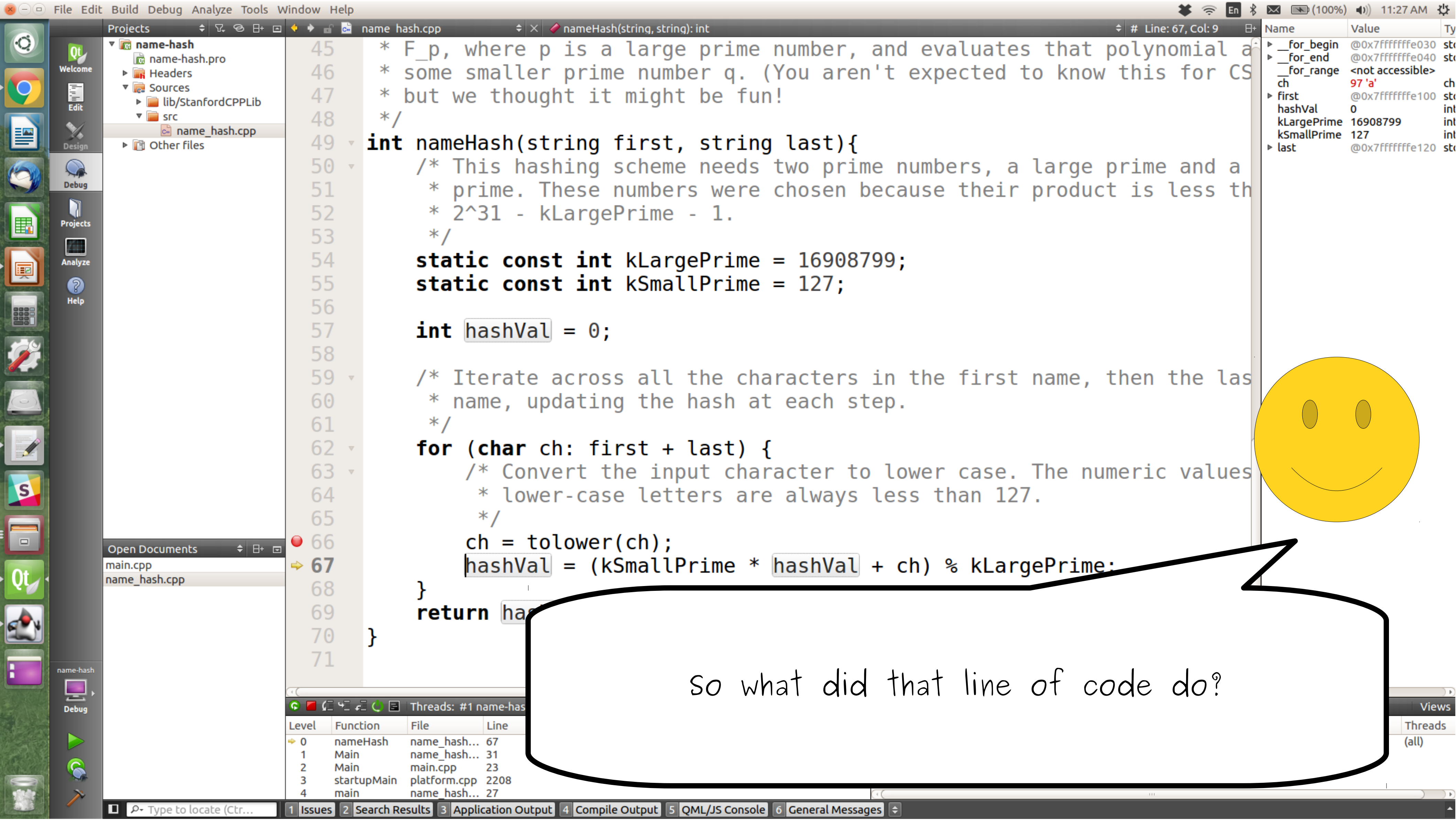


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

```
Name          Value              Ty
__for_begin   @0x7fffffffe030    st
__for_end     @0x7fffffffe040    st
__for_range   <not accessible>
ch            97 'a'             ch
first         @0x7fffffffe100    st
hashVal       0                  in
kLargePrime   16908799           in
kSmallPrime   127                in
last          @0x7fffffffe120    st
```

```
Level  Function     File          Line
0      nameHash     name_hash...  67
1      Main         name_hash...  31
2      Main         main.cpp      23
3      startupMain  platform.cpp  2208
4      main         name_hash...  27
```

We're now at the line right after the one where we stopped. You just ran a single line of the program! Pretty cool!

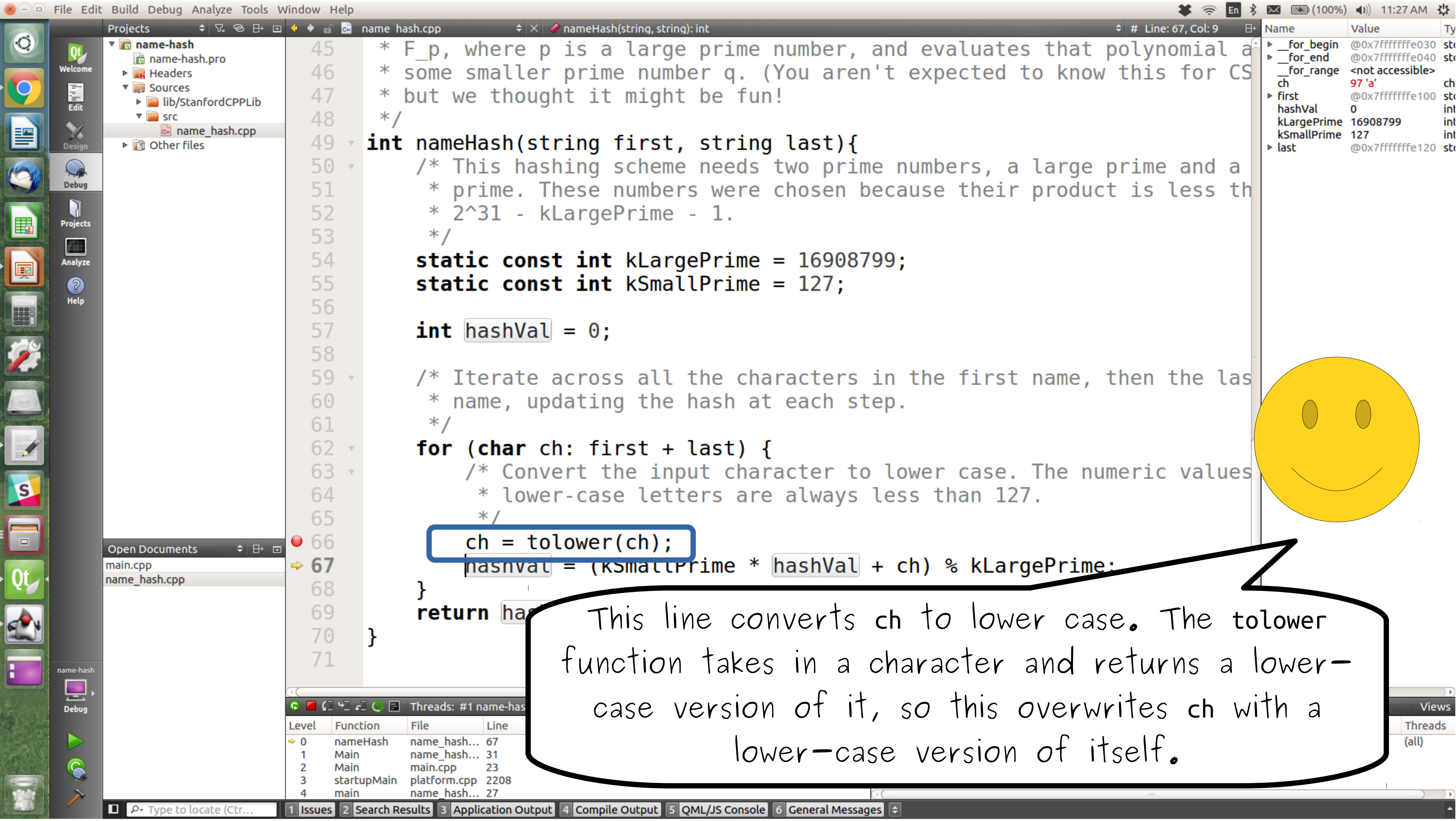


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

So what did that line of code do?

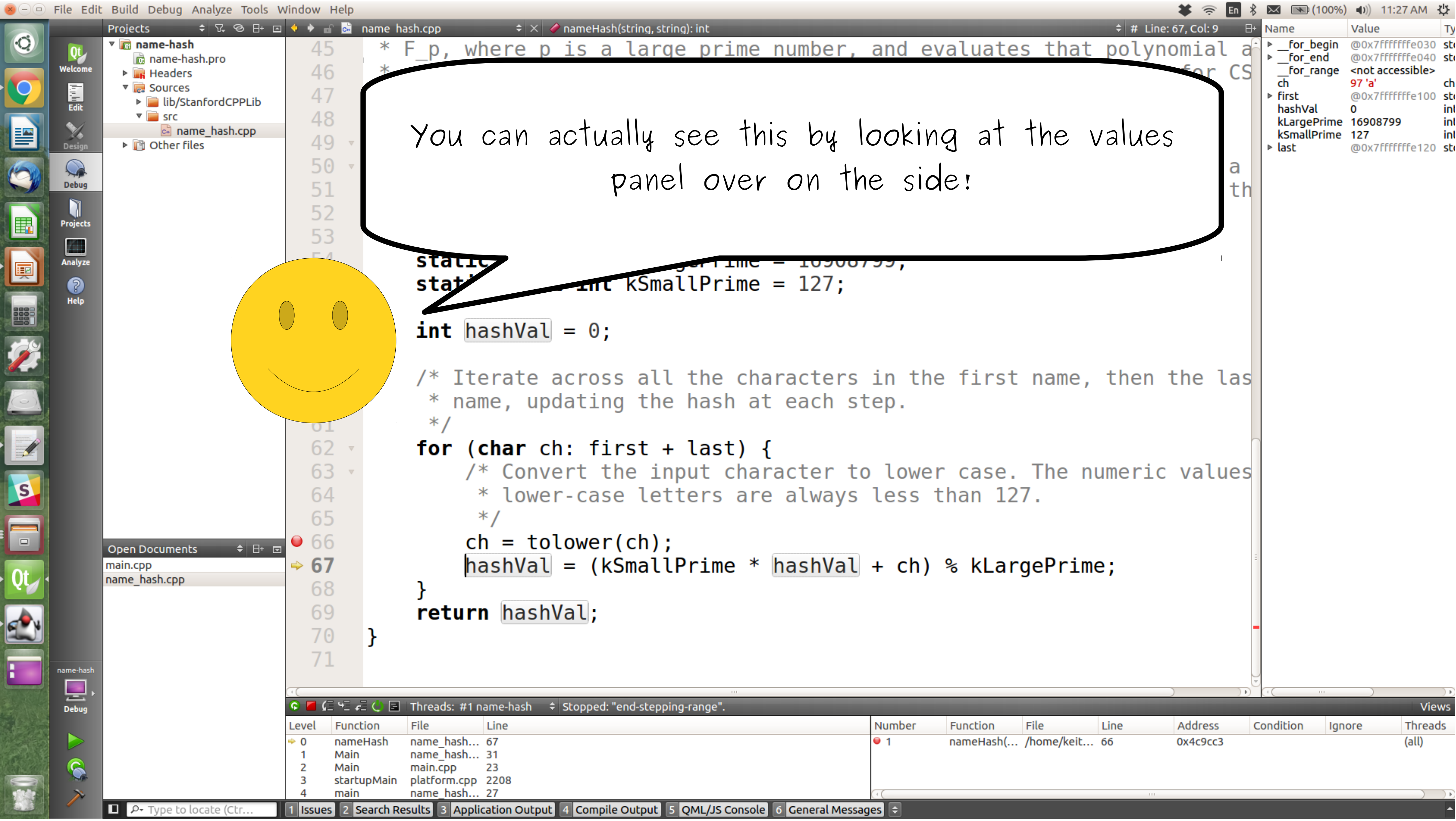


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return ha
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

This line converts **ch** to lower case. The **tolower** function takes in a character and returns a lower-case version of it, so this overwrites **ch** with a lower-case version of itself.



You can actually see this by looking at the values panel over on the side!

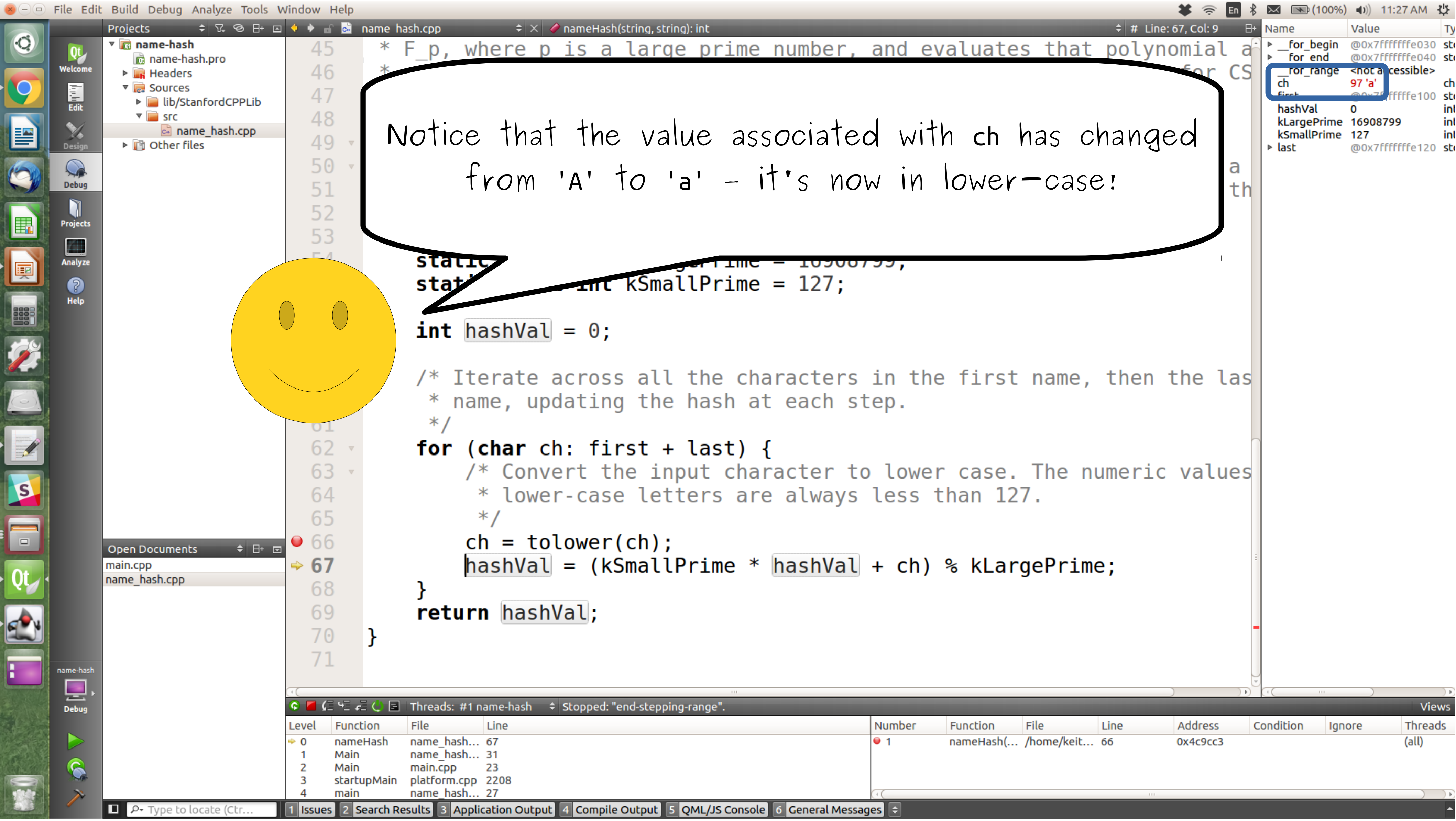
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                               for CS
                                                                      a
                                                                      th
    static                 Prime = 16908799;
    stat         int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
▸ __for_begin	@0x7fffffffe030	st
▸ __for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
▸ first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
▸ last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Notice that the value associated with **ch** has changed from 'A' to 'a' – it's now in lower-case!

name_hash.cpp — nameHash(string, string): int — Line: 67, Col: 9

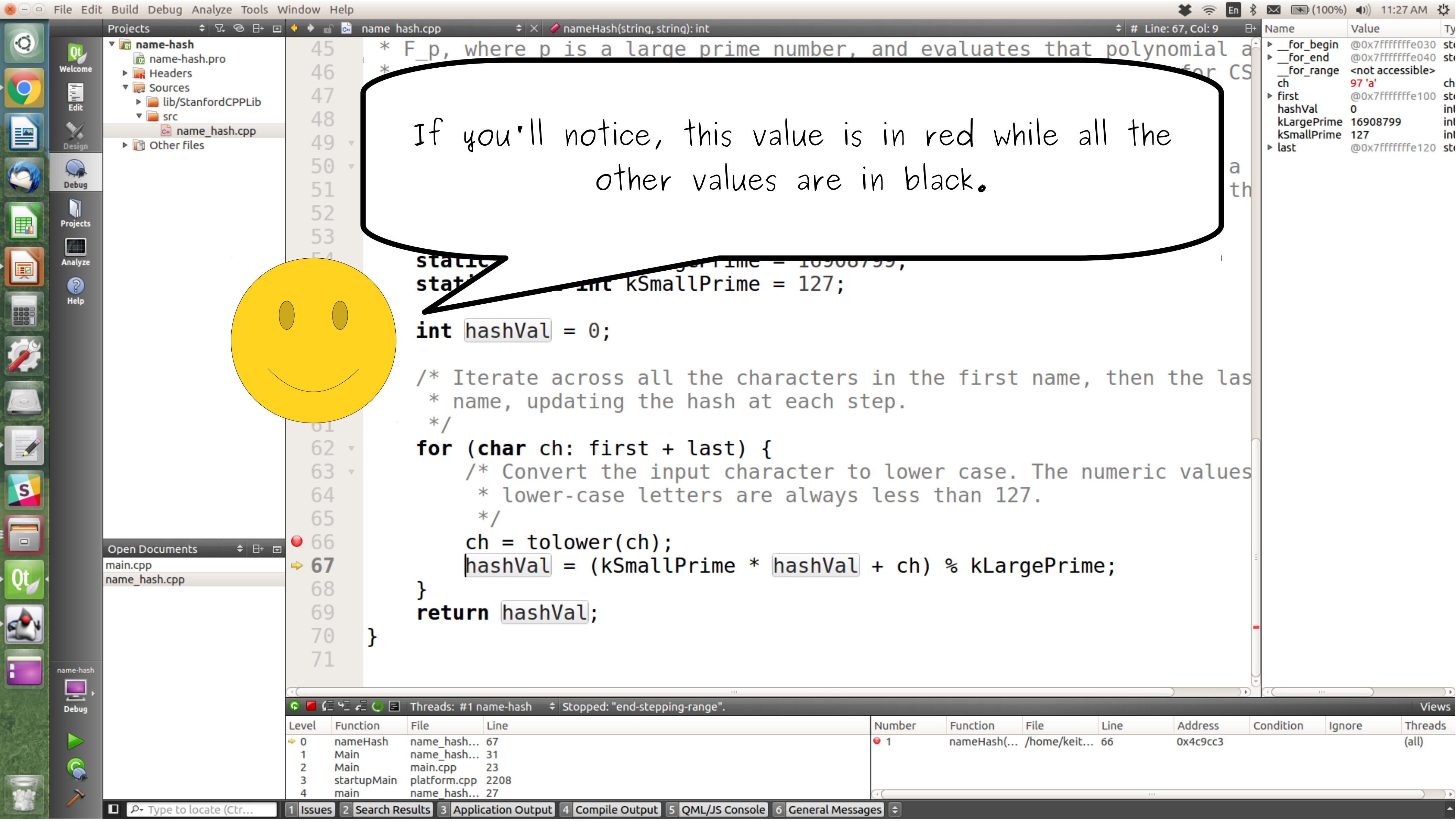
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                                for CS
                                                                       a
                                                                       th

    static                        rime = 16908799;
    stat          int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash — Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



If you'll notice, this value is in red while all the other values are in black.

Projects
- name-hash
  - name-hash.pro
  - Headers
  - Sources
    - lib/StanfordCPPLib
    - src
      - name_hash.cpp
  - Other Files

Open Documents
- main.cpp
- name_hash.cpp

name_hash.cpp — nameHash(string, string): int — Line: 67, Col: 9

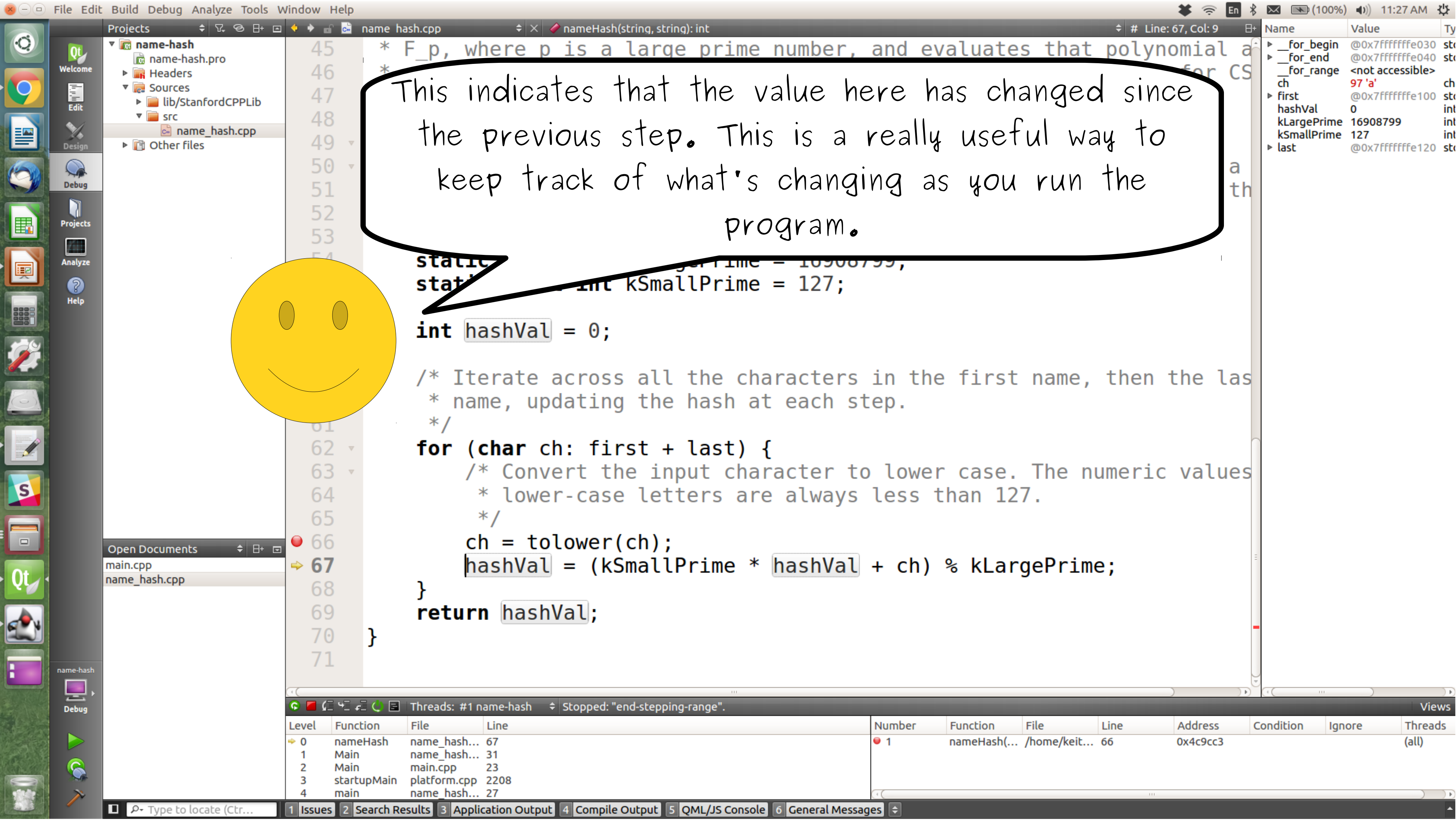
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                                 for CS
                                                                         a
                                                                         th
    static                  Prime = 16908799;
    stat         int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



This indicates that the value here has changed since the previous step. This is a really useful way to keep track of what's changing as you run the program.

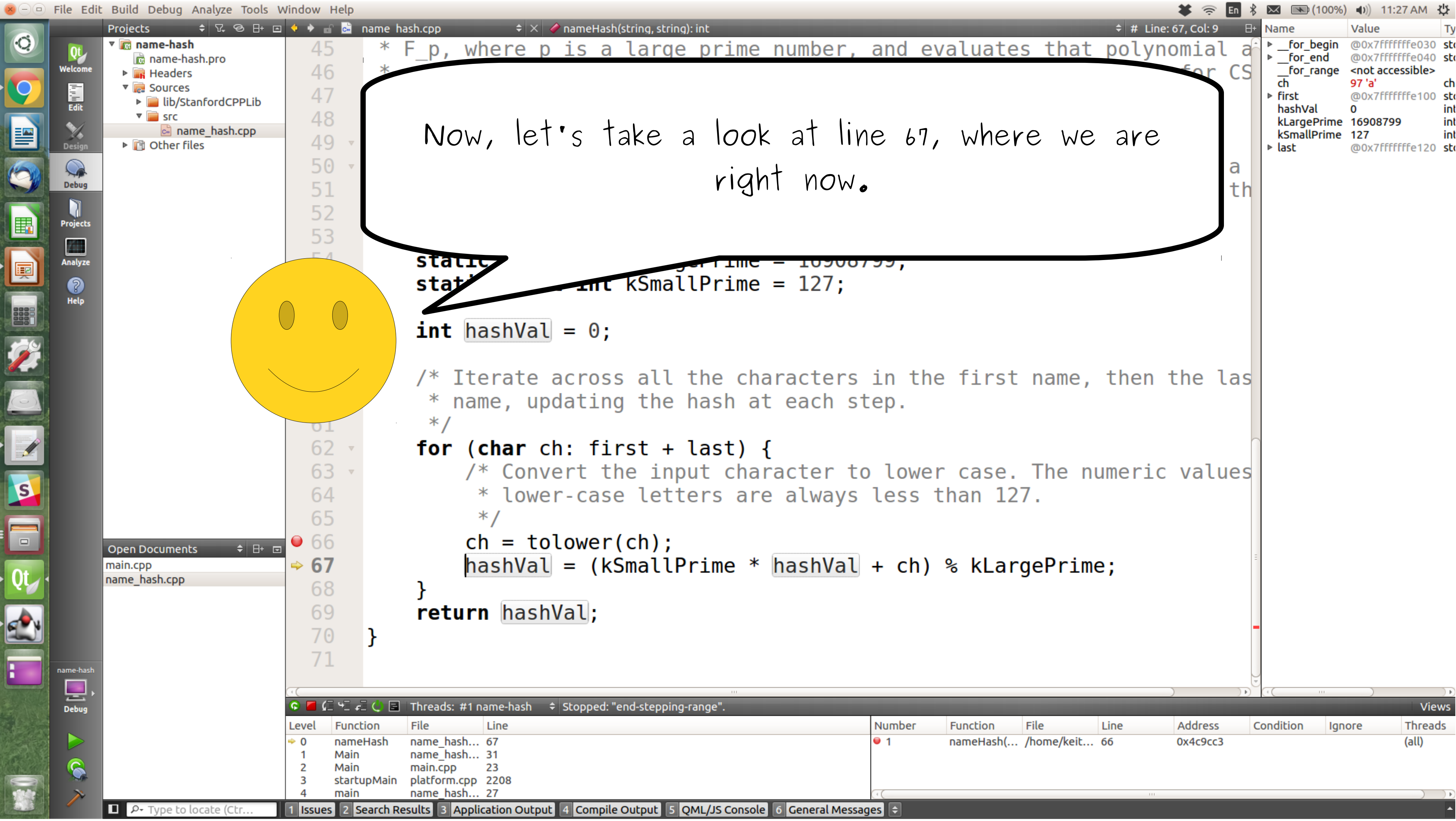
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                           for CS
                                                                 a
                                                                 th
    static                 Prime = 16908799;
    stat       int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
▸ __for_begin	@0x7fffffffe030	st
▸ __for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
▸ first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
▸ last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Now, let's take a look at line 67, where we are right now.

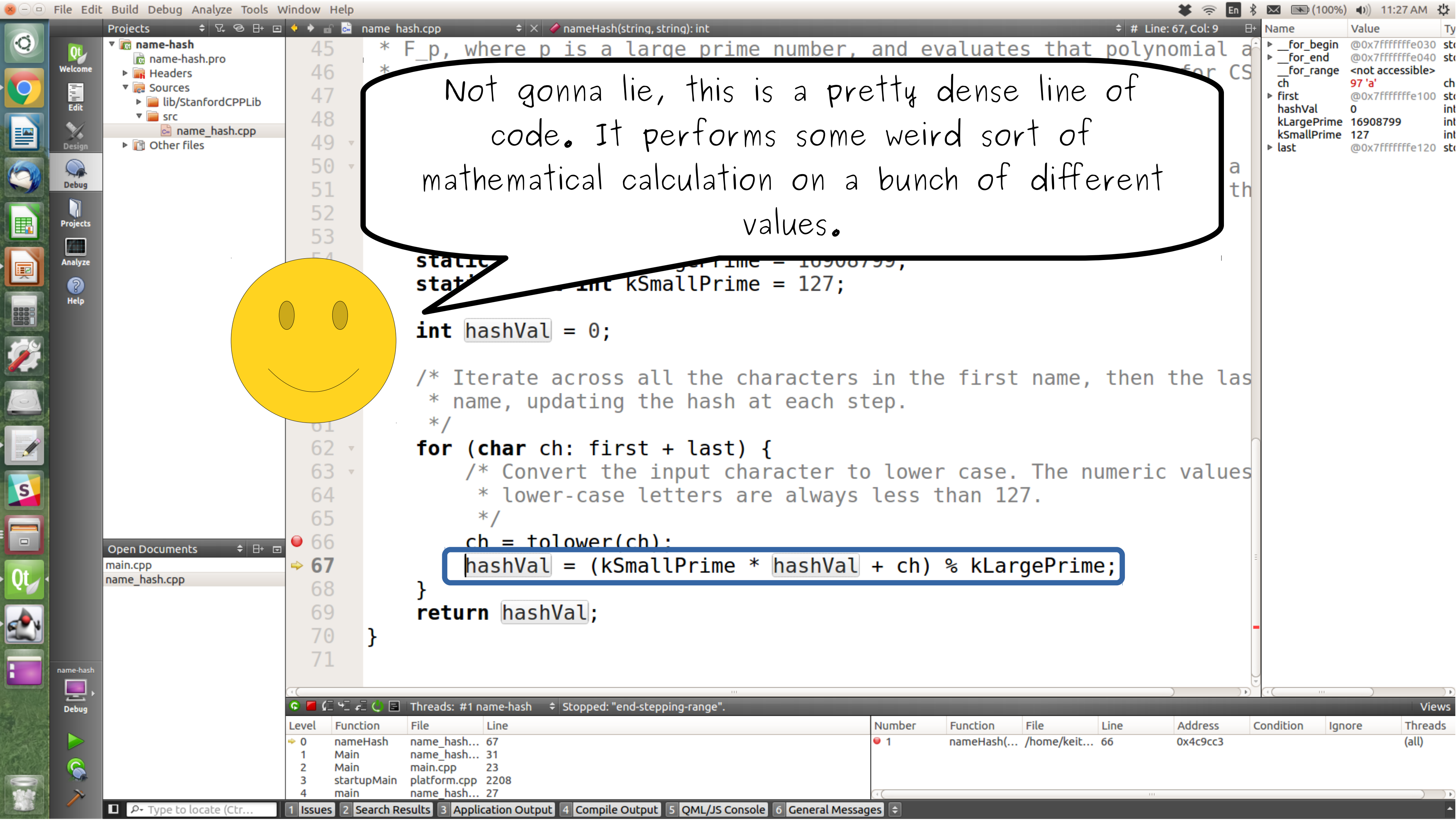
```
45  * F_p, where p is a large prime number, and evaluates that polynomial a
46  *                                                              for CS
47
48
49
50                                                                 a
51                                                                 th
52
53
    static                    rime = 16908799;
    stat          int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
62  for (char ch: first + last) {
63      /* Convert the input character to lower case. The numeric values
64       * lower-case letters are always less than 127.
65       */
66      ch = tolower(ch);
67      hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
68  }
69  return hashVal;
70 }
71
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



Not gonna lie, this is a pretty dense line of code. It performs some weird sort of mathematical calculation on a bunch of different values.

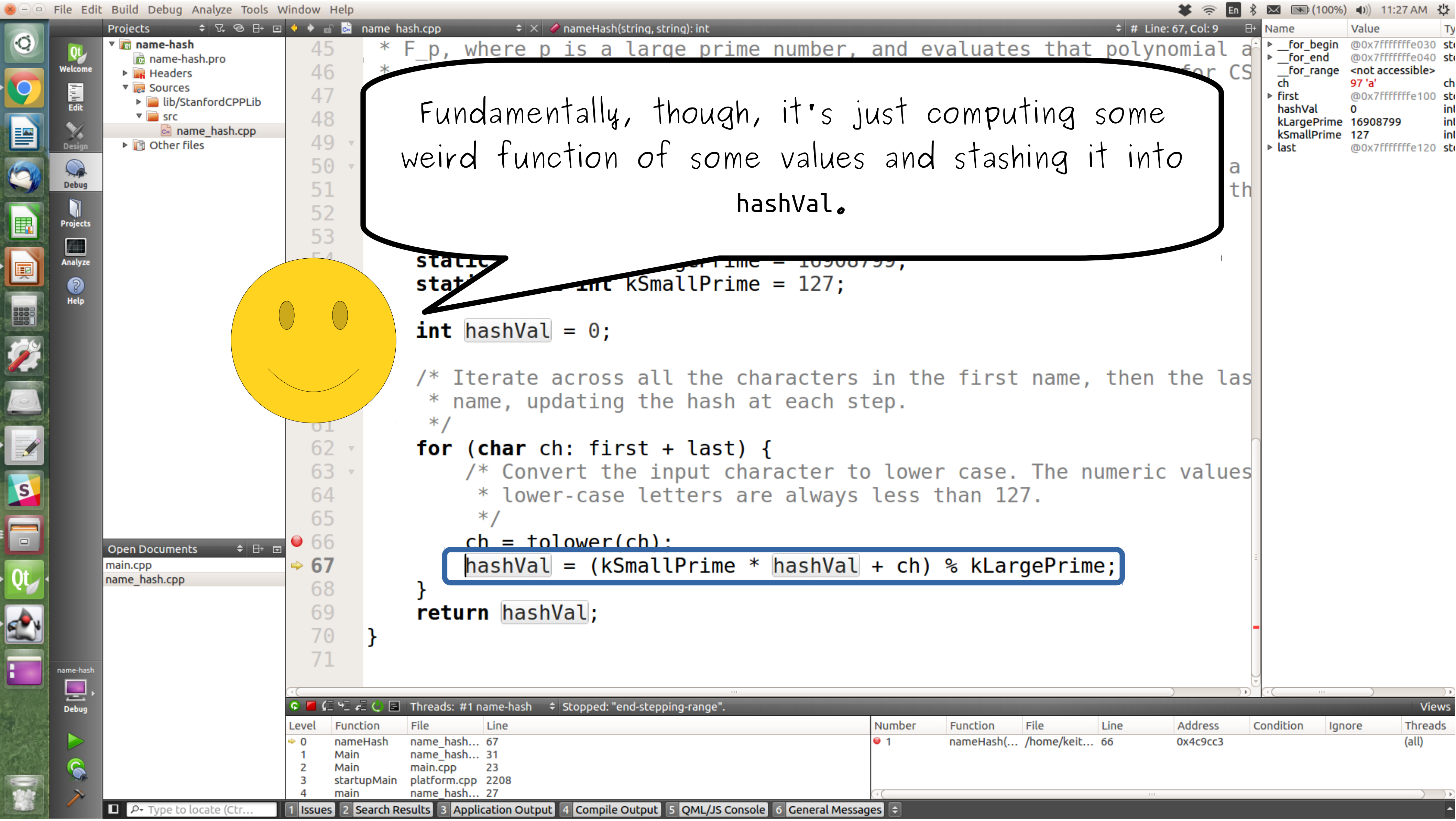
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                          for CS
                                                            a
                                                            th
    static                      gePrime = 16908799;
    stat          int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash…	67
1	Main	name_hash…	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash…	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(…	/home/keit…	66	0x4c9cc3			(all)



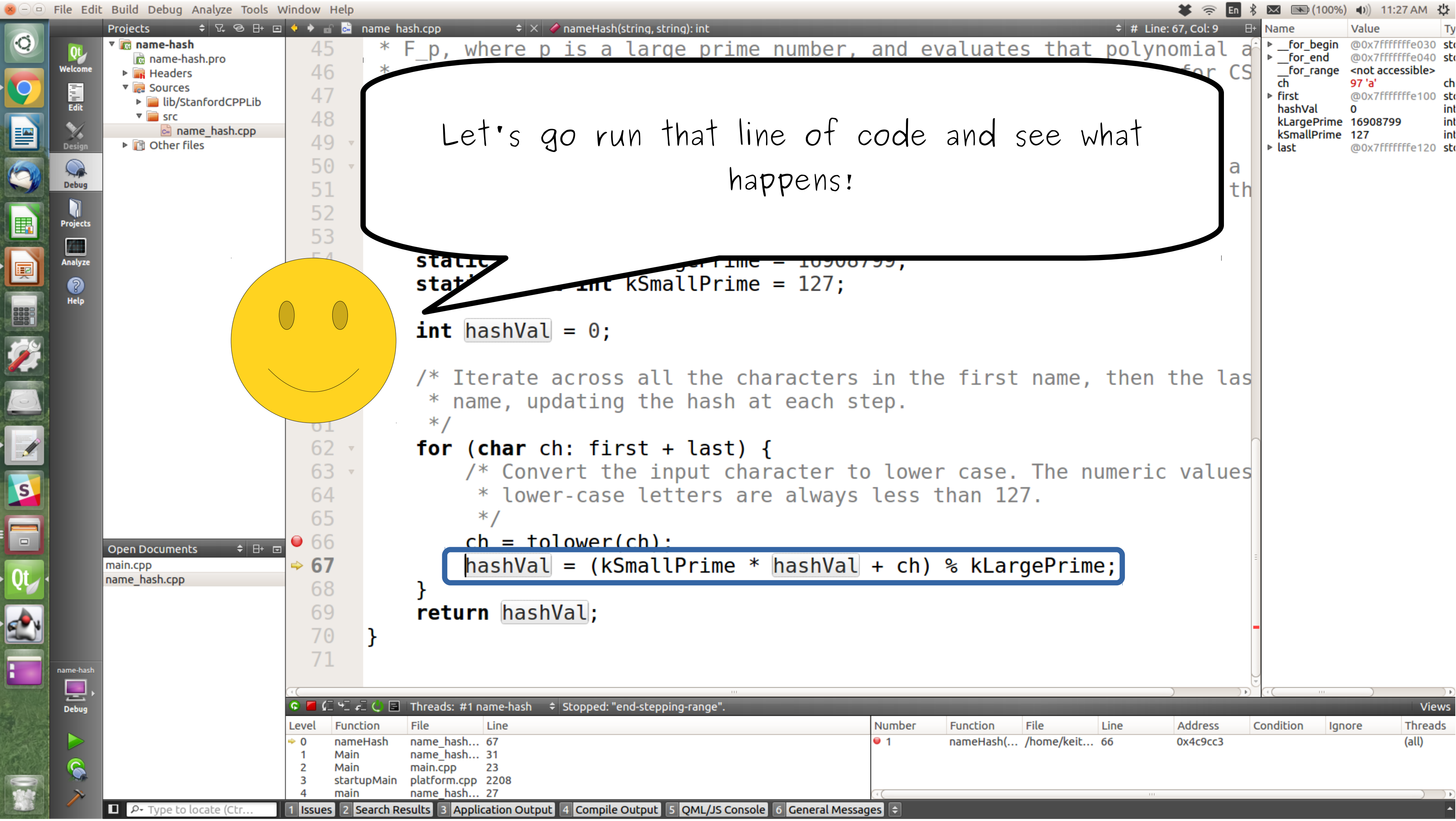
Fundamentally, though, it's just computing some weird function of some values and stashing it into hashVal.

```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                          for CS
                                                                  a
                                                                  th
    static               gePrime = 16908799;
    stat          int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



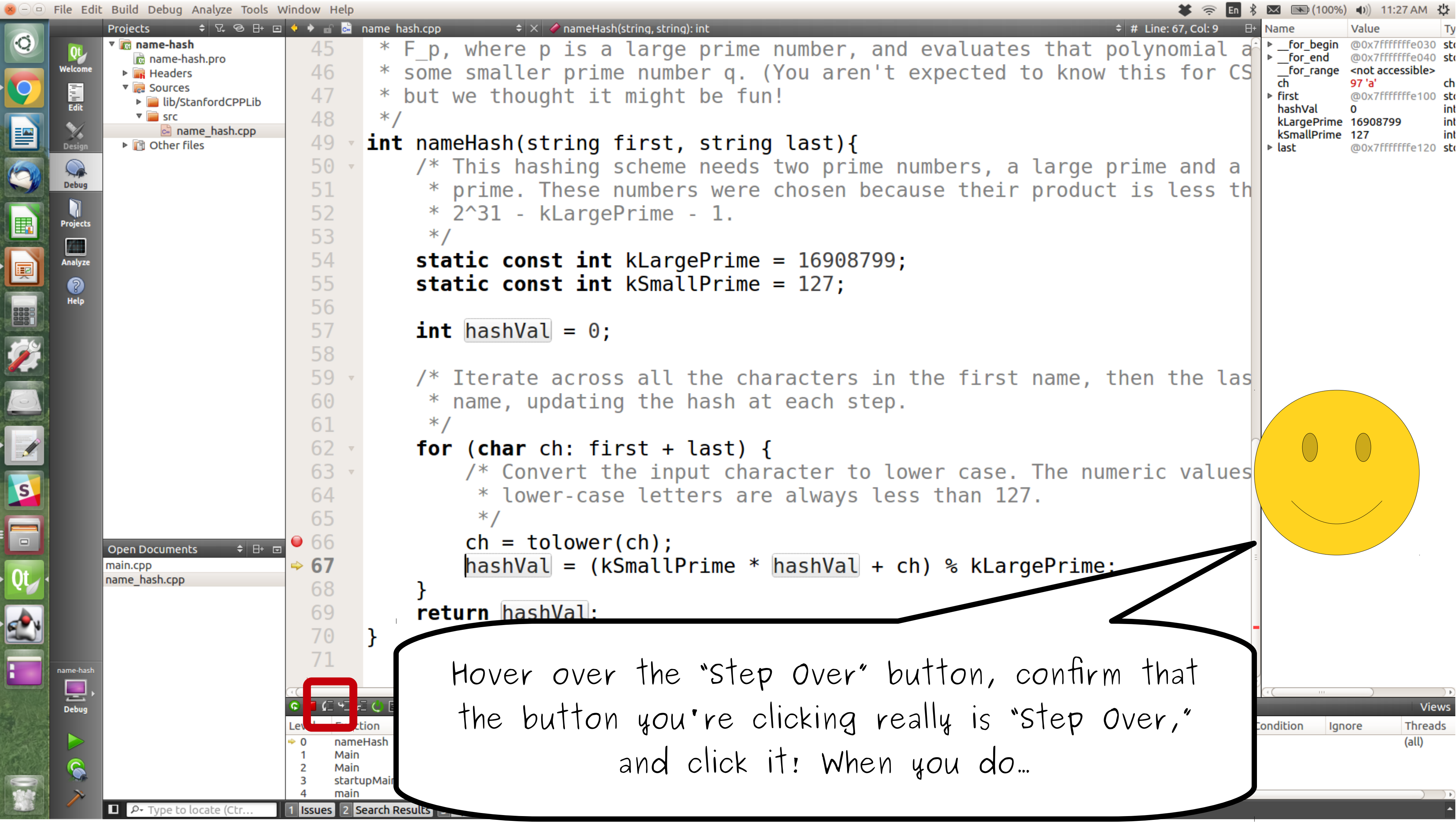
Let's go run that line of code and see what happens!

```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 *                                                              for CS
    static                   gePrime = 16908799;
    stat        int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

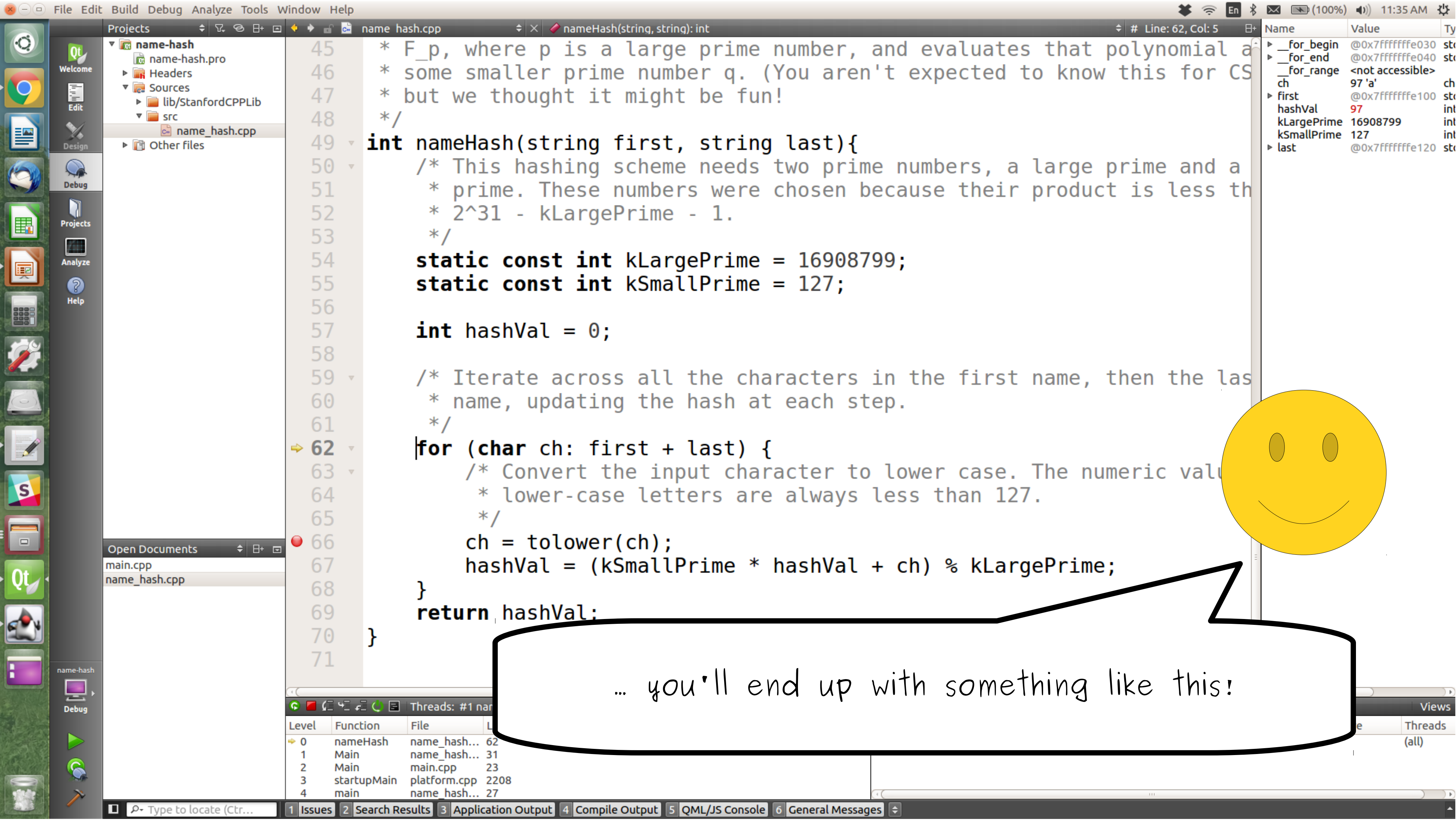


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric values
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	0	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function
0	nameHash
1	Main
2	Main
3	startupMai
4	main

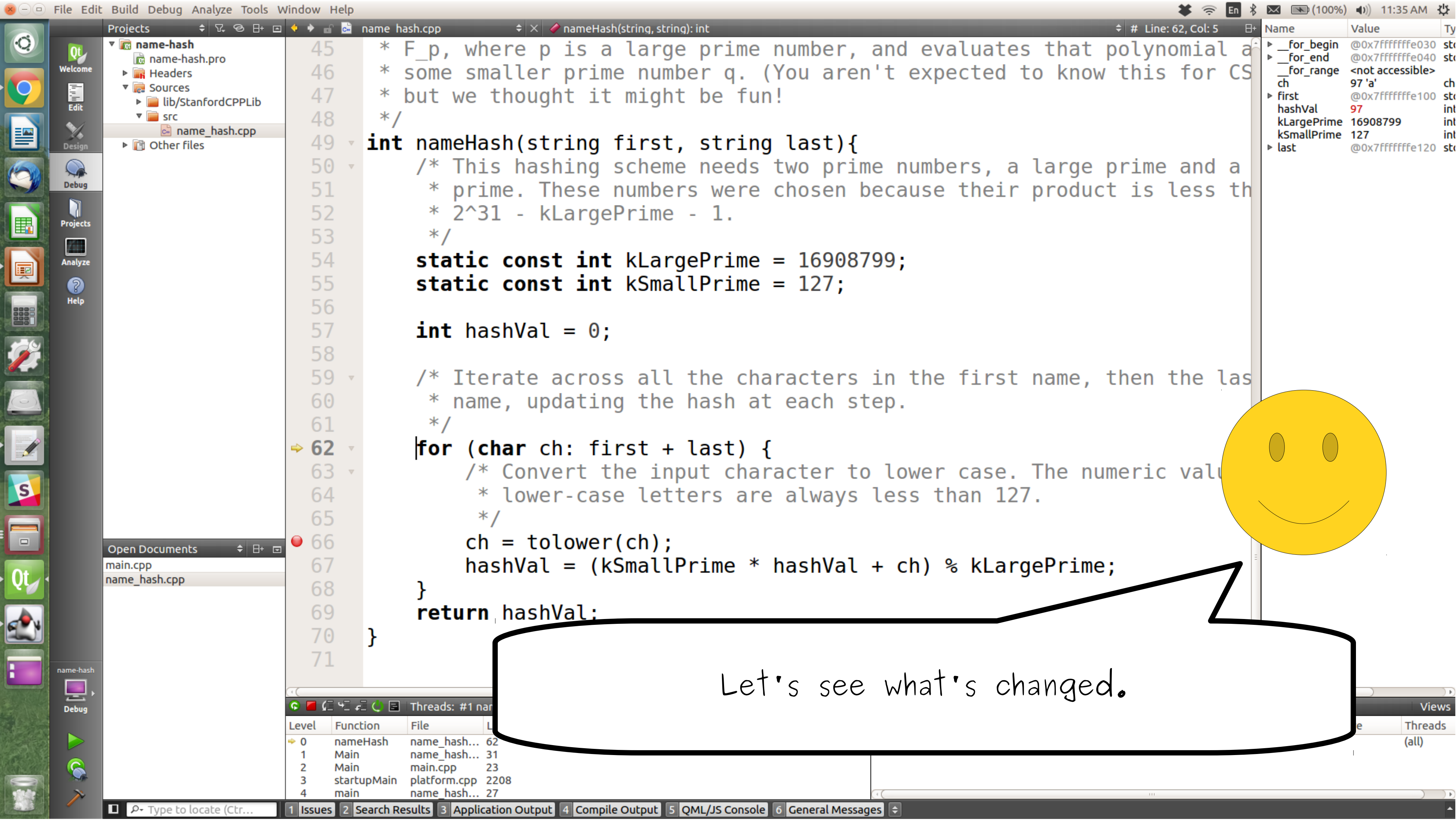
Hover over the "Step Over" button, confirm that the button you're clicking really is "Step Over," and click it! When you do…



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric valu
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

```
Name          Value              Ty
__for_begin   @0x7fffffffe030    st
__for_end     @0x7fffffffe040    st
__for_range   <not accessible>
ch            97 'a'             ch
first         @0x7fffffffe100    st
hashVal       97                 int
kLargePrime   16908799           int
kSmallPrime   127                int
last          @0x7fffffffe120    st
```

```
Level  Function     File           L
0      nameHash     name_hash...   62
1      Main         name_hash...   31
2      Main         main.cpp       23
3      startupMain  platform.cpp   2208
4      main         name_hash...   27
```

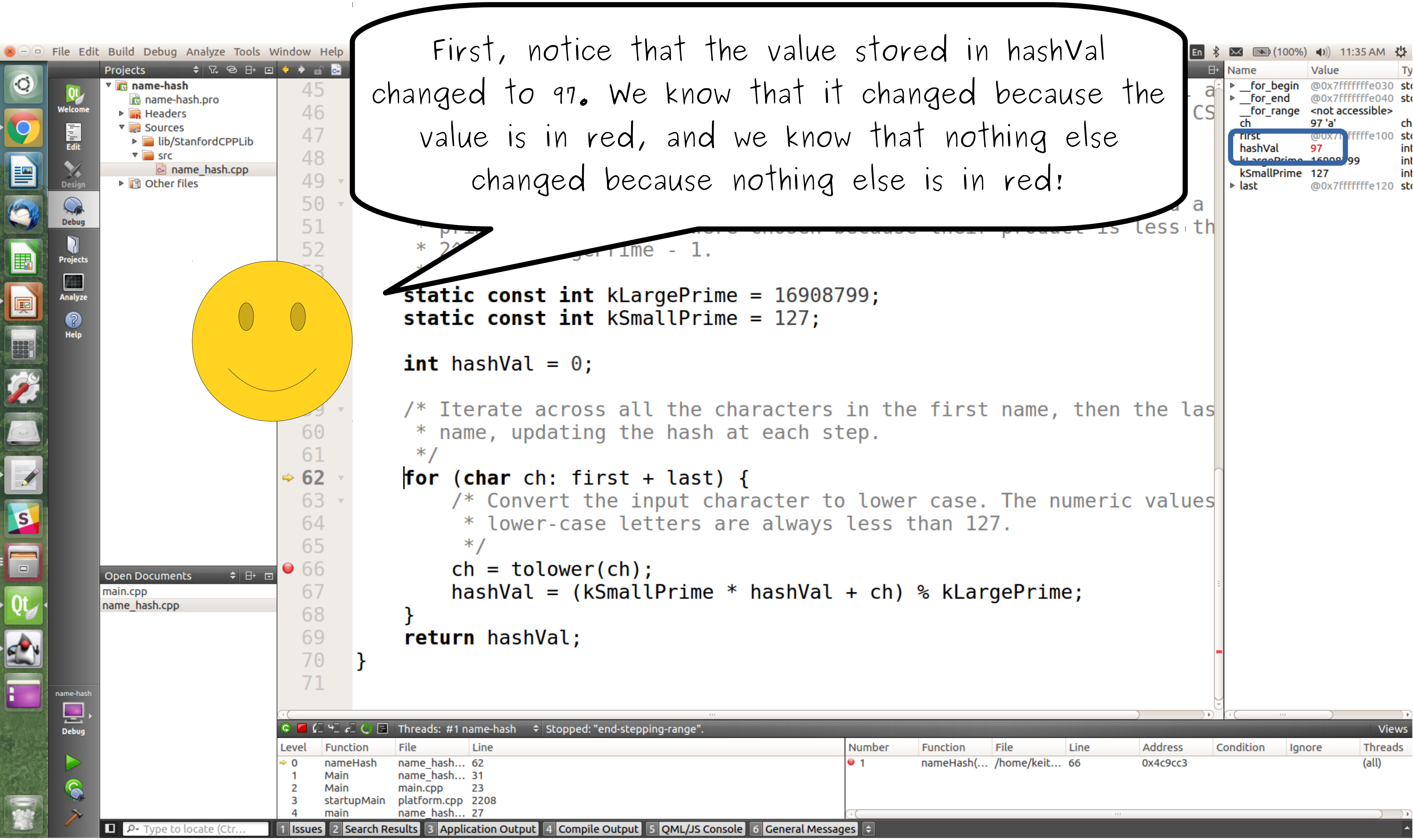


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric valu
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	97	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level	Function	File	L
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

First, notice that the value stored in hashVal changed to 97. We know that it changed because the value is in red, and we know that nothing else changed because nothing else is in red!



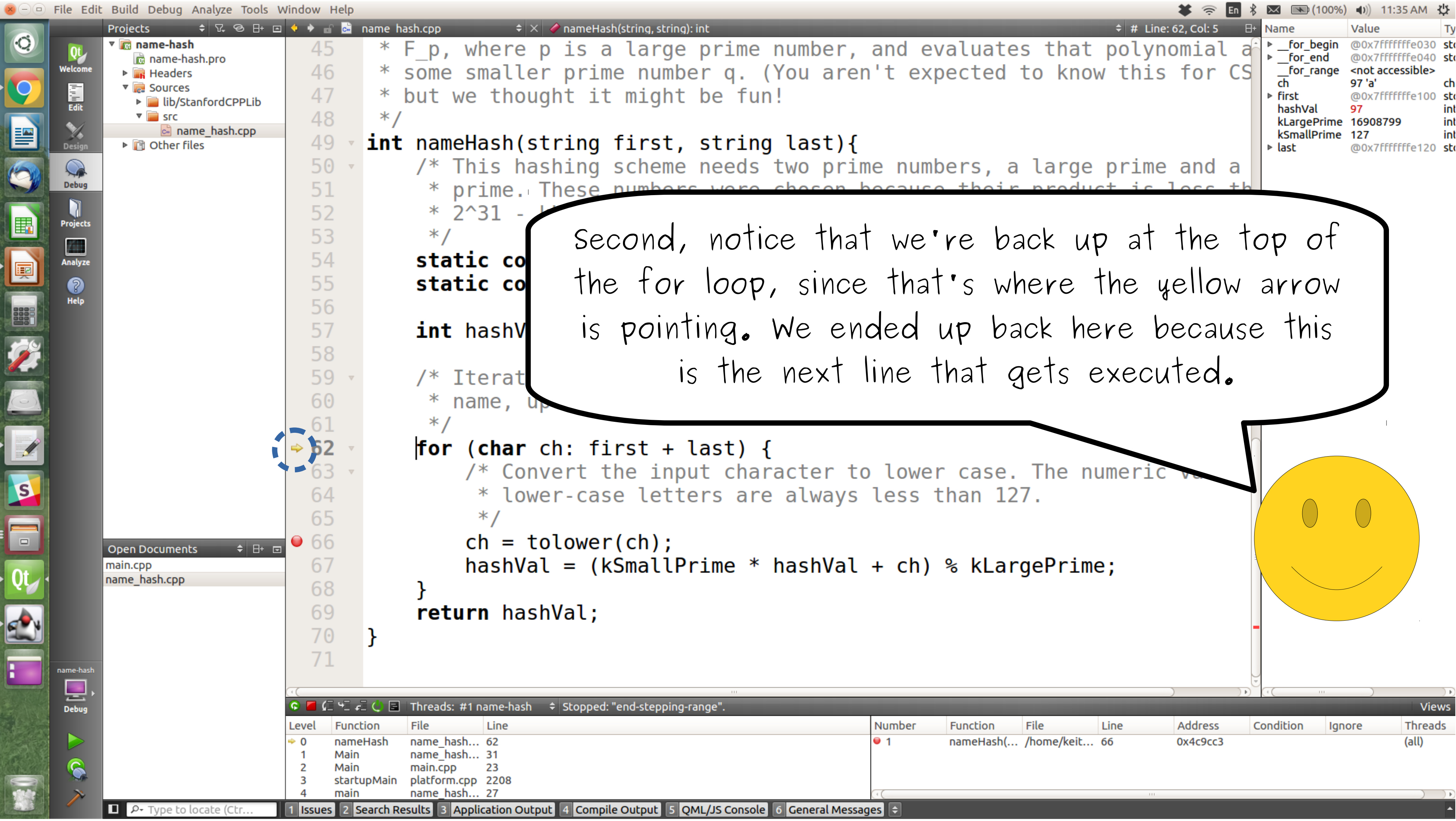
```
   * pri                                    less th
   * 20                 rime - 1.
   */
  static const int kLargePrime = 16908799;
  static const int kSmallPrime = 127;

  int hashVal = 0;

  /* Iterate across all the characters in the first name, then the las
   * name, updating the hash at each step.
   */
  for (char ch: first + last) {
      /* Convert the input character to lower case. The numeric values
       * lower-case letters are always less than 127.
       */
      ch = tolower(ch);
      hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
  }
  return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	97	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



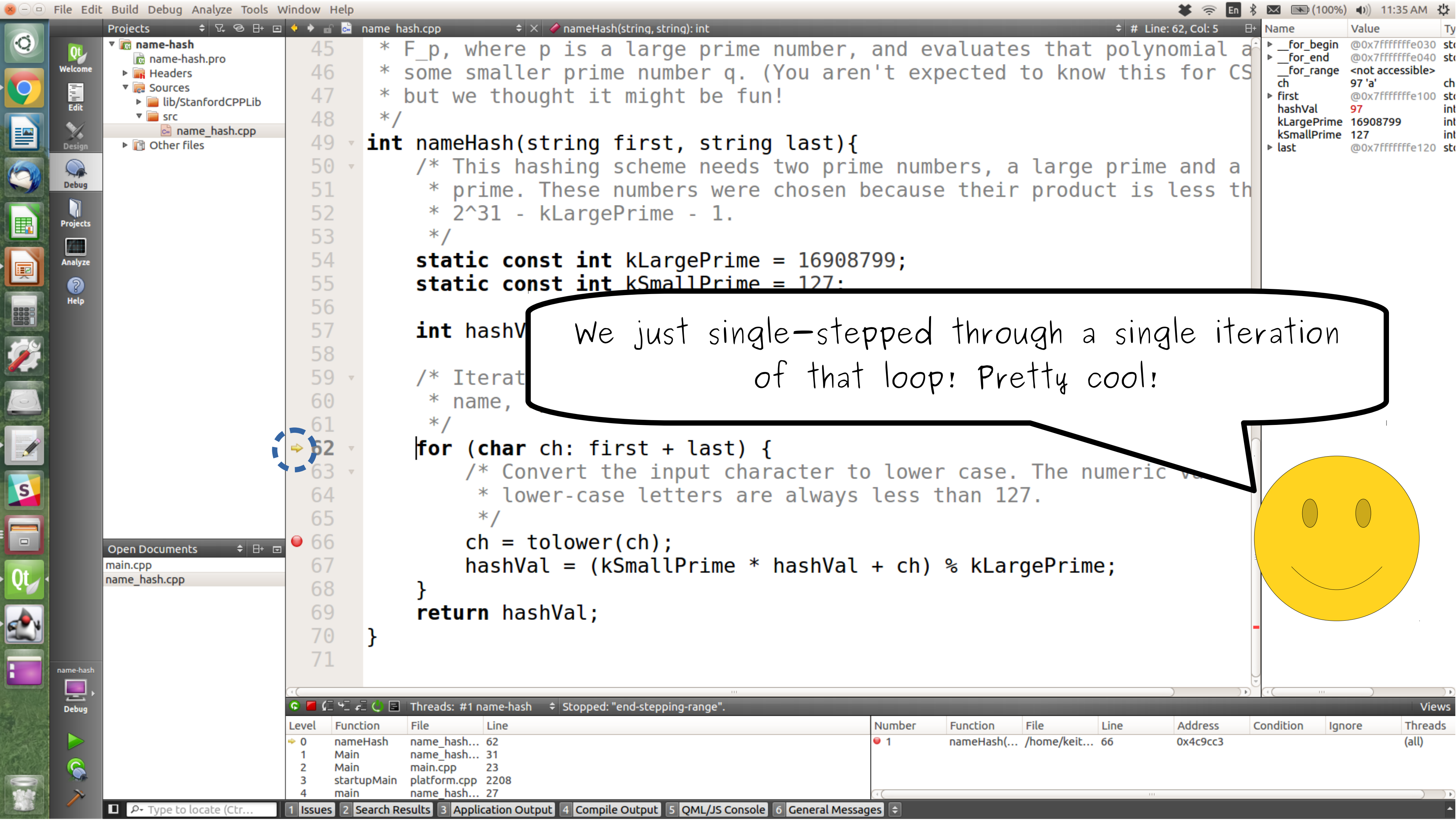
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - 
     */
    static co
    static co

    int hashV

    /* Iterat
     * name, up
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric va
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	97	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Second, notice that we're back up at the top of the for loop, since that's where the yellow arrow is pointing. We ended up back here because this is the next line that gets executed.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric va
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

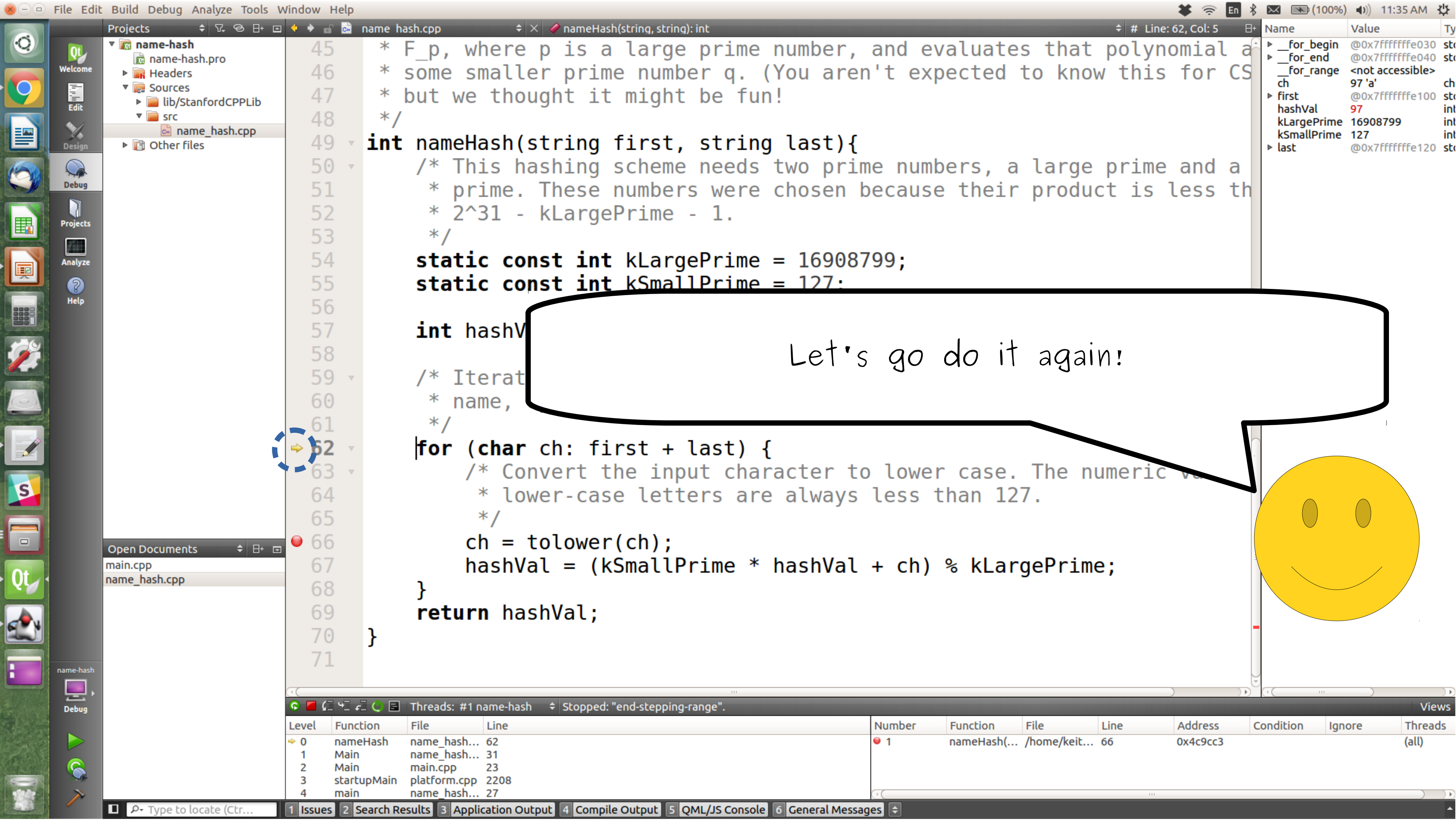
Name	Value	Ty
▸ __for_begin	@0x7fffffffe030	st
▸ __for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
▸ first	@0x7fffffffe100	st
hashVal	97	int
kLargePrime	16908799	int
kSmallPrime	127	int
▸ last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

We just single-stepped through a single iteration of that loop! Pretty cool!



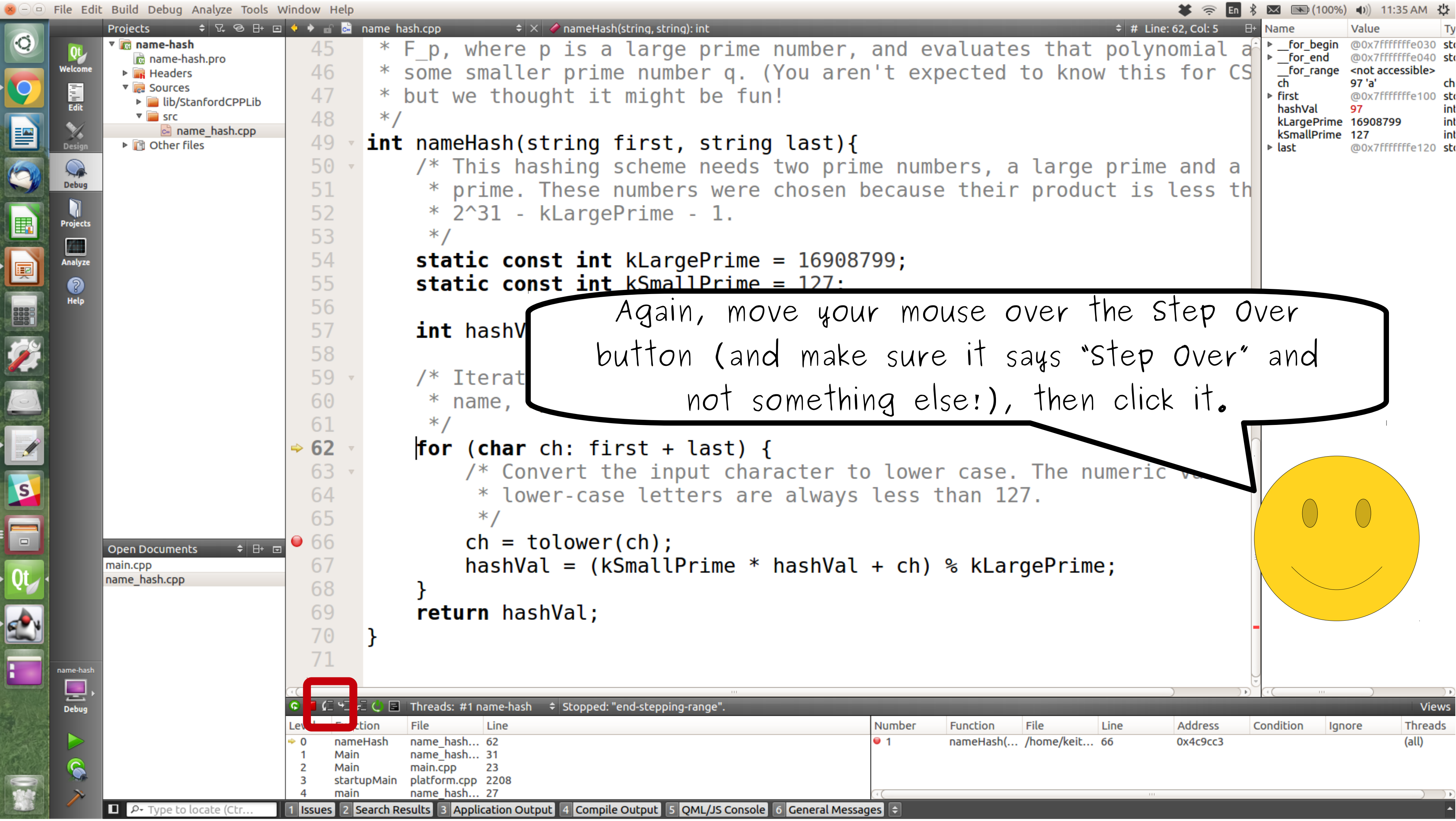
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric v
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	97	int
kLargePrime	16908799	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric va
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

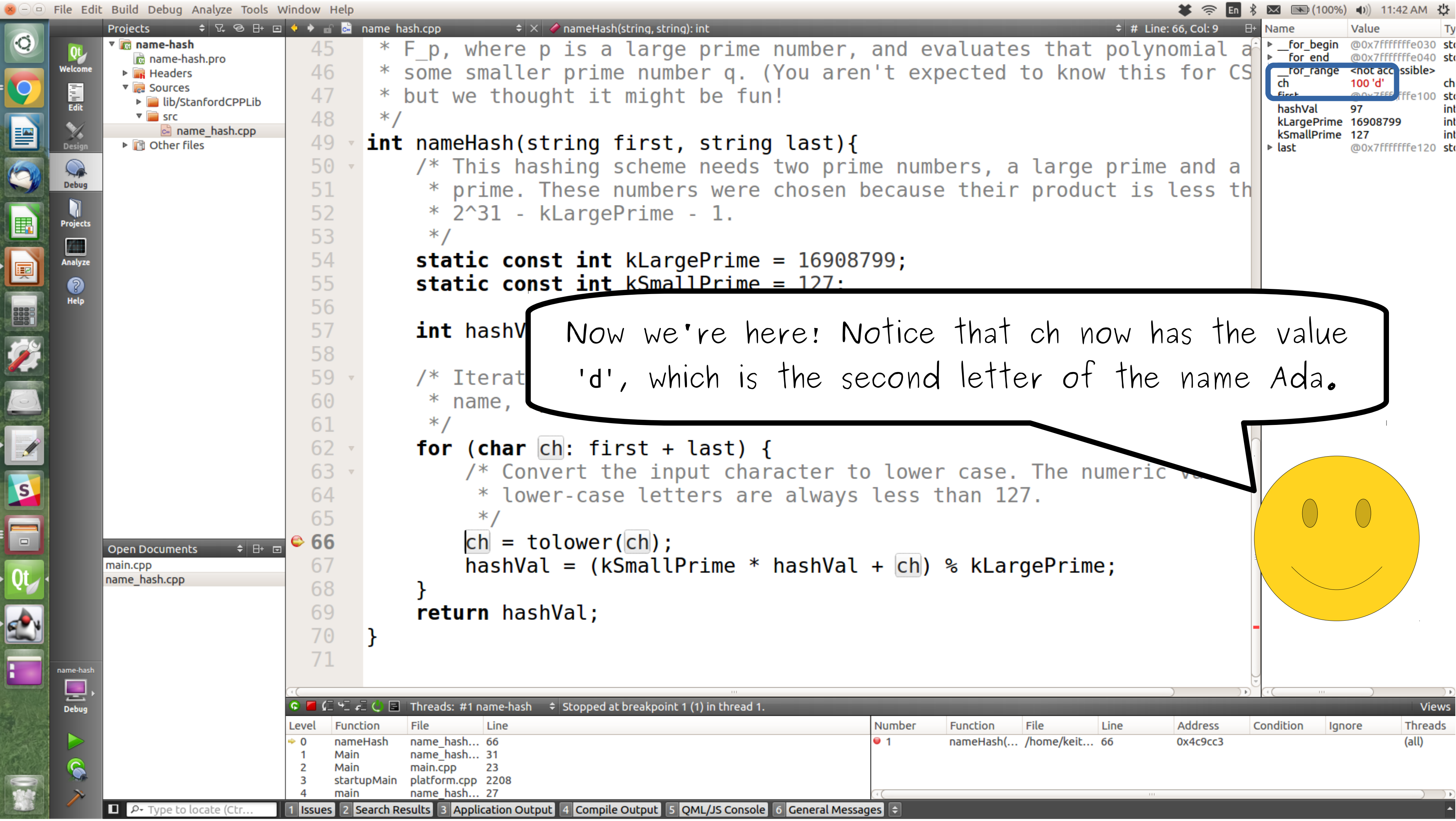
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	97 'a'	ch
first	@0x7fffffffe100	st
hashVal	97	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped: "end-stepping-range".

Le	ction	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Again, move your mouse over the Step Over button (and make sure it says "Step Over" and not something else!), then click it.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric va
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

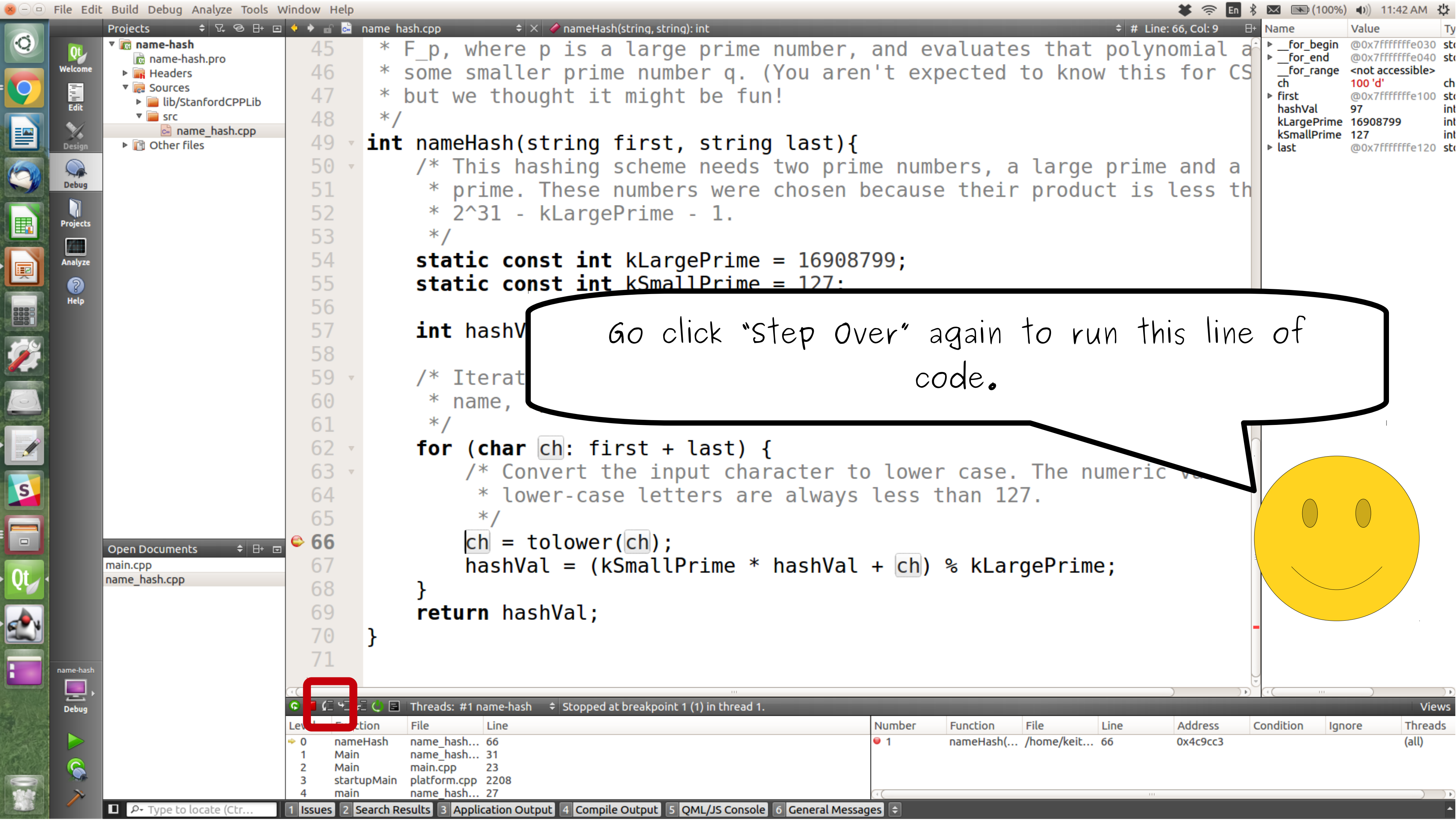
Name	Value	Ty
▸ __for_begin	@0x7fffffffe030	st
▸ __for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	@0x7fffffffe100	st
hashVal	97	int
kLargePrime	16908799	int
kSmallPrime	127	int
▸ last	@0x7fffffffe120	st

Stopped at breakpoint 1 (1) in thread 1.

Level	Function	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Now we're here! Notice that ch now has the value 'd', which is the second letter of the name Ada.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric va
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

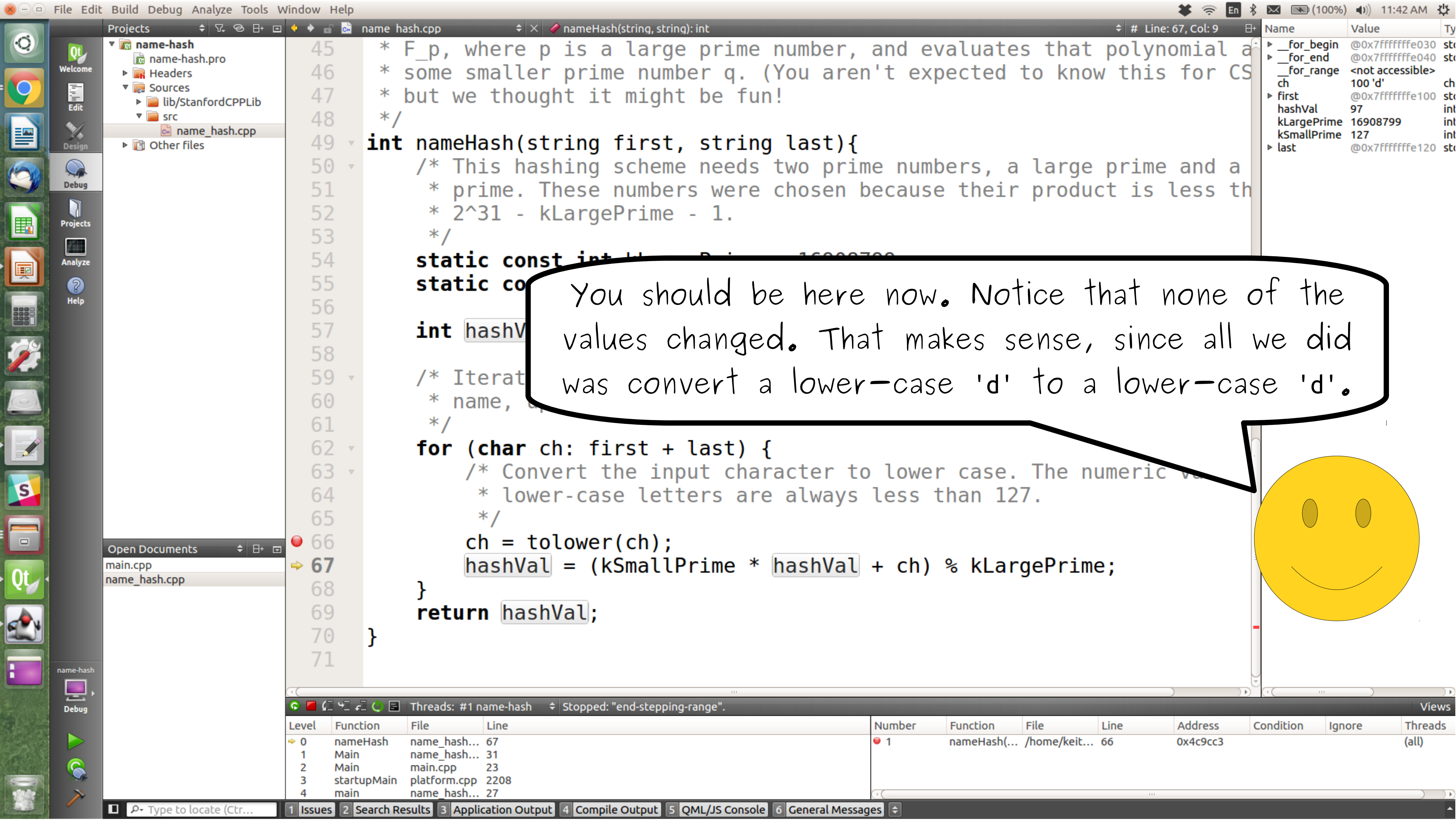
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	@0x7fffffffe100	st
hashVal	97	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash   Stopped at breakpoint 1 (1) in thread 1.

Lev	ction	File	Line
0	nameHash	name_hash...	66
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
● 1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

Go click "Step Over" again to run this line of code.



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int
    static co

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

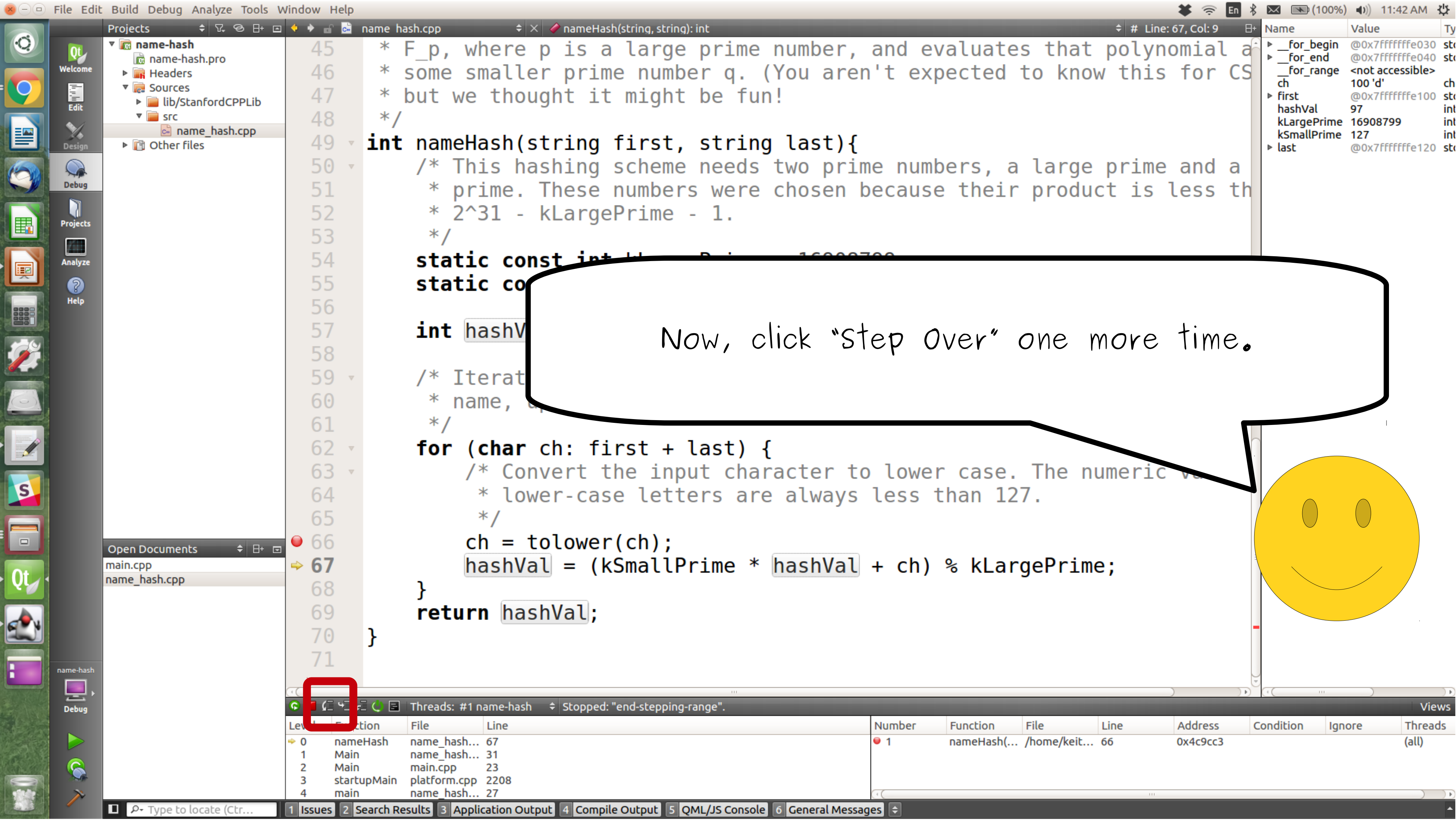
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	@0x7fffffffe100	st
hashVal	97	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

You should be here now. Notice that none of the values changed. That makes sense, since all we did was convert a lower-case 'd' to a lower-case 'd'.



Now, click "Step Over" one more time.

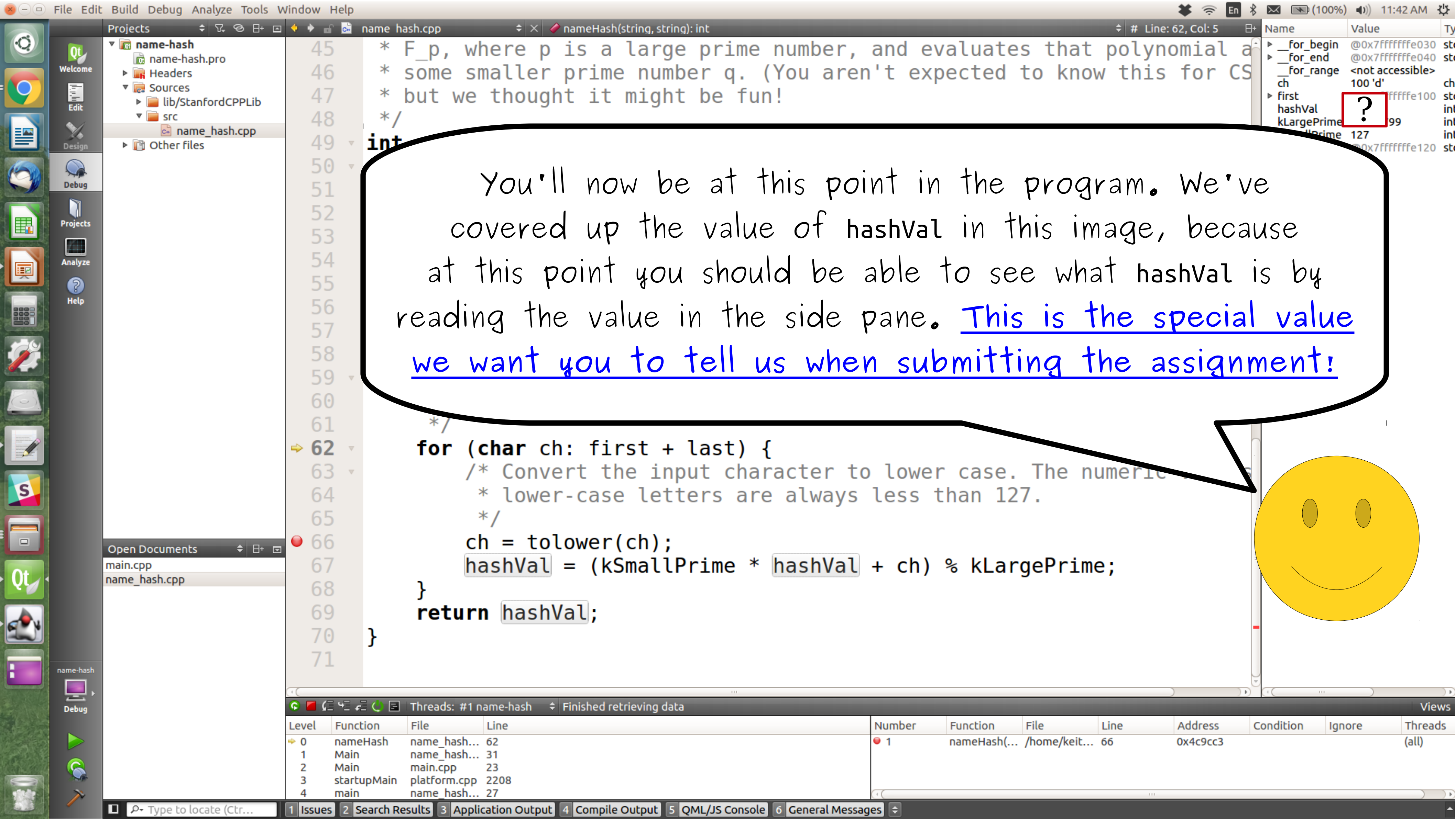
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int
    static co

    int hashV

    /* Iterat
     * name,
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric va
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	@0x7fffffffe100	st
hashVal	97	in
kLargePrime	16908799	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	67
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int
```

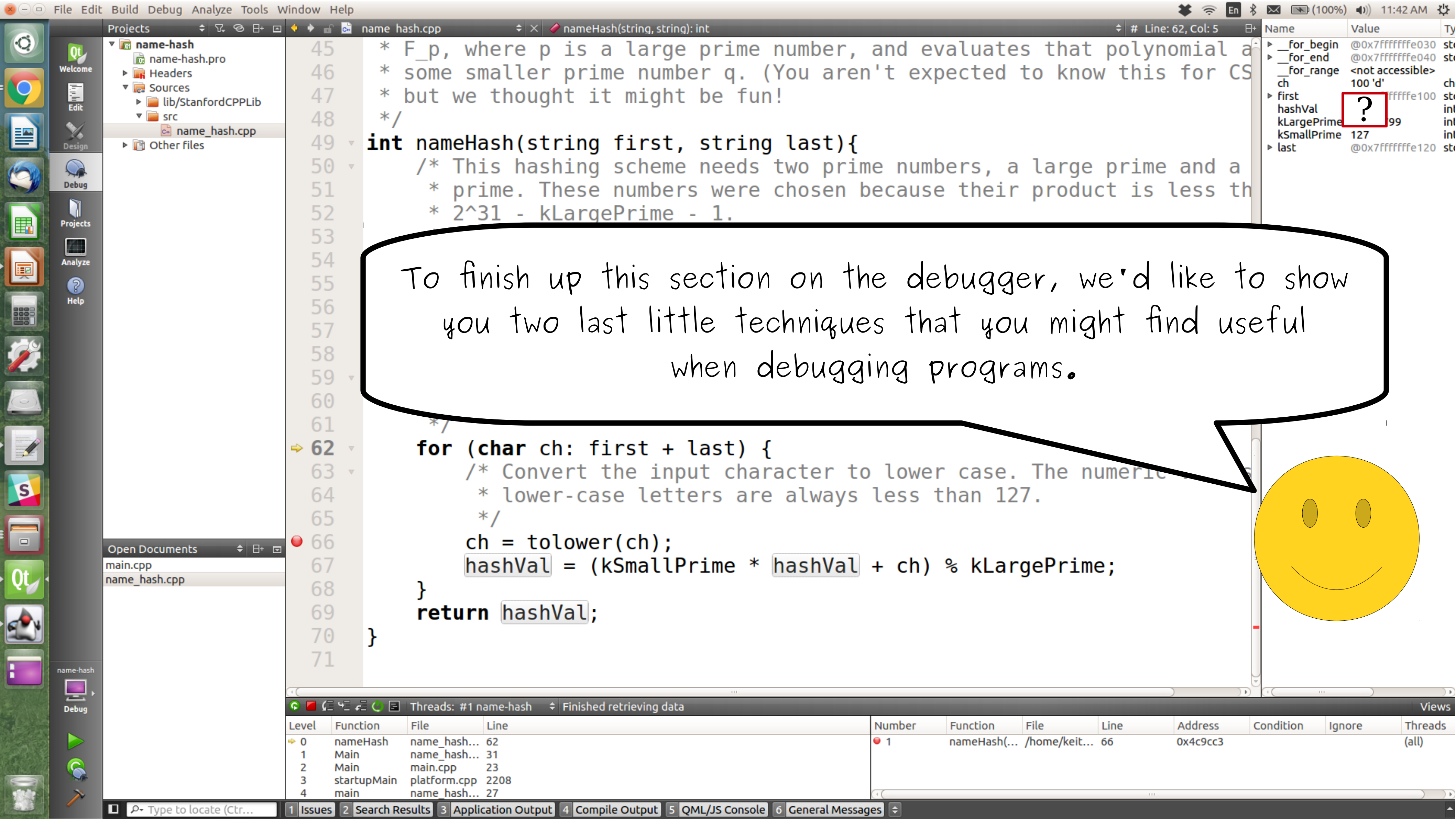
```
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	ffffffe100	st
hashVal	?	in
kLargePrime	99	in
llPrime	127	in
	0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

You'll now be at this point in the program. We've covered up the value of **hashVal** in this image, because at this point you should be able to see what **hashVal** is by reading the value in the side pane. This is the special value we want you to tell us when submitting the assignment!



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
```

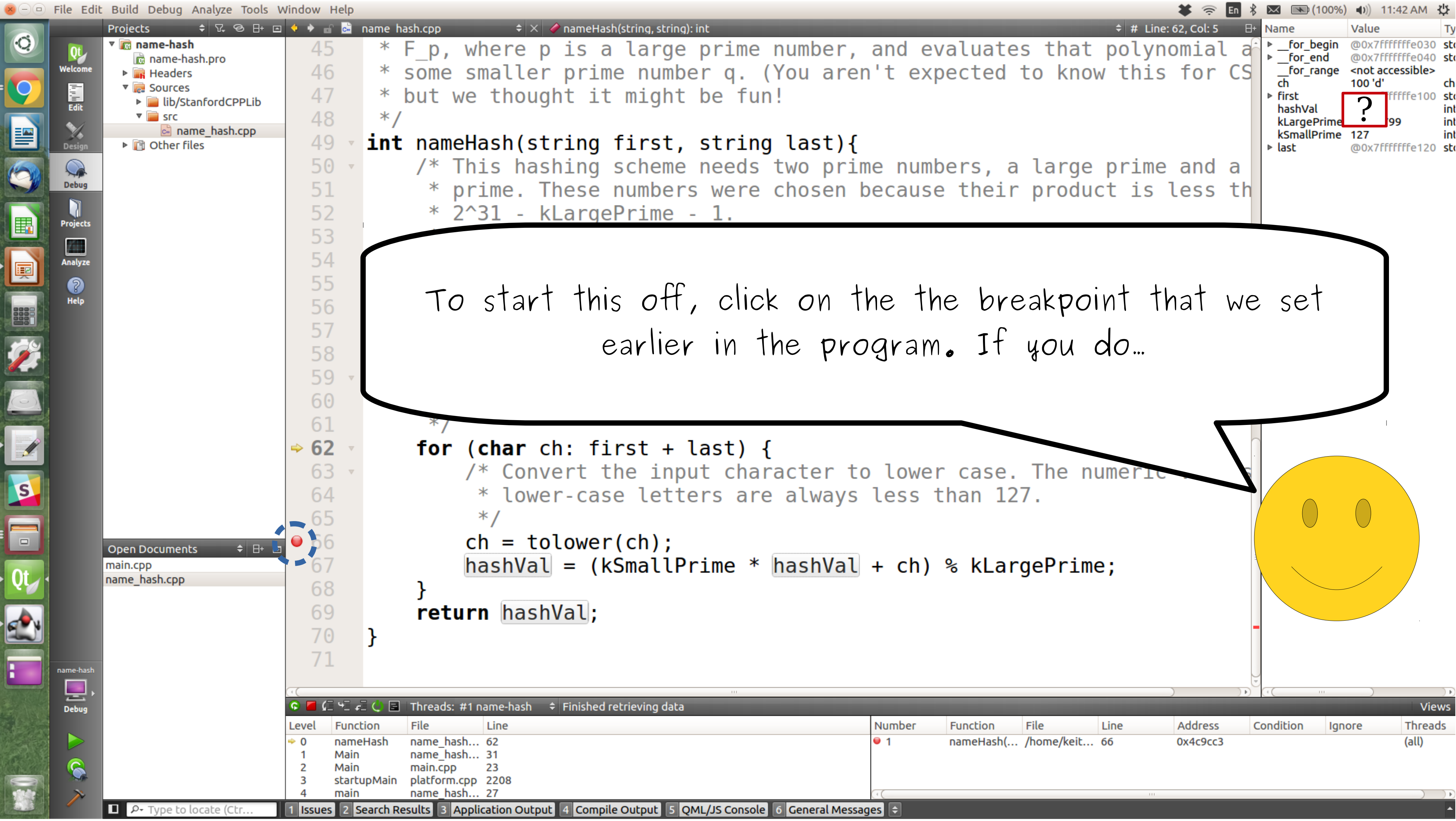
```
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	ffffe100	st
hashVal		in
kLargePrime	99	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

To finish up this section on the debugger, we'd like to show you two last little techniques that you might find useful when debugging programs.



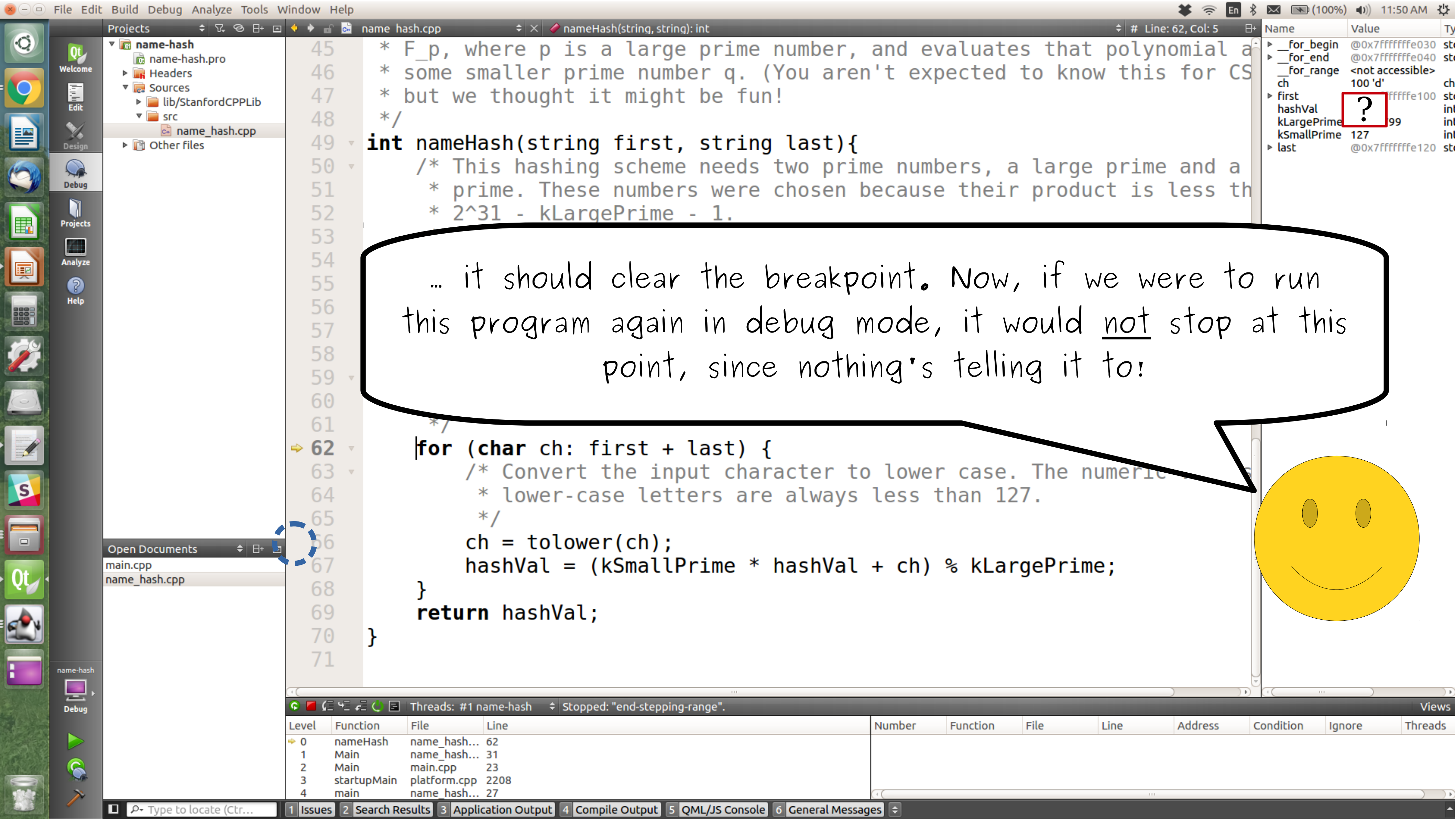
```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.

     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	fffffe100	st
hashVal	?	int
kLargePrime	99	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads
1	nameHash(...	/home/keit...	66	0x4c9cc3			(all)

To start this off, click on the the breakpoint that we set earlier in the program. If you do…



```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
```

```
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

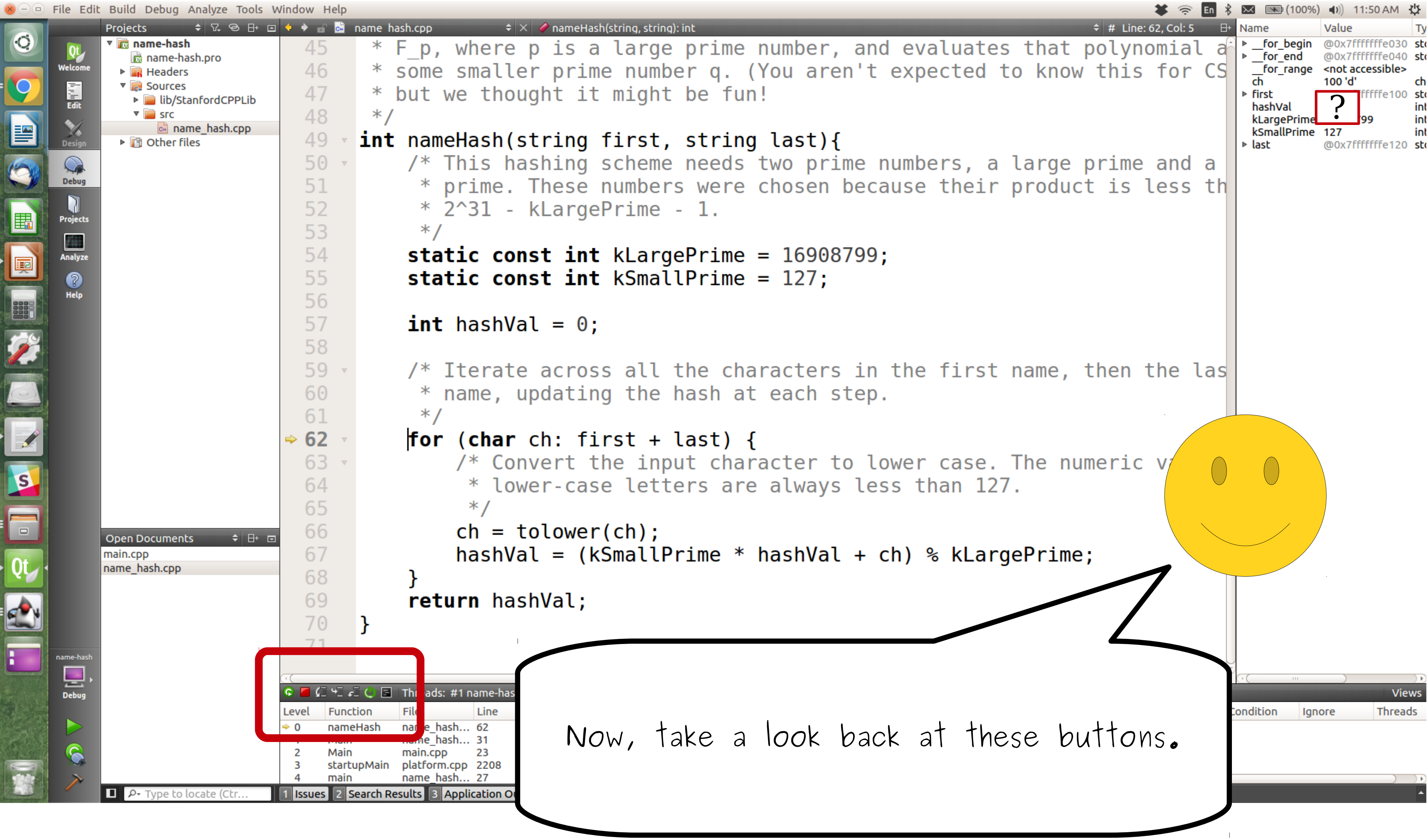
Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	?ffffe100	st
hashVal		in
kLargePrime	99	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Threads: #1 name-hash Stopped: "end-stepping-range".

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

… it should clear the breakpoint. Now, if we were to run this program again in debug mode, it would not stop at this point, since nothing's telling it to!

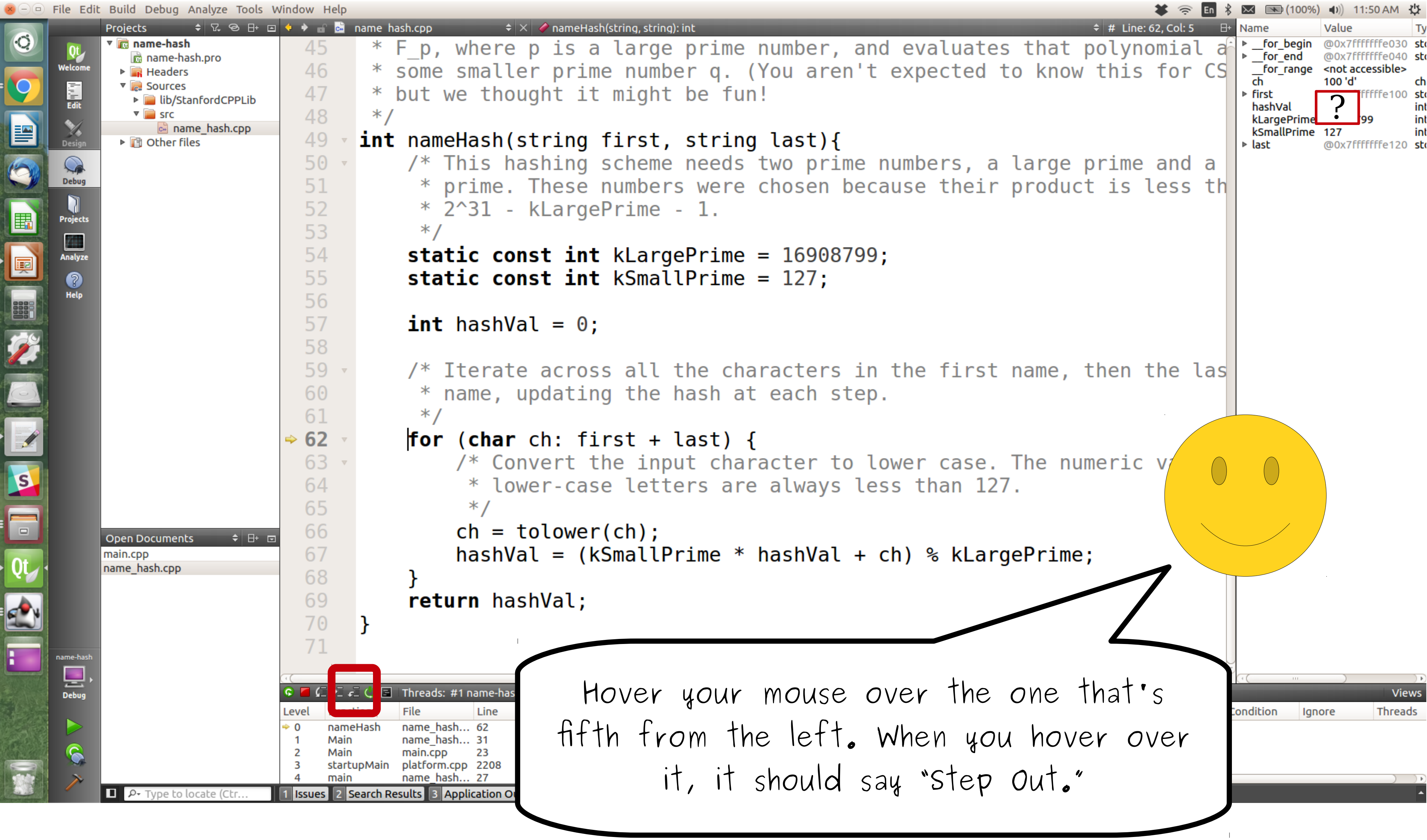


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric v
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Level	Function	File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	ffffe100	st
hashVal		in
kLargePrime	99	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Now, take a look back at these buttons.

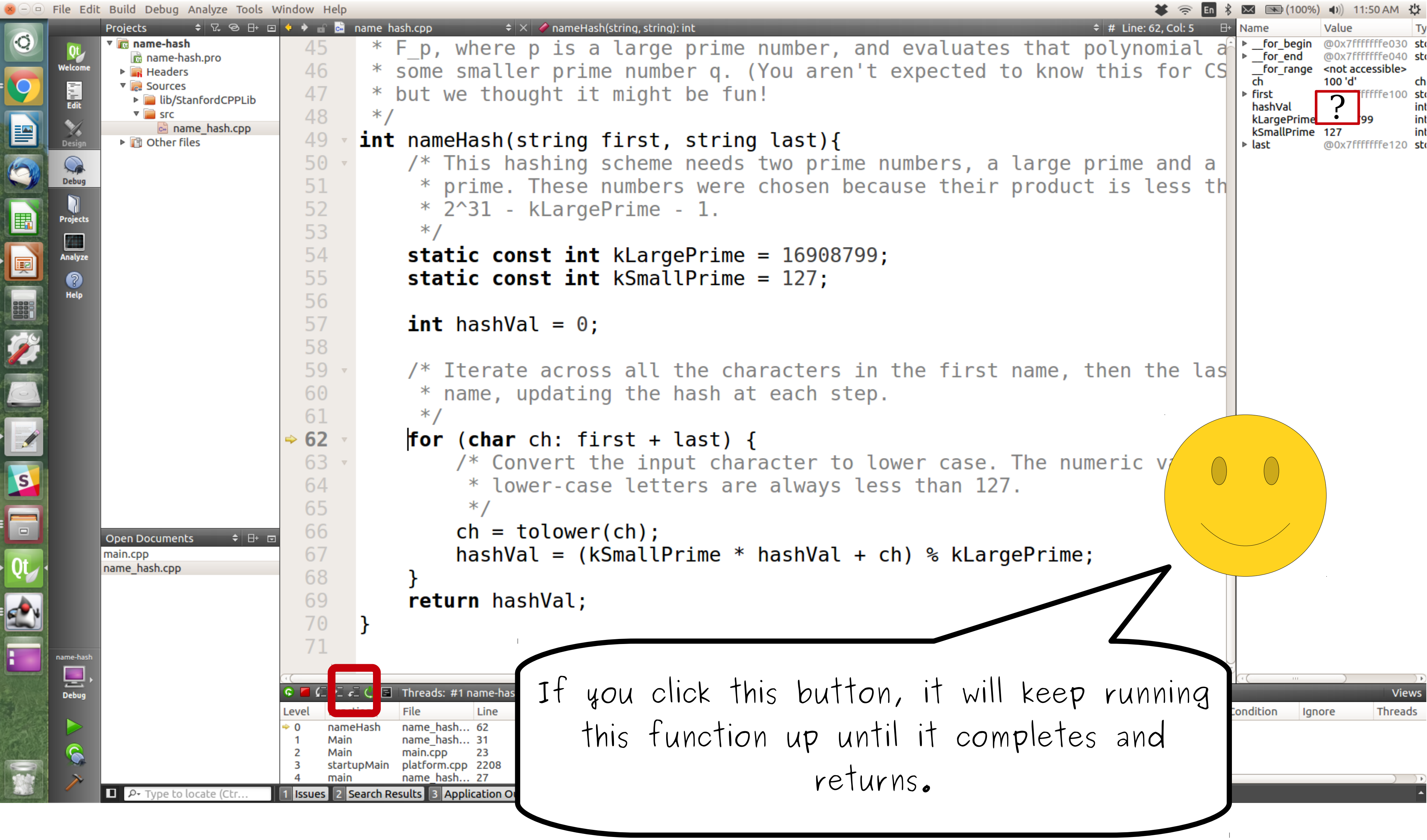


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric v
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	fffffe100	st
hashVal		int
kLargePrime	99	int
kSmallPrime	127	int
last	@0x7fffffffe120	st

Level		File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Hover your mouse over the one that's fifth from the left. When you hover over it, it should say "Step Out."

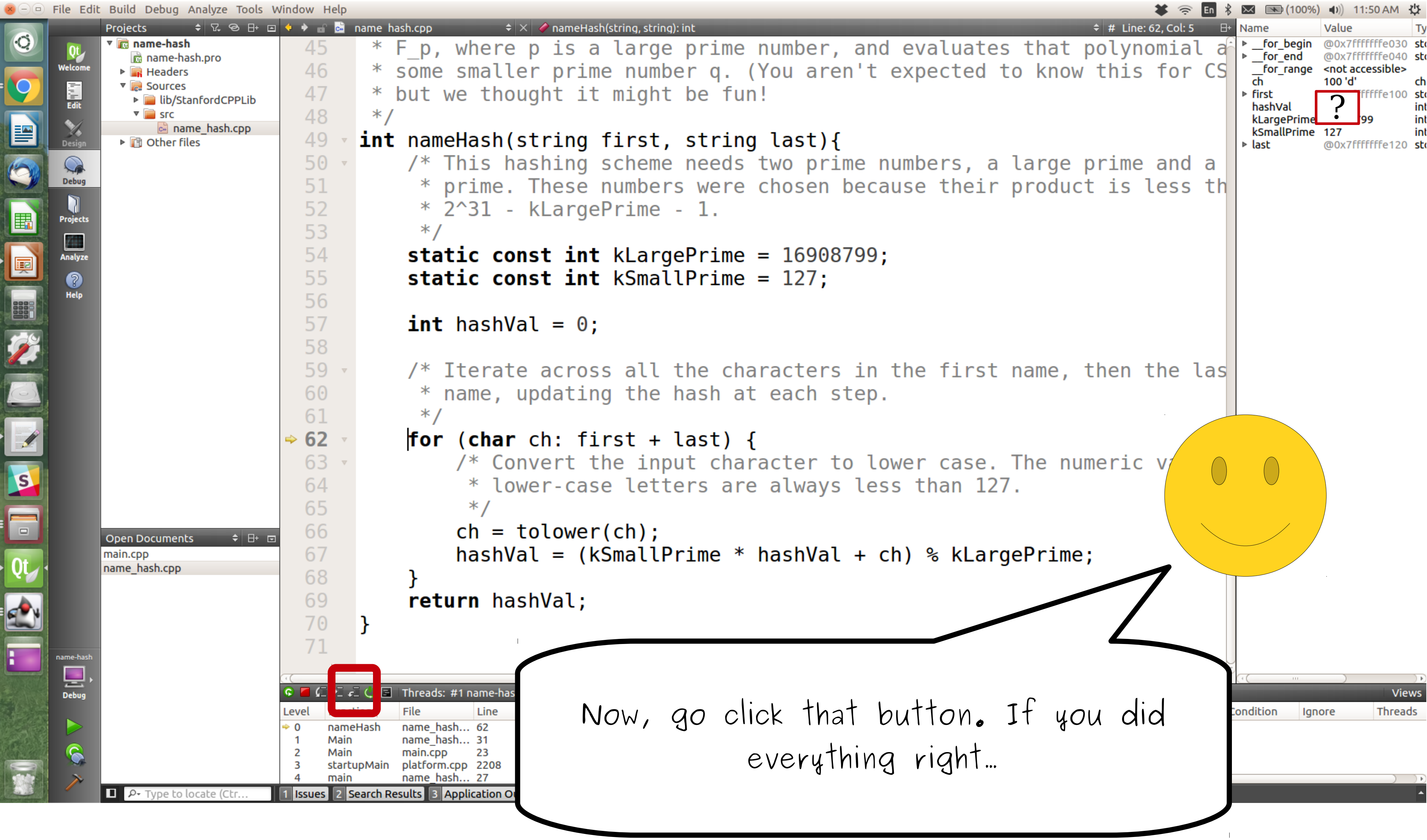


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric v
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	ffffffe100	st
hashVal		in
kLargePrime	99	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level		File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

If you click this button, it will keep running this function up until it completes and returns.

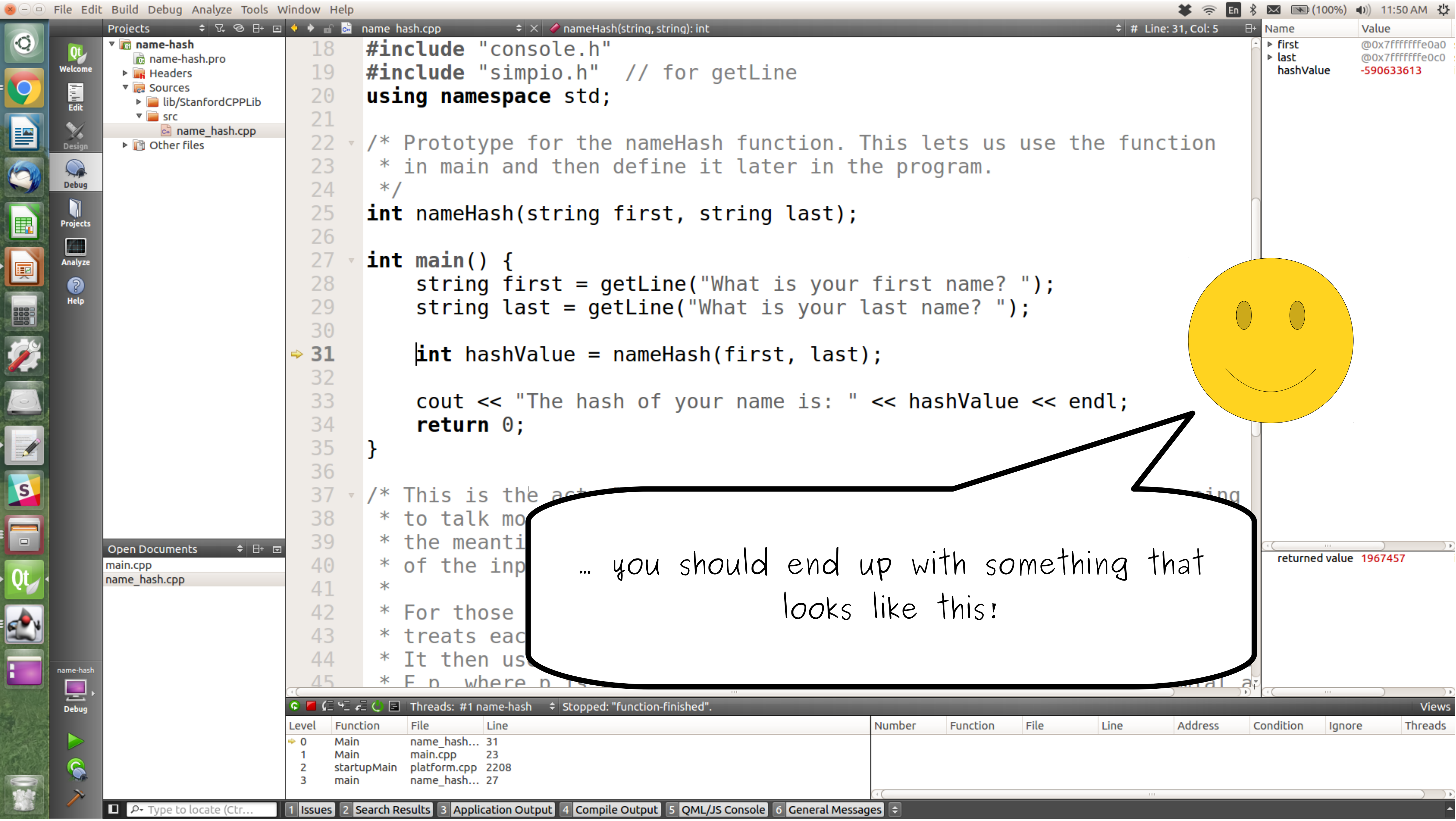


```
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
 */
int nameHash(string first, string last){
    /* This hashing scheme needs two prime numbers, a large prime and a
     * prime. These numbers were chosen because their product is less th
     * 2^31 - kLargePrime - 1.
     */
    static const int kLargePrime = 16908799;
    static const int kSmallPrime = 127;

    int hashVal = 0;

    /* Iterate across all the characters in the first name, then the las
     * name, updating the hash at each step.
     */
    for (char ch: first + last) {
        /* Convert the input character to lower case. The numeric v
         * lower-case letters are always less than 127.
         */
        ch = tolower(ch);
        hashVal = (kSmallPrime * hashVal + ch) % kLargePrime;
    }
    return hashVal;
}
```

Name	Value	Ty
__for_begin	@0x7fffffffe030	st
__for_end	@0x7fffffffe040	st
__for_range	<not accessible>	
ch	100 'd'	ch
first	fffffe100	st
hashVal	?	in
kLargePrime	99	in
kSmallPrime	127	in
last	@0x7fffffffe120	st

Level		File	Line
0	nameHash	name_hash...	62
1	Main	name_hash...	31
2	Main	main.cpp	23
3	startupMain	platform.cpp	2208
4	main	name_hash...	27

Now, go click that button. If you did everything right...



```
#include "console.h"
#include "simpio.h"   // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the ac                                                    ing
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F n  where n
```

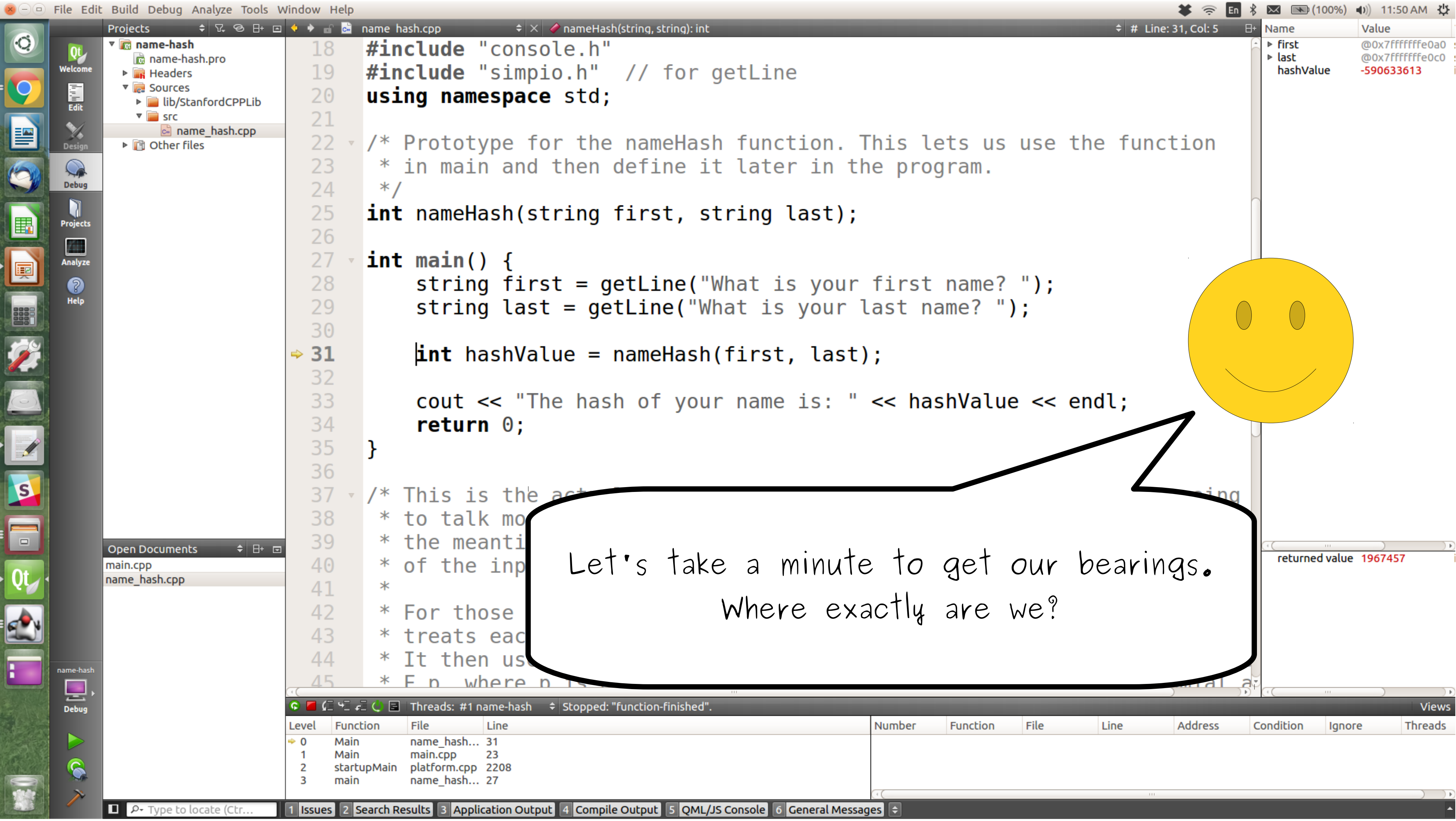
Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

… you should end up with something that looks like this!



```
#include "console.h"
#include "simpio.h"  // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the ac                                            ing
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F n  where n is
```

Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

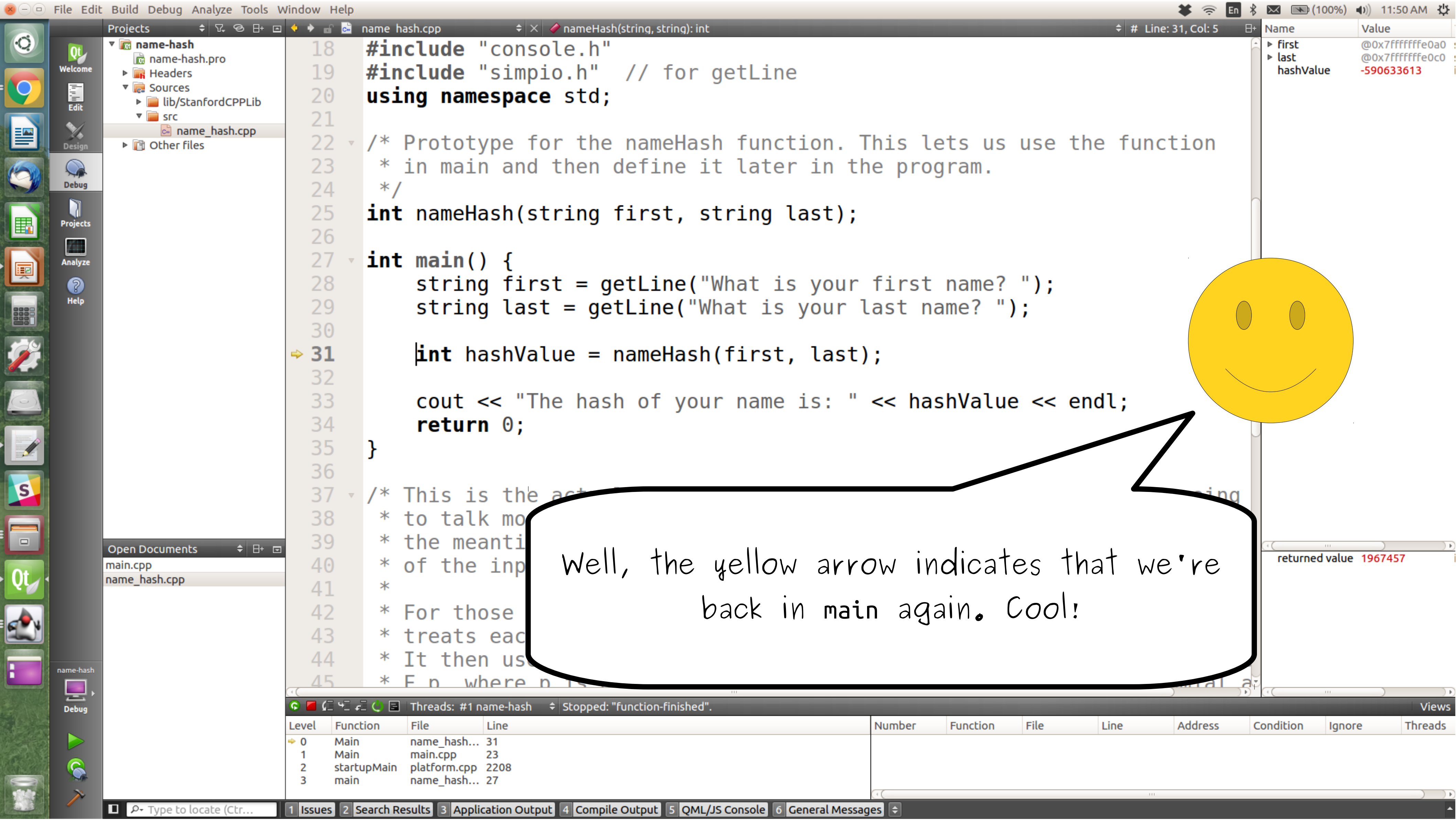
returned value 1967457

Threads: #1 name-hash Stopped: "function-finished".

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

Let's take a minute to get our bearings.
Where exactly are we?



```
#include "console.h"
#include "simpio.h"  // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the act                                              ing
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F n  where n
```

Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

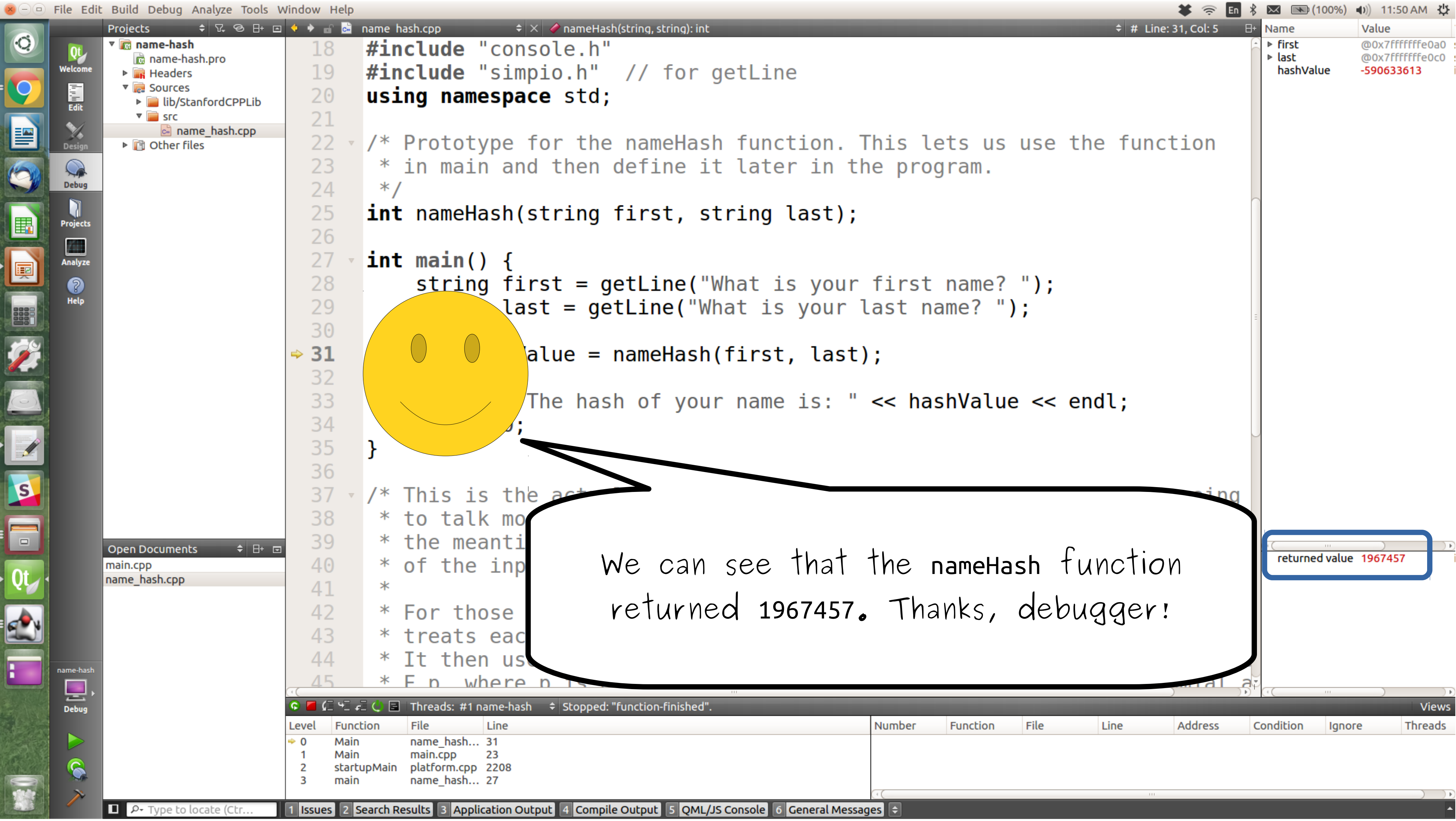
returned value 1967457

Threads: #1 name-hash   Stopped: "function-finished".

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

Well, the yellow arrow indicates that we're back in main again. Cool!



```
#include "console.h"
#include "simpio.h"  // for getLine
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
               last = getLine("What is your last name? ");

               alue = nameHash(first, last);

               The hash of your name is: " << hashValue << endl;
               ;
}

/* This is the act                                                  ing
 * to talk mo
 * the meanti
 * of the inp
 *
 * For those
 * treats eac
 * It then us
 * F_n where n is                                                al a
```

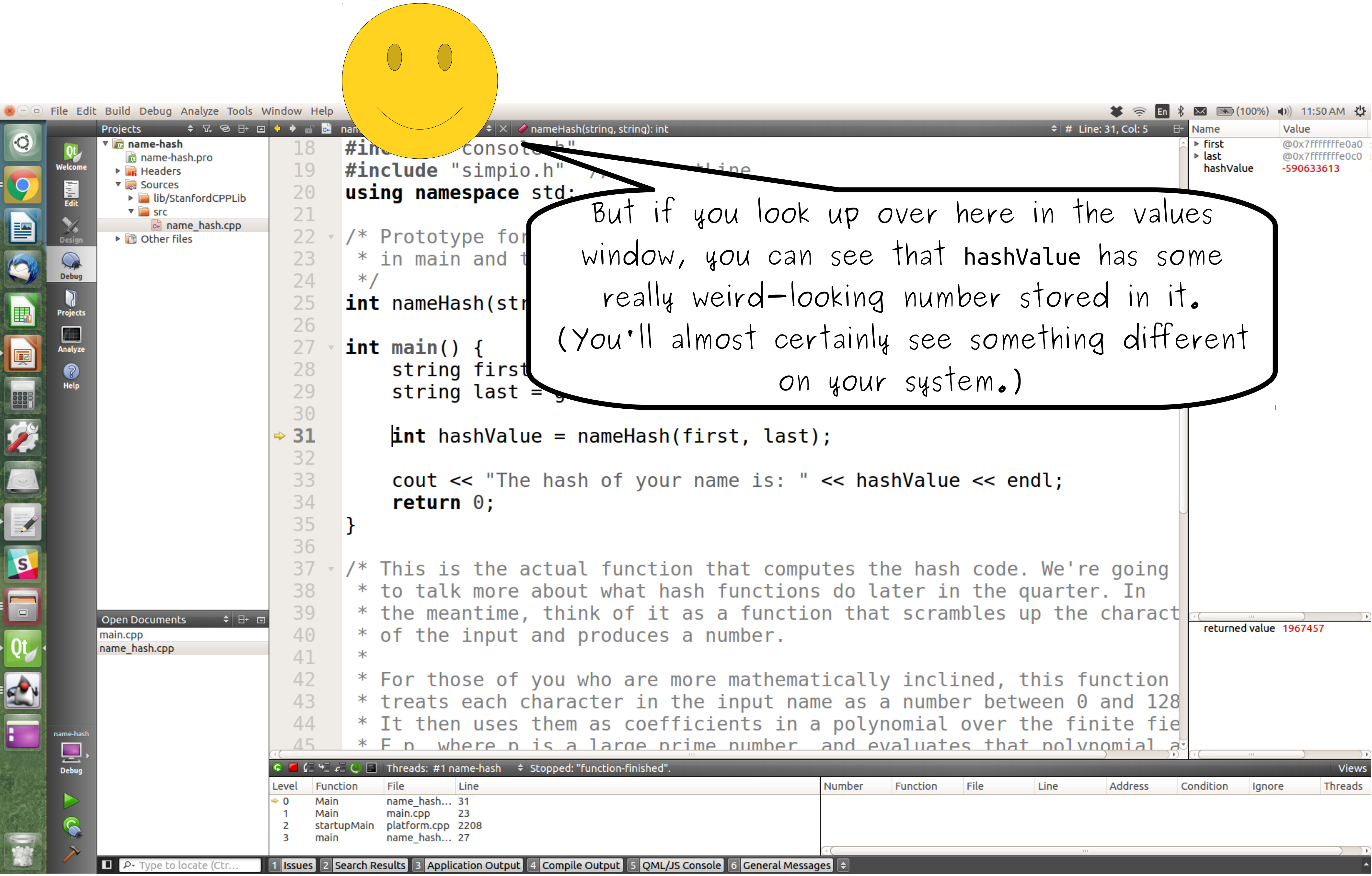
Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

We can see that the nameHash function returned 1967457. Thanks, debugger!



```
#in          consol  h"
#include "simpio.h"   //          tline
using namespace std;

/* Prototype for
 * in main and t
 */
int nameHash(str

int main() {
    string first
    string last = g

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
```

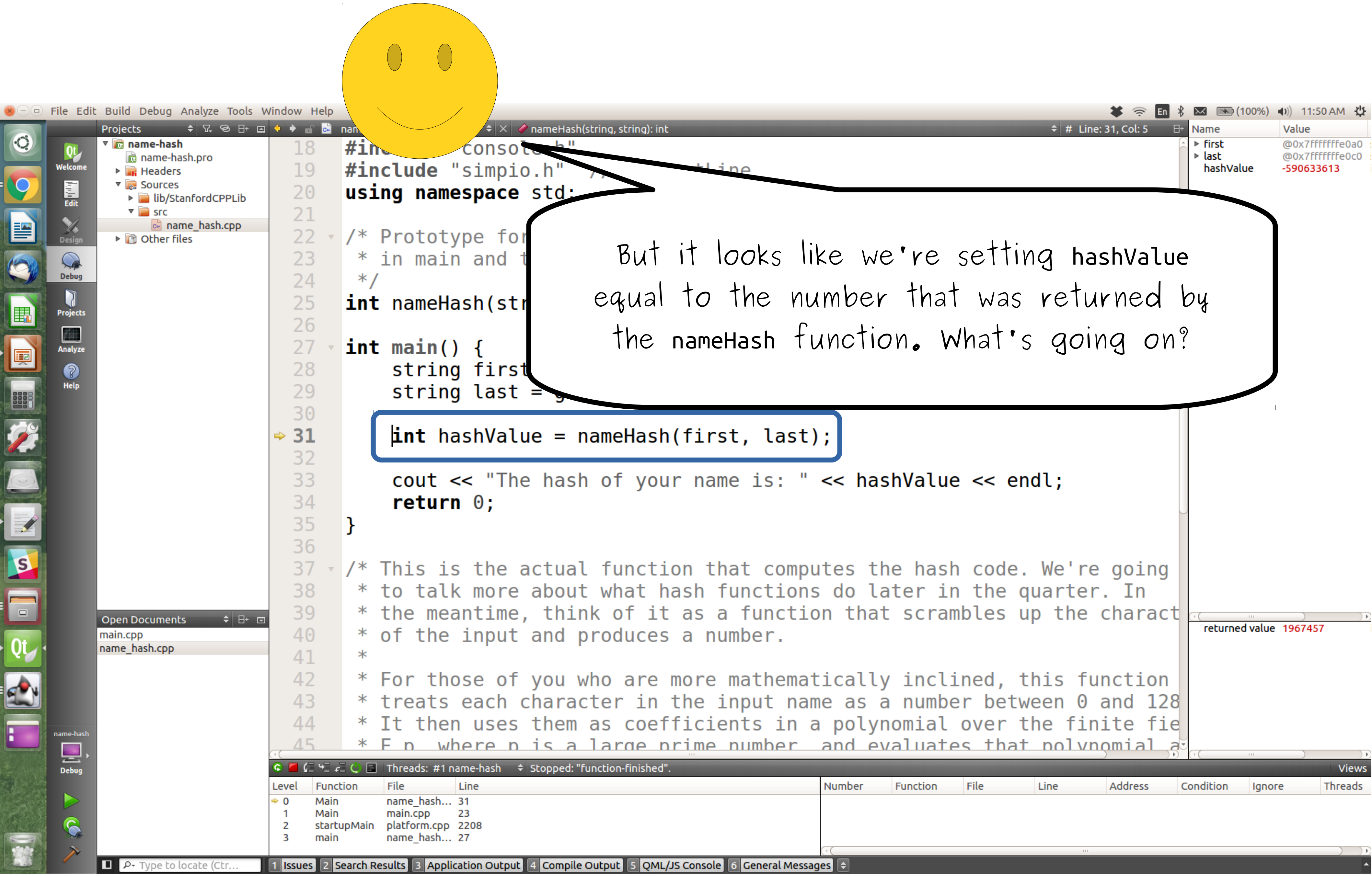
Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

But if you look up over here in the values window, you can see that **hashValue** has some really weird-looking number stored in it. (You'll almost certainly see something different on your system.)



But it looks like we're setting **hashValue** equal to the number that was returned by the **nameHash** function. What's going on?

```
#in       console.h"
#include "simpio.h"   //
using namespace std;

/* Prototype for
 * in main and t
 */
int nameHash(str

int main() {
    string first
    string last = g

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
```

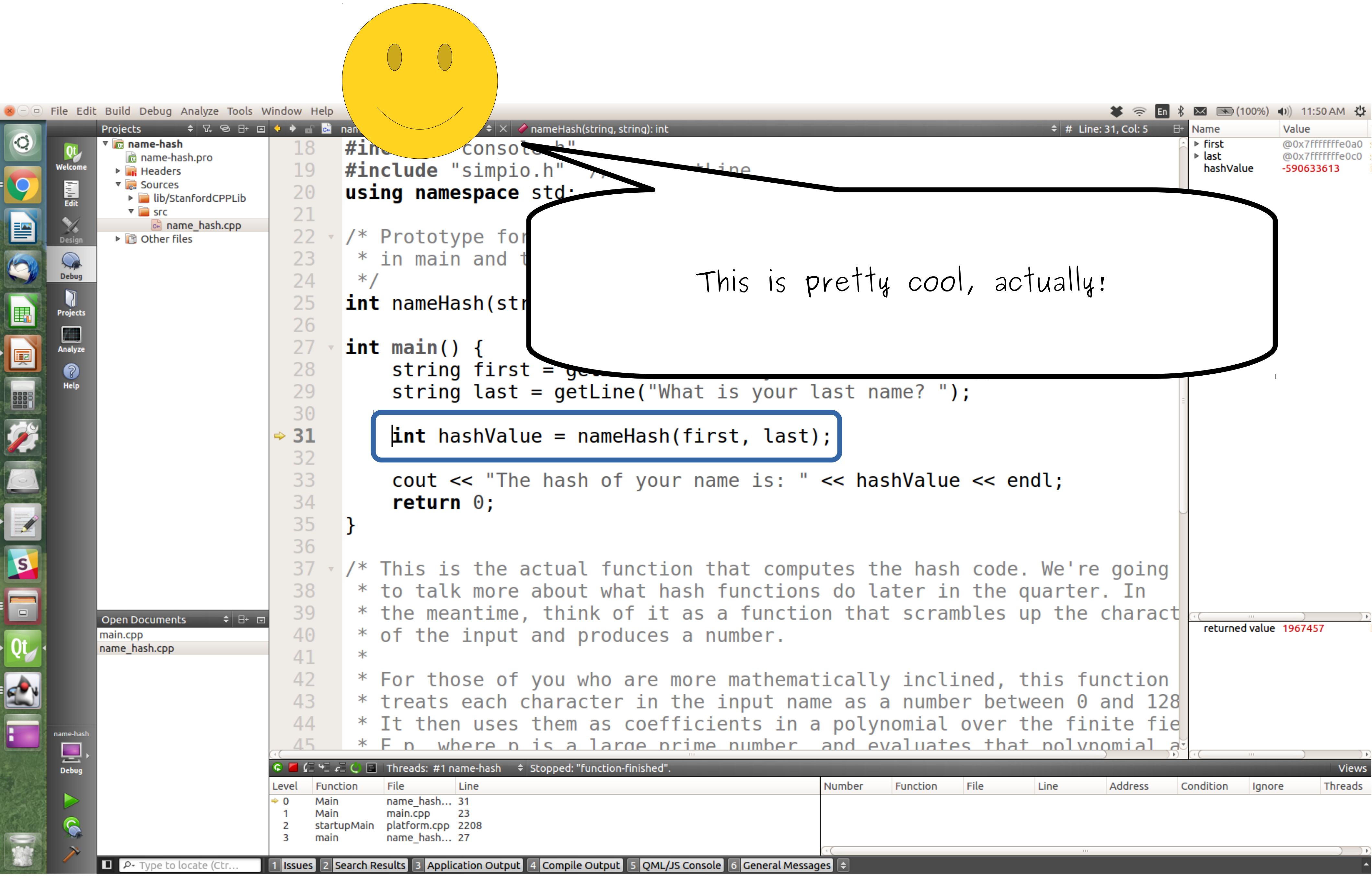
Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Threads: #1 name-hash   Stopped: "function-finished".

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads



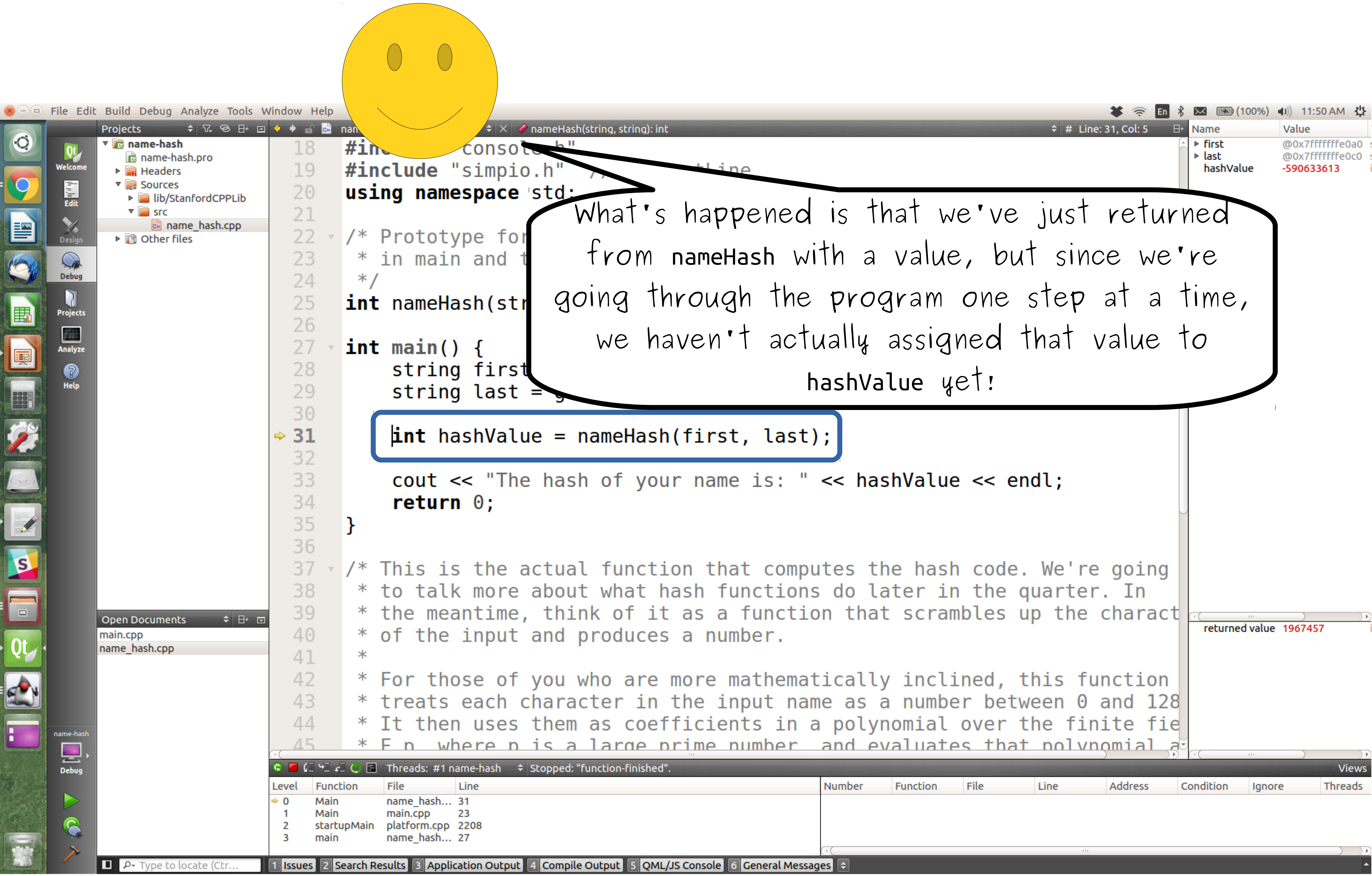
```
#in       conso  h"
#include "simpio.h"   /
using namespace std;

/* Prototype for
 * in main and t
 */
int nameHash(str

int main() {
    string first = ge
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
```

Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads



What's happened is that we've just returned from **nameHash** with a value, but since we're going through the program one step at a time, we haven't actually assigned that value to **hashValue** yet!

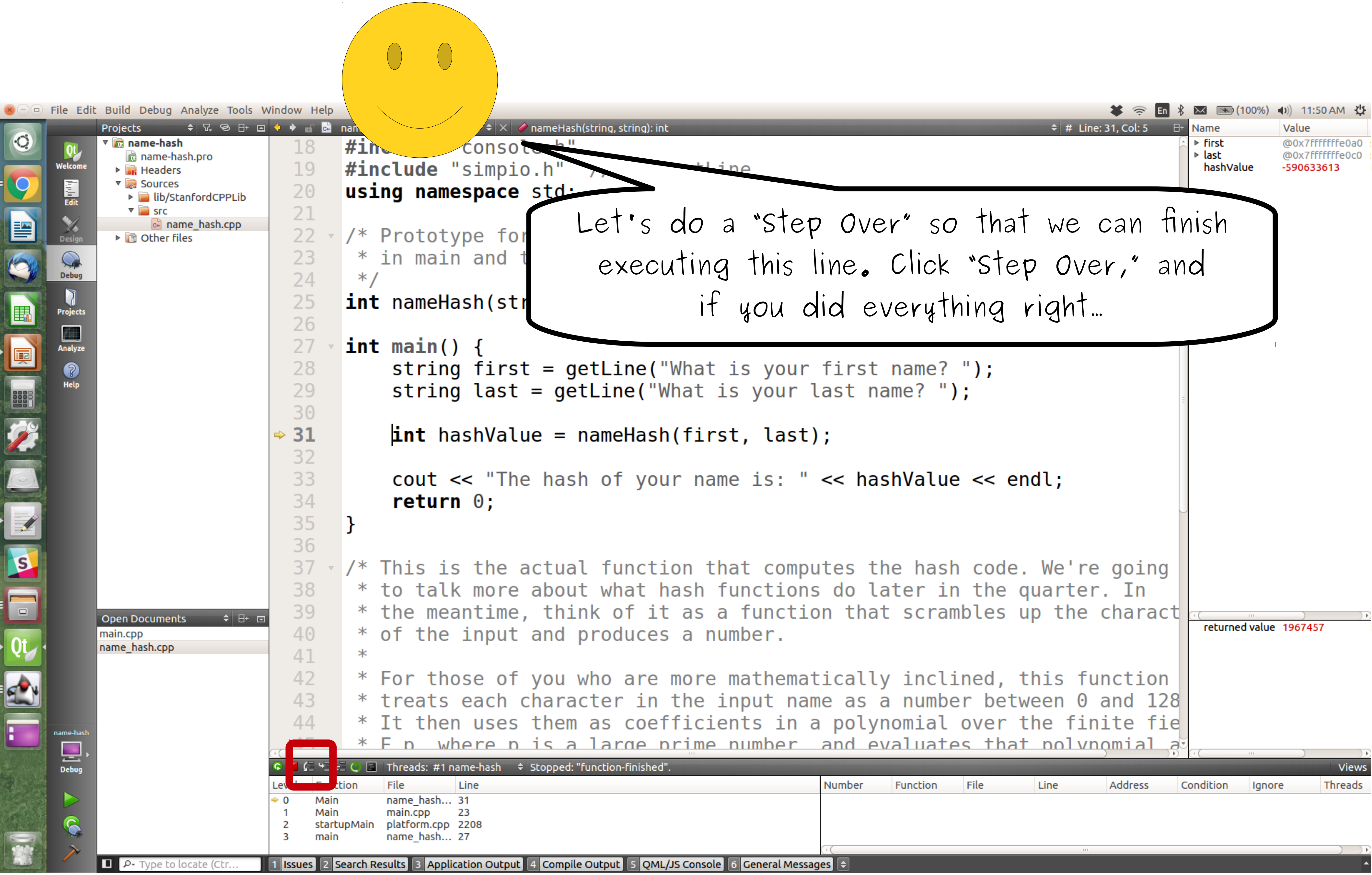
```
#in          console.h"
#include "simpio.h"   //            tline
using namespace std;

/* Prototype for
 * in main and t
 */
int nameHash(str

int main() {
    string first
    string last = g

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
```

Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads



Let's do a "Step Over" so that we can finish executing this line. Click "Step Over," and if you did everything right…

Projects
- name-hash
  - name-hash.pro
  - Headers
  - Sources
    - lib/StanfordCPPLib
    - src
      - name_hash.cpp
  - Other files

Open Documents
- main.cpp
- name_hash.cpp

```
#in        consol   h"
#include "simpio.h"   //           tline
using namespace std;

/* Prototype for
 * in main and t
 */
int nameHash(str

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
```

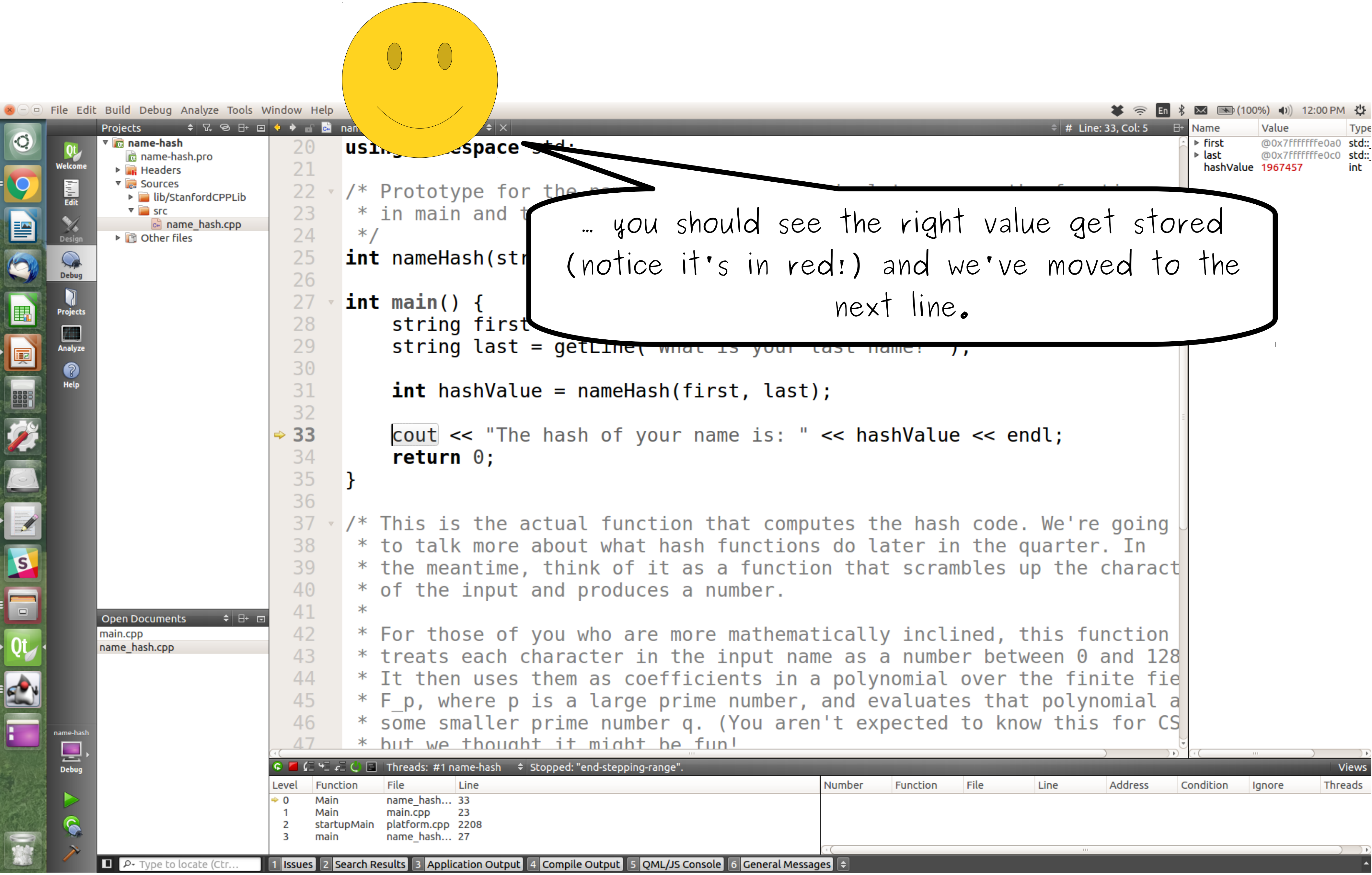
Name	Value
first	@0x7fffffffe0a0
last	@0x7fffffffe0c0
hashValue	-590633613

returned value 1967457

Threads: #1 name-hash   Stopped: "function-finished".

Level	Function	File	Line
0	Main	name_hash...	31
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads



… you should see the right value get stored (notice it's in red!) and we've moved to the next line.

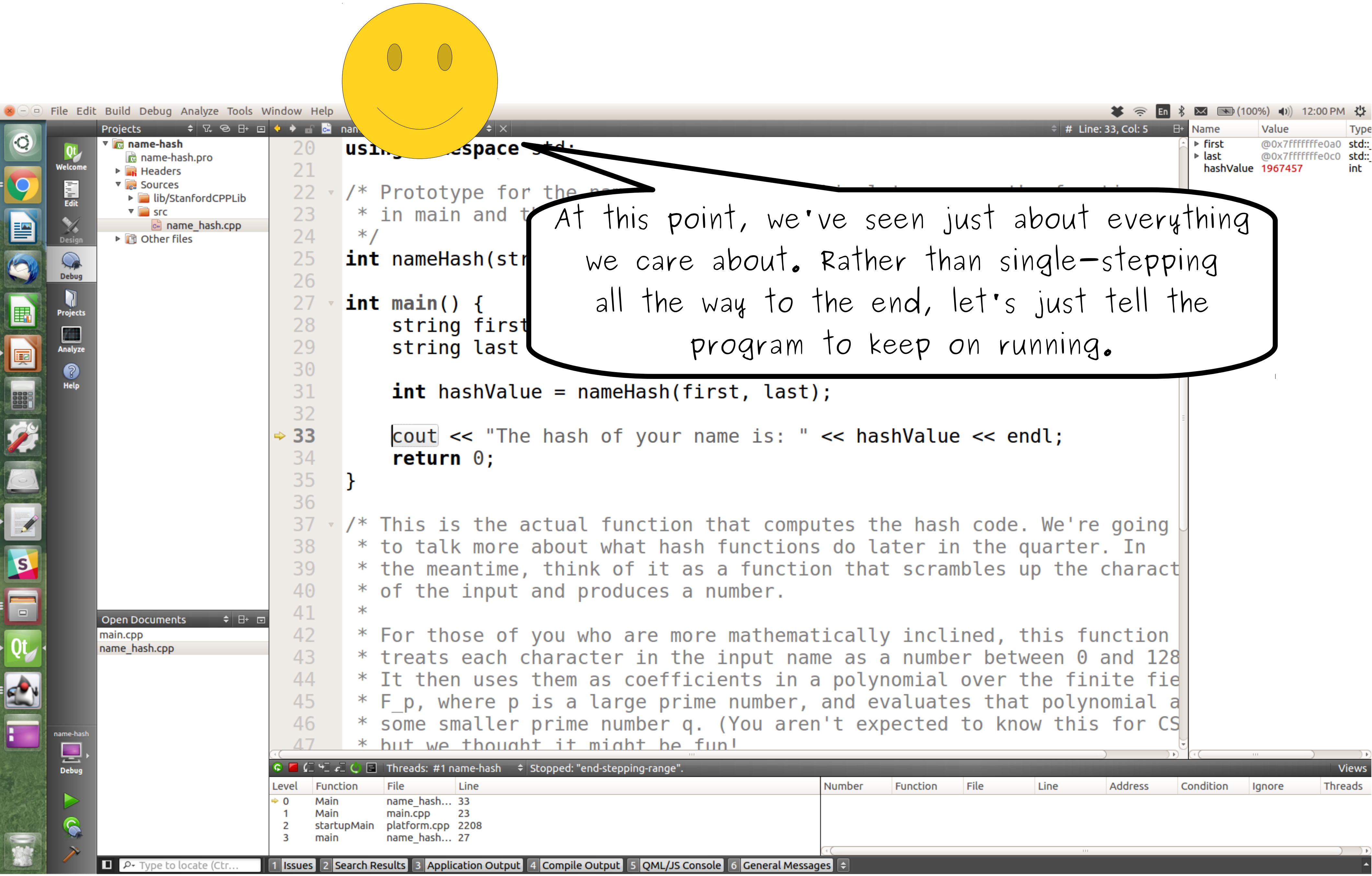
```
usi          espace std;

/* Prototype for the na
 * in main and t
 */
int nameHash(str

int main() {
    string first
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
```

Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	1967457	int

Threads: #1 name-hash   Stopped: "end-stepping-range".

Level	Function	File	Line
0	Main	name_hash...	33
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads



At this point, we've seen just about everything we care about. Rather than single-stepping all the way to the end, let's just tell the program to keep on running.

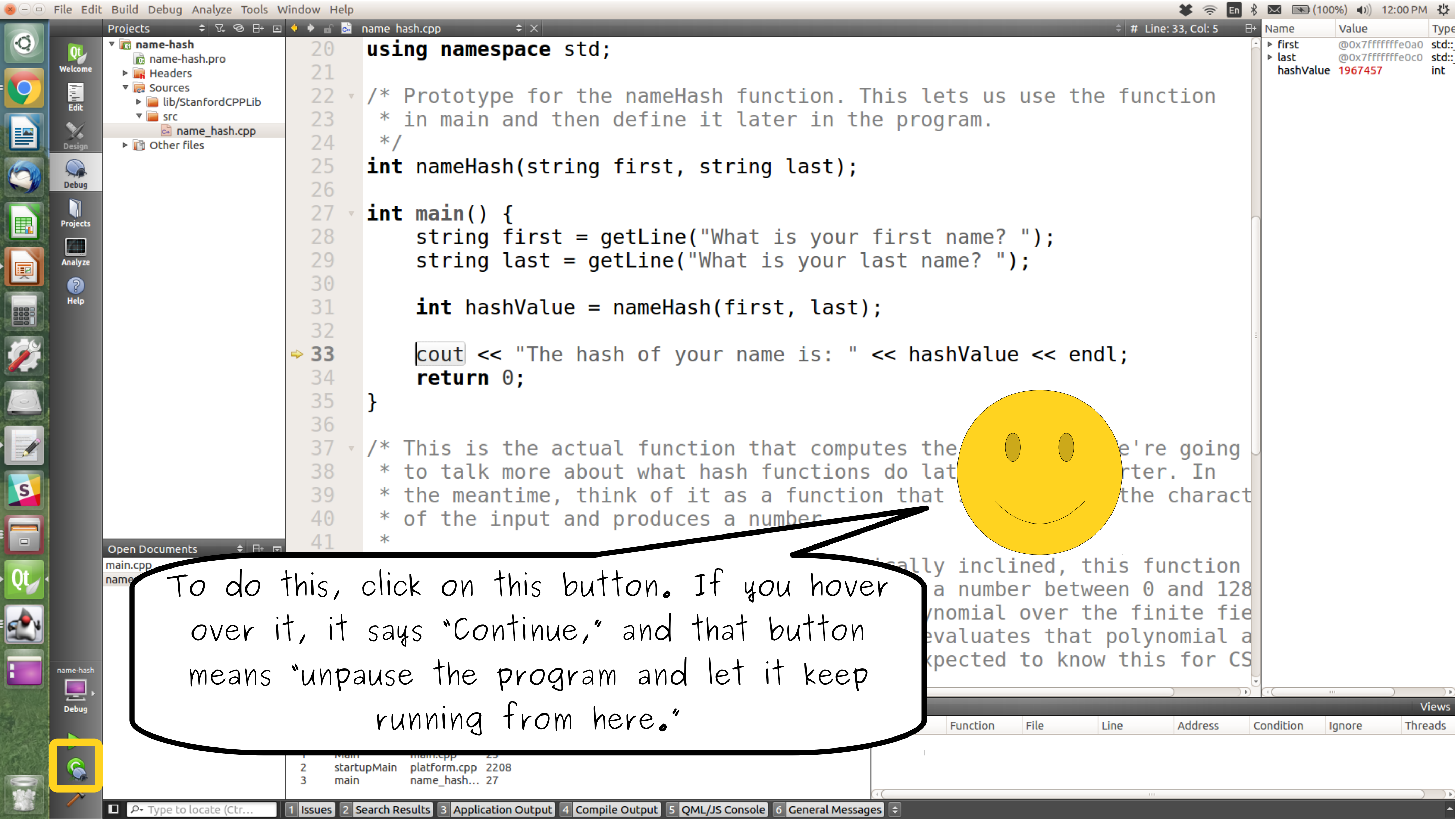
```
usi espace std;

/* Prototype for the na
 * in main and t
 */
int nameHash(str

int main() {
    string first
    string last

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the hash code. We're going
 * to talk more about what hash functions do later in the quarter. In
 * the meantime, think of it as a function that scrambles up the charact
 * of the input and produces a number.
 *
 * For those of you who are more mathematically inclined, this function
 * treats each character in the input name as a number between 0 and 128
 * It then uses them as coefficients in a polynomial over the finite fie
 * F_p, where p is a large prime number, and evaluates that polynomial a
 * some smaller prime number q. (You aren't expected to know this for CS
 * but we thought it might be fun!
```

Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	1967457	int

Threads: #1 name-hash Stopped: "end-stepping-range".

Level	Function	File	Line
0	Main	name_hash...	33
1	Main	main.cpp	23
2	startupMain	platform.cpp	2208
3	main	name_hash...	27

Number	Function	File	Line	Address	Condition	Ignore	Threads

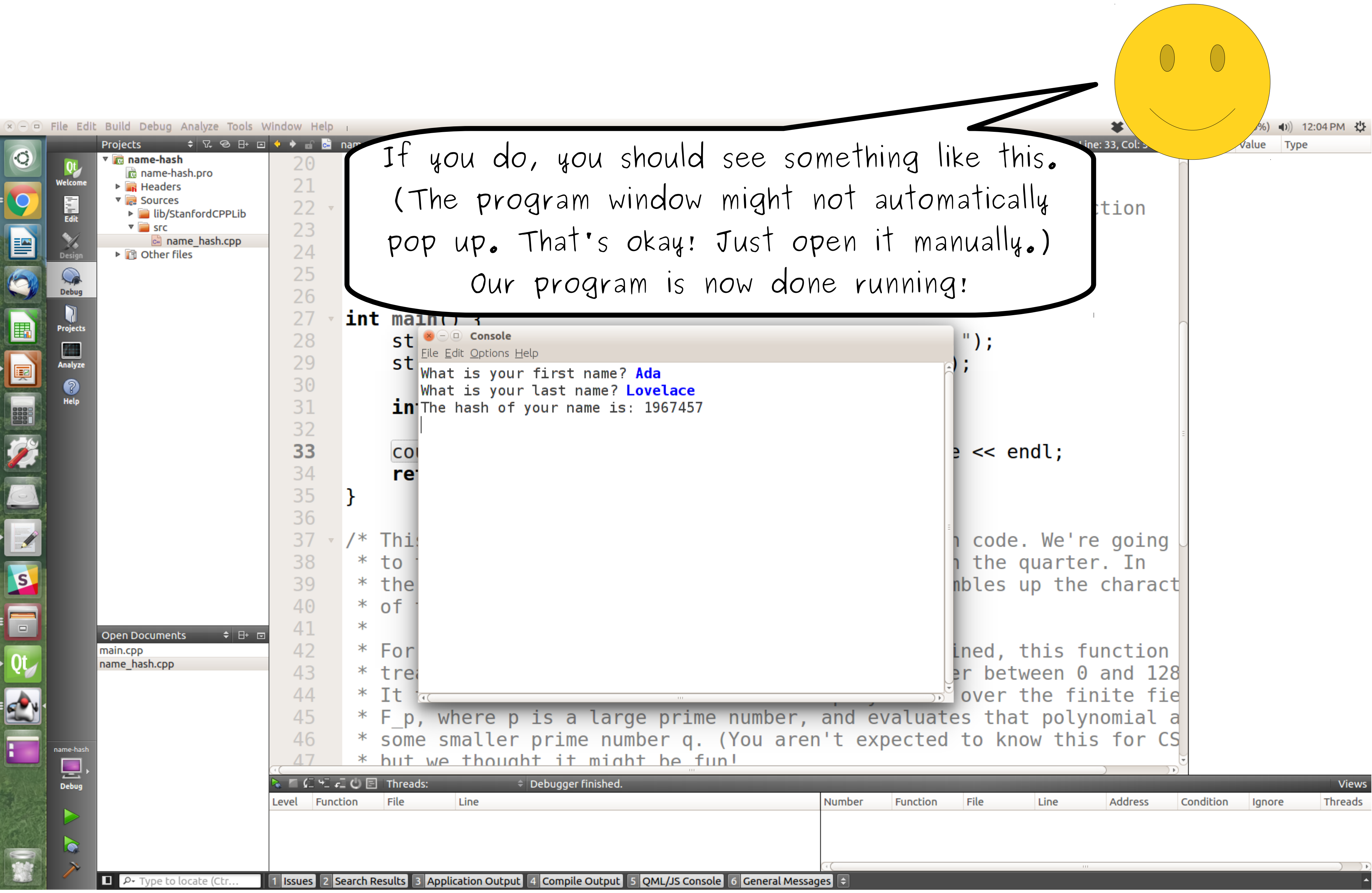


```
using namespace std;

/* Prototype for the nameHash function. This lets us use the function
 * in main and then define it later in the program.
 */
int nameHash(string first, string last);

int main() {
    string first = getLine("What is your first name? ");
    string last = getLine("What is your last name? ");

    int hashValue = nameHash(first, last);

    cout << "The hash of your name is: " << hashValue << endl;
    return 0;
}

/* This is the actual function that computes the               e're going
 * to talk more about what hash functions do lat               rter. In
 * the meantime, think of it as a function that                the charact
 * of the input and produces a number
 *
                                          ically inclined, this function
                                           a number between 0 and 128
                                          ynomial over the finite fie
                                          evaluates that polynomial a
                                          xpected to know this for CS
```

Name	Value	Type
first	@0x7fffffffe0a0	std::_
last	@0x7fffffffe0c0	std::_
hashValue	1967457	int

```
2   startupMain   platform.cpp   2208
3   main          name_hash...   27
```

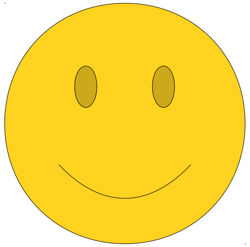
To do this, click on this button. If you hover over it, it says "Continue," and that button means "unpause the program and let it keep running from here."



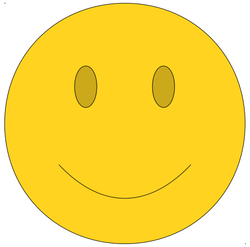
If you do, you should see something like this. (The program window might not automatically pop up. That's okay! Just open it manually.) Our program is now done running!

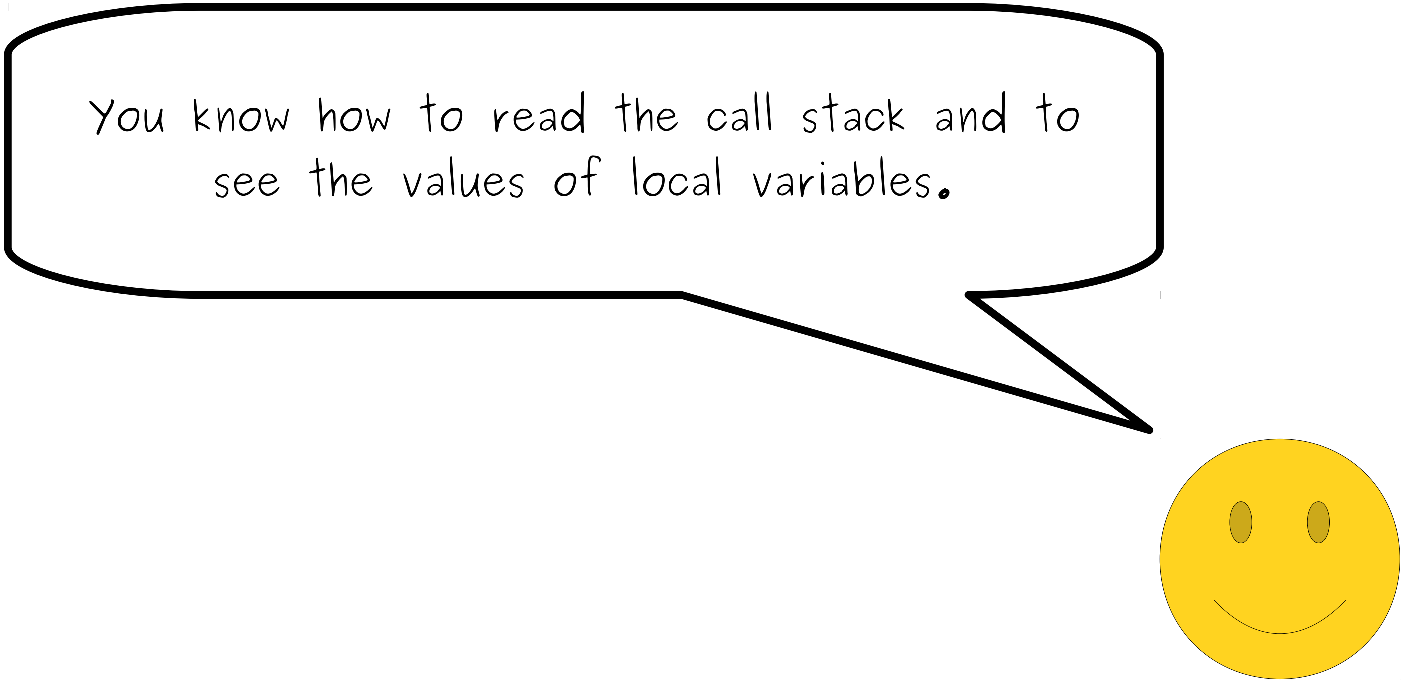
```
What is your first name? Ada
What is your last name? Lovelace
The hash of your name is: 1967457
```

So there you have it! You've now gotten more familiar with the debugger!



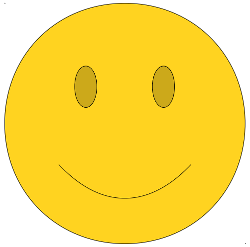
You know how to set a breakpoint to pause the program at a particular point.

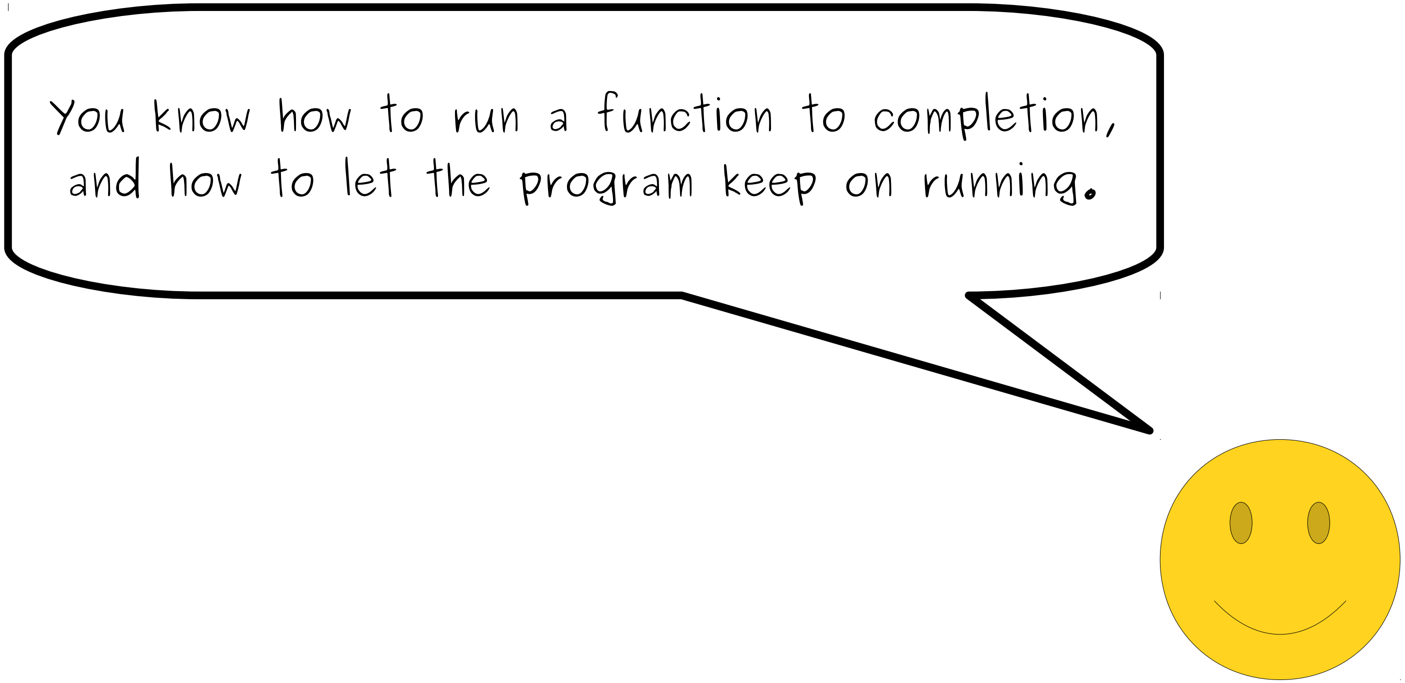




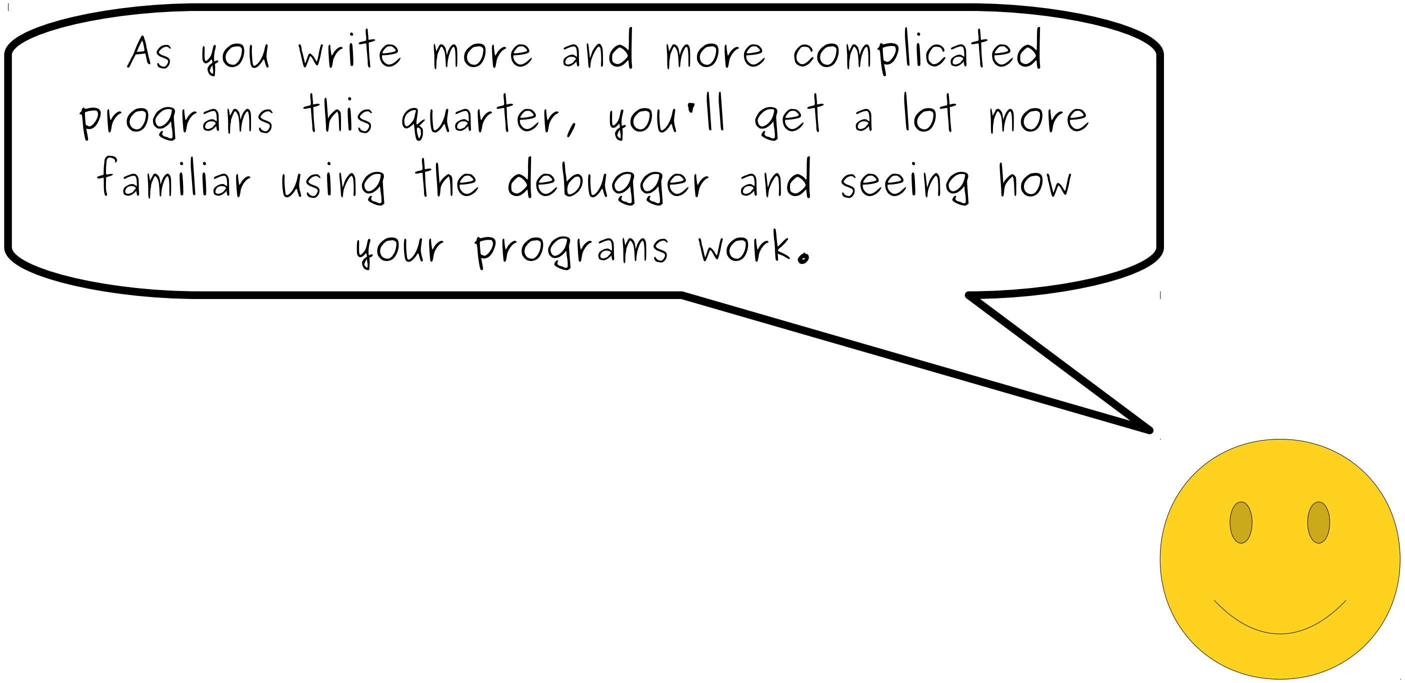
You know how to read the call stack and to see the values of local variables.

You know how to single-step the program and see what values change.

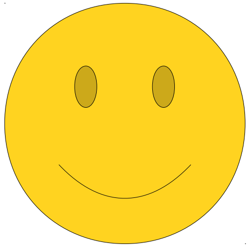




You know how to run a function to completion, and how to let the program keep on running.


As you write more and more complicated programs this quarter, you'll get a lot more familiar using the debugger and seeing how your programs work.

And, if you continue to build larger and larger pieces of software, you'll find that knowing how to use a debugger is a surprisingly valuable skill!



Hope this helps, and welcome to CS106B!