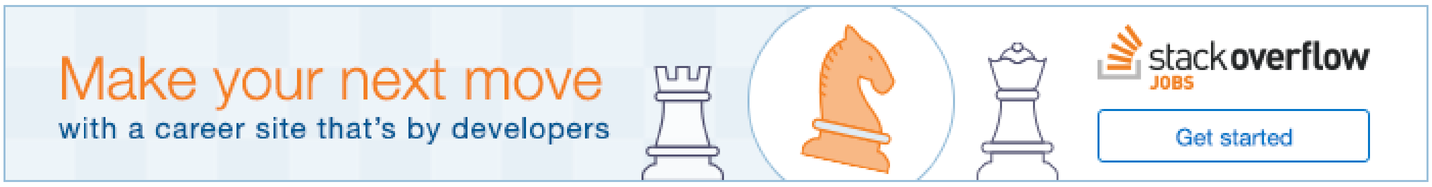
# setjmp/longjmp and local variables

Ask Question



My questions aims at the behaviour of setjmp/longjmp concerning local variables.

18

Example code:

7

```
jmp_buf env;


void abc()
{
  int error;

  ...
  if(error)
    longjmp(env);
}


void xyz() {
  int v1;           // non-volatile; changed between setjmp and longjmp
  int v2;           // non-volatile; not changed between setjmp and longjmp
  volatile int v3;  // volatile;     changed between setjmp and longjmp
  volatile int v4;  // volatile;     not changed between setjmp and longjmp

  ...

  if(setjmp(env)) {
    // error handling
    ...
    return;
  }

  v1++; // change v1
  v3++; // change v3

  abc();
}


int main(...) {
  xyz();
}
```

The documentation of setjmp/longjmp says:

"All accessible objects have values as of the time longjmp() was called, except that the values of objects of automatic storage duration which are local to the function containing the invocation of the corresponding setjmp() which do not have volatile-qualified type and which are changed between the setjmp() invocation and longjmp() call are indeterminate."

I see following two possible interpretations:

intepretation1:

Local variables are restored, except those that are both

- non-volatile and
- changed

intepretation2:

Local variables are restored, except

- those that are non-volatile and
- those that are changed

According to interpretation1 after longjmp only v1 is undefined. v2, v3, v4 are defined. According to interpretation2 after longjmp only v4 is defined. v1, v2, v3 are undefined.

Which one is right?

BTW: I need a general ("portable") answer that is valid for all compilers, i.e. trying out with one particular compiler doesn't help.

c

share improve this question

asked Sep 8 '09 at 11:21
Curd
**8,291** 2 24 41

Implementation note: Variables which are changed and non-volatile may be as they were at the time of longjmp, or may be restored to what they were at the time of setjmp, depending on code generation. Thus 'indeterminate'. So, if they were *not* changed, these two values are the same and that's why unchanged vars are safe. – greggo Apr 13 '12 at 21:27

add a comment

## 2 Answers

active oldest votes

9

Interpretation 1 is correct. If the Interpretation 2 was intended, the original text would have used "**or**which are changed" instead of "and".

share improve this answer

answered Sep 8 '09 at 11:29
caf
**169k** 21 222 353

add a comment

25

setjmp/longjmp is implemented by saving the registers (including stack and code pointers etc) when first passed, and restoring them when jumping.

Automatic (aka "local", stack-allocated) variables that are not 'volatile' ***may*** be stored in registers rather than on the stack.

In these circumstances, the longjmp will restore these registers variables to their value at the point when the setjmp() was first called.

Additionally, a particularly clever compiler might avoid variables that can be inferred from the state of another variable, and calculating them on demand.

However, if the variable is automatic but not been assigned a register, it may be changed by code between the setjmp and the longjmp..

Volatile is explicitly telling the compiler not to store the variable in a register.

So unless you explicitly say a variable is volatile, if you changed the variable between the setjmp/longjmp, its value will depend upon the choices the compiler makes, and is therefore nothing you should rely upon ('indeterminate').

share improve this answer

edited Jun 4 '16 at 12:46
Nubok
**1,336** 3 15 33

answered Sep 8 '09 at 11:29
Will
**43.3k** 26 126 195

And I guess conversely, automatic variables declared as 'register' *may* be stored in registers but the compiler might ignore the hint, so this can't guarantee they will be restored either. – Michael Apr 17 '15 at 17:33

@Michael Actually There is not even a guarantee that anything is "restored". The 'normal' case is that variables will have the value they had, before you called whatever function resulted in the longjmp. In some cases as discussed above, they may be 'clobbered' by the blind restoring of all call-saved registers done by setjmp/longjmp. In practice this means that, if clobbered, they will be probably be restored to the 'setjmp' value, but the language only states that they are indeterminate rather than what you last set them to. – greggoNov 4 '16 at 15:07