

Bounds checked arrays

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2004-02/3096.html

From: jacob navia (jacob_at_jacob.remcomp.fr)

Date: 02/14/04

Date: Sat, 14 Feb 2004 22:27:37 +0100

As everybody knows, the C language lacks a way of specifying bounds checked arrays.

This situation is intolerable for people that know that errors are easy to do, and putting today's powerful microprocessor to do a few instructions more at each array access will not make any difference what speed is concerned.

Not all C applications are real-time apps.

Besides, there are the viruses and other malicious software that are using this problem in the C language to do their dirty work.

Security means that we avoid the consequences of mistakes and expose them as soon as possible.

It would be useful then, if we introduced into C

```
#pragma STDC bounds_checking(ON/OFF)
```

When the state of this toggle is ON, the compiler would accept declarations (like now)

```
int array[2][3];
```

The compiler would emit code that tests each index for a well formed index. Each index runs from zero to n-1, i.e. must be greater than zero and less than "n".

In arrays of dimension "n", the compiler would emit code that tests "n" indices, before using them.

Obviously, optimizations are possible, and good compilers will optimize away many tests specially in loops. This is left unspecified.

Important is to know that the array updates can't overflow in neighboring memory areas.

How many machine instructions does this cost?

Each test is a comparison of an index with a constant value, and a conditional jump. If the compiler only emits forward branches, the branch predictor can correctly predict that in most cases the branch will NOT be taken.

In abstract assembly this is 4 instructions:

```
test if index >= 0
jump if not "indexerror"
test if index < "n"
jump if not "indexerror"
```

where "n" is a compile time constant.

We have something like 4 cycles then, what a 2GHZ machine does in 0,000 000 004 seconds.

Yes, table access is a common operation but it would take millions of those to slow the program a negligible quantity of time. We are not in the PDP-11 any more.

This would make C a little bit easier to program, and the resulting programs of better quality. Buffer overflows happen of course, but the language limits the consequences by enforcing limits.

By default the behavior is to stop the program. The user can override this, and different schemas can be specified by him/her to take actions when a buffer overflow happens.

A simple strategy is to just do nothing.

```
int fn(char *input)
{
    char tmpbuf[BUFSIZ];
    int i=0;
    bool result = false;

    while (*input) {
        tmpbuf[i++] = *input++;
    }
}
```

```

    }
    // Do things with the input
    // set result
    return result;
indexerror:
    return false;
}

```

This function uses the built-in error checking to avoid any bad consequence for an overflow. If the input data is too long, it is a mal-formed input that should be discarded.

This frees the programmer from the tedious task of writing

```
if (i >= sizeof(tmpbuf)) goto indexerror;
```

at EACH array access. This can be done better by a machine and the compiler.

Because a program like that today ***assumes*** the input length can't be bigger than BUFSIZ.

This is always *implicitly* assumed and nowhere *enforced* by the way. The current state implies that catastrophic errors can happen if the index starts overwriting separate memory areas like the return address...

Everyone knows this. Let's do something to stop it. Something simple, without too much fuzz.

In this case the compiler generates code that in case of index error jumps to this label and does what the programmer specifies.

The motto of C is that: Trust the programmer.

We have just to allow him/her to specify what to do in case of overflow.

Trust the programmer doesn't mean that we trust that he never does a mistake of course. It means that the programmer can specify what actions to take in case of error and provide sensible defaults.

comp.lang.c: Bounds checked arrays

Default is then, to finish the program like the `assert()` macro, another useful construct.

Note that this proposal doesn't change anything in the language. No new constructs, even if compilers could provide arrangements like the one proposed above.

I propose then:

```
#pragma STDC bounds_checking(ON/OFF)
```

that should be written outside a function scope.

That's all.

This proposal is an invitation to brain-storming...:-)

I know that anyone using C is aware of this.
So, let's fix it.

jacob