

Re: data types

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2007-03/msg00383.html

- *From:* Flash Gordon <spam@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sat, 03 Mar 2007 08:24:18 +0000
-

koolj96825@xxxxxxxxx wrote, On 03/03/07 07:00:

16-bit. It may be out of date.

Could be.

Yes, I think the manual was not kept up.
sizeof tells me there are 4 bytes to an int.

Just remember, next year it might be 8, or you might get in to embedded work and come across systems with 16 bit chars (you might not want to worry about this, I don't for what I'm currently doing).

Anyway, now that I need to go back over and look closely at my code,
my question is: is there a way to declare a variable say a 16 bit
unsigned integer in C? Or is declaring it "short" the only specifier
that may work?

Why do you need an object of exactly N bits?

I have at least two issues where it matters. In one case, I created a data type which is similar in concept to a bcd (binary coded decimal) for working with degrees-minutes-seconds. I am wasting too much memory per dms structure if they are 32-bit integers. I'll change it to chars.

Depending on what you are doing, you might want to consider (if you are not already using) as an alternative what is generally used where I used to work and generally called with "wrap angles" or BAMs (Binary Angular Measurement, I think). Basically, it was a 16 bit scaled integer where 90 degrees was represented by 0x4000. We generally assumed 2s complement signed numbers, but with hindsight the code would have worked identically if the variables were declared as unsigned 16 bit integers (this was before I found this group and learnt a lot more about C). Unsigned integers are guaranteed to wrap as you expect.

Re: data types

Next, I was using them as keys and since some quirk in the compiler didn't let me use the constant for a max unsigned int, I made my own constant.

If `UINT_MAX` is not declared (or declared incorrectly) through away the implementation and use something else. First make sure that the error was not yours!

> I also expected the number to "wrap around" at that value.

Unsigned integer types are guaranteed by the C standard to wrap.

Another issue is that I used a bit vector to map out used numbers, as a 16-bit key, an acceptably small amount of memory would be used up, but a 32-bit value may be overboard especially since I don't expect more than a few hundred keys to be in my data structure.

Use "unsigned short" if space is a concern and 16 bits is definitely large enough. It will be 16 bits on all the systems you are likely to come across until 128 bit processors come out and desktops by default come with 16GB of RAM.

If your compiler is not a C99 one, as it appears to be, then you'll have to define a 16-bit type yourself. It's not difficult. For example, if, under your implementation, `int` happens be 16 bits then unsigned int is the type you want. You can create an alias like:

```
typedef unsigned int int16;
```

I have now implemented more typedefs for this.

Note that `short` is almost certainly going to be 16 bits on machines with either 16 or 32 bit `int`, so using `short` in your typedef will mean having to change it less often.

But unless you have a specific requirement, I suggest installing a current compiler system. You have many choices for Windows, though I recommend either a version of `gcc`, (Cygwin or MinGW), or Visual Studio Express.

I only program on the hobby level, although I am doing this project for my office, I thus have no budget. So, I am using one of the free compilers. I would love to upgrade but since I do this on the hobby level, I don't know one compiler from another. I will investigate

Re: data types

your suggestions.

They are good suggestions. There are also various IDEs using gcc. I suggest you look at http://clc-wiki.net/wiki/C_resources:Compilers and http://clc-wiki.net/wiki/C_resources:IDEs

Note that mention on one of those pages is not an endorsement, nor is being left off a sign it is bad.

Personally I am using gcc for the commercial SW I work on.

If your implementation does support C99, (and you don't mind partial support), have a look at the types in `stdint.h`. `uint16_t` or `uint_least16_t` may meet your needs.

I will need to google this. Thank you.

If you search the archives of this group you will find references to a fairly portable implementation of `stdint.h` for implementations that do not provide it.

It would be useful if C had `#ifncexists` to check if an include file is available, but it does not.

--

Flash Gordon

.