

Re: #define and (brackets)

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2008-11/msg00785.html>

- *From:* "Alan Carre" <alan@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sat, 29 Nov 2008 00:16:52 +0700
-

"Igor Tandetnik" <itandetnik@xxxxxxxx> wrote in message
[news:%23\\$3ZwcXUJHA.6092@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:%23$3ZwcXUJHA.6092@xxxxxxxxxxxxxxxxxxxxxxxx)

"Alan Carre" <alan@xxxxxxxxxxxxxxxxxxxx> wrote in message
news:eAOBdgWUJHA.5244@xxxxxxxxxxxxxxxxxxxxxxxx

lots and lots and lots of information which I can hardly (pre)process myself...

Ok well, then so it separates everything into these tokens and then pulls them out (or pops them off or whatever) as separate entities which presumably are all separable by spaces correct? At least by stage 7: "White-space characters separating tokens are no longer significant."

... You know, actually I find that a little difficult to comprehend: We all know that white-space characters are significant in the case of consecutive minus-sign tokens. So how can that be? How can it be that there exists a point where we can eliminate all whitespace without changing the program? I mean what about `*pnNum1/ *pnNumber` ? I can't remove that space following the division symbol, the whole program would become a comment...

In any case, what if one's macro was, in fact, `--X` (ie. intentionally decrementing X)? Should the preprocessor separate those minus signs as well? How come they are treated differently? Is the preprocessor aware of C++ syntax? Is it actually compiling when it's deciphering macro expressions?

Personally, I think instead of learning all these so-called rules and standards and so on, I'll just let the compiler teach me what the compiler does. For one thing I KNOW it converts `#define -X` to `--10`. That's fact, end of story. These rules and standards and lofty committees etc etc... are just pissing in the wind.

The ultimate arbiter is the compiler.

– Alan Carre

Re: #define and (brackets)