We need to specify that the initial tape cant contain # bc that’s our delimeter

Testing /\*\*/

1. /\*HelloWorld\*/ - works

2. /\*/\* - fails

3. \*/\*/ -> we arent checking whether this is inside //

4. /\*a\*b\*/ - Works

5./\*a/b\*/ - Works

6./\*a\*b – rejects

7.c=a\*b - Works

8.d=f/g - Works

9. /\*HelloWorld\*/\*/ - rejects

10. /\*/\*\*/ -> this Works bc it’s treating /\* as comment, but in real compilers this should generate a warning

11. /\*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ - Works

12. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ - works

13. /\*/\*/ - Works

14. /\*\*/\*/ - rejects

15. /\*}\*/ - works

16. /\*{\*/ - Works

17. /\*/\*\*/\*/ -> rejects

19. /\*/\*\*\*\*\*\*\*\*d – should reject

20./\*a\*b\* -> reject

21. \* - Works

22. / - Work

23. {\* - reject

24. (/ - reject

25. (\*/)\* - reject (need opening one /\*)

26. {\*/}/ - should reject (need opening one /\*)

27. {/\*\*/}/ - works

28. (/\*\*/)\* - works

29. (/\*) – reject - ) is ignored in /\*

30. (/\*\*\*\*\*\*)\* - reject

31. (/\*\*\*\*\*)\*/ - reject

32. (/\*\*\*\*\*)\*/\* - reject

32. (/\*\*\*\*\*)\*/\*) – accept

32. (/\*\*\*\*\*)\*/)\* - accept

30. (/\*)\*/ - reject – ignore ) in comment

31. (/\*)\*/)\* – works

32. (/\*)g5\*\*\*/}”” – reject, rejects as son as } doenst match )

/(/) - accepts

/(\*) – accepts

\*(/) – accepts

/[] – accepts

\*[/] – accepts

\*{/} – accepts

/(\*) – accepts

\*(\*) - accepts

\*[/} - rejects

\*”” – accepts

/’’ -> rejects bc empty character is not allowed in C

\*’’ -> rejects bc empty character is not allowed in C

/”” – accepts

X, space, and # are reserved… so dont use them in the tape

Testing “”

“” – Works

“”” - rejects

"Hello\"" -> success

"Hello\"""-> rejects

"Hello\"awr"" -> rejects

"Hello\"awr" -> success

"Hello\"awr\"" ->success

"Hello\"awr\" -> rejects

"Hello\"yyy\"" -> success

"Hello\"""h" -> success

“{“{[/\*}\*/]} -> success

“{“{[/\*/\*(}[\*/]} -> success

“{“{[/\*/\*\*/]} -> success

“{“{(/\*/\*(}[\*/]} -> rejects

Single character

‘a’ -> work

‘aa’ -> rejects

‘\’’ -> work

‘\’’’ -> rejects

‘\’’’’ -> rejects – bc of empty character

‘’ -> not allowing empty string -> C doesnt

‘\*’ – accepts

‘/\*’ – rejects

/\*'/\*'\*/ - accepts

/\*'/\*'\*/{}[]"(" – accepts

/\*(/\*\*\*\*\*)\*/\*\*/ -accepts

"(/\*\*\*\*\*)\*/\*" – accepts

‘(/\*\*\*\*\*)\*/\*’ – rejects

"'(/\*\*\*\*\*)\*/\*'" - accepts

v[i]=a[i][j];

for(;;)break;

{x=7};

for(i=0;i<v.size();i++)v[i]=2;

v[i]=a[i,

for(;;

okay, so could u say that anytime you have two '' followed by each other that's invalid syntax? either bc that could mean we are trying to print an empty character or because we are trying to print teo characters one followed by the other?

ChatGPT

ChatGPT

Yes, you are correct. In C programming, anytime you see two single quotes '' immediately followed by each other, it represents invalid syntax. This can be invalid for two primary reasons:

Empty Character Constant: If it's just two single quotes with nothing between them (''), it's an attempt to create an empty character constant, which is not allowed in C. Character constants in C must contain exactly one character.

Two Characters Without Separation: If it's an attempt to put two characters consecutively within single quotes (like 'ab' or 'a''b'), it's also invalid. C requires that a character constant must contain only a single character. To represent multiple characters, you would typically use a string (enclosed in double quotes, like "ab"), or handle them as separate character constants ('a' and 'b').

Both scenarios you described fall into the category of invalid syntax in C. The correct approach depends on what exactly you are trying to achieve with the code.

okay, the only exception to that is when you have a escape character right? '\'' is there nay other exception?

ChatGPT

ChatGPT

Yes, you are correct. The use of escape characters is a key exception to the rule about single characters within single quotes in C. Escape sequences allow you to include special characters in a character constant that would otherwise be impossible to express directly.

The sequence '\'' is an example of this. It represents a single quote character ('). The backslash (\) is used as an escape character to indicate that the following single quote is part of the character constant, not the end of it.

Here are some common escape sequences in C:

Newline: '\n' - Inserts a newline.

Tab: '\t' - Inserts a horizontal tab.

Backslash: \\ - Represents a backslash.

Double Quote: \" - Represents a double quote (often used in strings).

Single Quote: '\'' - Represents a single quote (as in your example).

These are standard escape sequences. The general rule is that if you want to include a character in a character constant that has a special meaning in C (like a single quote, double quote, or a backslash), you need to precede it with a backslash to "escape" its special meaning.

Other than escape sequences, there are no exceptions to the rule that a character constant in C is limited to a single character. The combination of an escape character with another character is still considered a single character in terms of how C processes character constants.