From: Frank Farance
Organization: Farance Inc.
Telephone: +1 212 486 4700
Fax: +1 212 759 1605
E-mail: frank@farance.com
Date: 1995-08-25
Document Number: WG14/N458 X3J11/95-059
Subject: Extended Characters Analysis

1. PROBLEMS

This paper summarizes several of the current activities in WG14 and related standards.

1.1 Extended Identifiers

There has been strong interest in adding support for identifiers that include international characters. Currently, C limits identifiers to the characters of the Latin alphabet, the digit 0-9, and underscore.

There are two approaches solving this problem. The first involves requiring C translators to support all (or a majority) encodings of character sets. This would require supporting many individual sets or a single set that includes all characters (e.g., ISO 10646). Regardless, this requirement seems too much to impose on *all* compiler vendors.

The second approach involves a ''standard'' C encoding of some of the characters, e.g., their Unicode name, but represented in C's basic character set (e.g., "\U1234" represents the Unicode character 1234). For example, an identifier might be:

```
int abc\U1234def = 1;
```

This would require very little effort compiler vendors to add, yet high quality text editors would write the international characters in this form rather than the native text form.

1.2 Extended Character Literals

Along with identifiers, it would be convenient for programmers to include this in their programs. For example, the extended character would be included in the string literal:

```
wchar_t x[] = L"abc\U1234def";
```

1.3 Character Programming

Many people have been searching for a more generic way of programming with characters and strings. For example, C has been and still is strongly tied to 7-bit character systems.

In C, the distinction between a character and a byte (smallest addressable unit of storage) is blurred. Some programmers use "char" because they mean ''bytes'' and others use "char" because they mean ''characters''.

2. EXISTING PRACTICE

2.1 Features of C

C provides two types of characters: plain and wide. Although there is no guarantee, the wide character provides the capability to store more values (i.e., range) than the plain character.

C provides two sets of libraries for plain and wide characters. For wide characters, C distinguishes between wide character strings (useful for processing similar to plain characters) and multibyte character strings (used for encoding wide character strings in files). It should be noted that a wide character string is a special type of multibyte character string, i.e., the limiting case. Typically, wide character strings take up more space, but are faster to process; multibyte character strings take up less space and require more effort to process.

C provides two diagraphs and trigraphs for accessing C language characters that are not in ISO 646. Diagraphs are a ''prettier'' way of spelling certain punctators. Diagraphs are never used in strings. Trigraphs provide textual substitution of the desirer character, i.e., they can be used in expressions and in string literals.

C provides two ways of specifying a character: by name or by value. For example, "\n" specifies the newline character by name -- it is independent of locale. "\x0a" specifies the character by value -- its encoding is the same regardless of locale. The character "\0" is a character specified by both name and value.

Typically, name characters are used in text files where the encoding is ''local'' and there are tools (e.g., FTP) used to exchange the file with a ''global'' format. Name characters can be used in binary files, but the programmer shouldn't expect to exchange the files with other systems *and have the same meaning*.

Value characters are used in binary files for a ''global'' format. For example, the string "\x31\x32\x33" would represent the ASCII string "123" regardless of locale.

2.2 Related Standards

There are several standards for *encoding* characters: ASCII, EBCDIC (several encodings), ISO 646, ISO 8859-1, and ISO 10646.

There has been much interest in ISO 10646 because it is being sold as a universal character set. ISO 10646 specifies an encoding of characters (specific mapping of names to values). The following are some of the encodings:

> 2-octet encoding. This might be suitable as a wide character encoding for C. However, this doesn't include all of the characters, only the popular ones they've defined. Even so, the 2-octet encoding may require 4-octets for certain character extensions they are planning. The long term solution is that characters will require more than 2 octets for encoding.
>
> 4-octet encoding. This might be suitable as a wide character encoding for C. This is simply the 4-octet encoding of the character. However, this doesn't include composite characters (similar to overstrikes) for certain levels of ISO 10646.
>
> 1,2,3,5-octet encoding. This encoding translates the 4-octet encoding into a non-control code encoding suitable for transmission in 8-bit transmission systems. The encoding varies in length for each character. This encoding is not suitable as a wide character encoding, but possible useful for a multibyte character set encoding.

There are no standards for the names of characters. Although ISO 10646 has descriptive names (e.g., ''LATIN LETTER CAPITAL A''), the committee doesn't see these has standardized names. The names of the characters change in the French version of the ISO standard.

3. POTENTIAL SOLUTIONS

3.1 Adding A New Character Type

This has been proposed by several people for C9X. The following considerations would direct the new type:

> - The type must store greater than 4 octets. 4 octets is not enough.
>
> - The type must be efficient for programming.
>
> - The type should be as small as possible to minimize storage.

This would suggest that that type would be 5 octets (smallest storage) or 6 or 8 octet (fastest processing). Like the extended integer proposals, the problem with a new type is the same:

> - What is the performance of the new type?

- How do I tune the performance.

My guess is that a single universal character type is impractical in for the same reasons a single universal integer is impractical.

3.2 Adding Prefix For Extended Identifiers

This seems like the easiest feature to add, but how would the character be identified? By name or by value? Probably, a name character would be useful since programs are text files. Regardless, which name or value convention (encoding) does C standardize?

3.3 Adding Prefix For Extended Characters

Like extended identifiers, we would need to refer to the character. Since character and string literals are used for both binary and text files, both name and value characters should be supported. Again, the question is what convention should C standardize?

3.4 Library

If a new character type is added, a new library would need to be created. This is probably too many names to add to C9X. Short of adding overloaded functions, I don't see a solution here.

3.5 Minimum Portability

Even if C9X includes extended characters, this still doesn't allow the programmer to write portable programs using the extended characters because the "wchar_t" type varies in precision from system to system.

Also, the availability and encoding of characters varies from system to system. It still isn't clear what the minimum portability is.

4. CONCLUSIONS

It is too early to standardize these features now. While there appears to be small features we can add (e.g., extended identifiers), I strongly recommend that WG14 consider solving the larger questions first. Probably the more important question is ''what is the minimum portable program we should support''.