From: Frank Farance
Organization: Farance Inc.
Telephone: +1 212 486 4700
Fax: +1 212 759 1605
E-mail: frank@farance.com
Date: 1995-12-22
Document Number: WG14/N532 X3J11/95-133
Subject: Extended Identifiers

Since the 1995-10 WG14 meeting in Nashua, the extended character problem has been divided into two parts: (1) extended identifiers in C programs, (2) extended characters in string literals. The companion paper is WG14/N533. This paper identifies several outstanding issues. We must resolve these issues prior to proposing syntax.

WHAT IS AN IDENTIFIER?

The problem with extended identifiers has discussed by several people in WG14, WG20, and WG21. C and C++ should be compatible in the extensions they choose. There seems to be agreement on what constitutes an identifier across all character sets. Using ISO 10646 terminology, the following characters are considered identifiers (16-bit hex values):

Latin: 0041-005a,0061-007a,00c0-00d6,00d8-00f6,00f8-01f5,01fa-0217, 0250-02a8,1e00-1e9a,1ea0-1ef9
Greek: 0384,0388-038a,038c,038e-03a1,03a3-03ce,03d0-03d6,03da,03dc,03de, 03e0,03e2-03f3, 1f00-1f15,1f18-1f1d,1f20-1f45,1f48-1f4d,1f50-1f57,1f59,1f5b,1f5d, 1f5f-1f7d,1f80-1fb4,1fb6-1fbc,1fc2-1fc4,1fc6-1fcc,1fd0-1fd3, 1fd6-1fdb,1fe0-1fec,1ff2-1ff4,1ff6-1ffc,
Cyrilic: 0401-040d,040f-044f,0451-045c,045e-0481,0490-04c4,04c7-04c8, 04cb-04cc,04d0-04eb,04ee-04f5,04f8-04f9
Armenian: 0531-0556,0561-0587
Hebrew: 05d0-05ea,05f0-05f4
Arabic: 0621-063a,0640-0652,0670-06b7,06ba-06be,06c0-06ce,06e5-06e7,
Devanagari: 0905-0939,0958-0962
Bengali: 0985-098c,098f-0990,0993-09a8,09aa-09b0,09b2,09b6-09b9, 09dc-09dd,09df-09e1,09f0-09f1
Gurmukhi: 0a05-0a0a,0a0f-0a10,0a13-0a28,0a2a-0a30,0a32-0a33, 0a35-0a36,0a38-0a39,0a59-0a5c,0a5e
Gujarati: 0a85-0a8b,0a8d,0a8f-0a91,0a93-0aa8,0aaa-0ab0,0ab2-0ab3, 0ab5-0ab9,0ae0,
Oriya: 0b05-0b0c,0b0f-0b10,0b13-0b28,0b2a-0b30,0b32-0b33,0b36-0b39, 0b5c-0b5d,0b5f-0b61,
Tamil: 0b85-0b8a,0b8e-0b90,0b92-0b95,0b99-0b9a,0b9c,0b9e-0b9f,0ba3-0ba4, 0ba8-0baa,0bae-0bb5,0bb7-0bb9,
Telugu: 0c05-0c0c,0c0e-0c10,0c12-0c28,0c2a-0c33,0c35-0c39,0c60-0c61,
Kannada: 0c85-0c8c,0c8e-0c90,0c92-0ca8,0caa-

0cb3,0cb5-0cb9,0ce0-0ce1,
Malayalam: 0d05-0d0c,0d0e-0d10,0d12-0d28,0d2a-0d39,0d60-0d61,
Thai: 0e01-0e30,0e32-0e33,0e40-0e46,0e4f-0e5b,
Lao: 0e81-0e82,0e84,0e87,0e88,0e8a,0e0d,0e94-0e97,0e99-0e9f,0ea1-0ea3, 0ea5,0ea7,0eaa,0eab,0ead-0eb0,0eb2,0eb3,0ebd,0ec0-0ec4,0ec6,
Georgian: 10a0-10c5,10d0-10f6,
Hiragana: 3041-3094,309b-309e
Katakana: 30a1-30fe,
Bopmofo: 3105-312c,
Hangul: 1100-1159,1161-11a2,11a8-11f9
CJK Unified Ideographs: f900-fa2d, fb1f-fb36,fb38-fb3c,fb3e,fb40-fb41,fb42-fb44,fb46-fbb1,fbd3-fd3f, fd50-fd8f,fd92-fdc7,fdf0-fdfb,fe70-fe72,fe74,5e76-fefc, ff21-ff3a,ff41-ff5a,ff66-ffbe,ffc2-ffc7,ffca-ffcf,ffd2-ffd7, ffda-ffdc,4e00-9fa5

ISSUE: We need to refer to a specific, published (i.e., someone can purchase it) document that contained the range information above.

These characters may be included in the source code using the trigraph notation for ISO 10646 16-bit characters ("x" is a hex digit):

??uxxxx

and the trigraph notation for ISO 10646 32-bit characters:

??Uxxxxxxxx

NOTE: This does not imply the use of any ISO 10646 encoding of the source code. This notation only identifies the *name* of the character, not its encoding.

This leaves us with the question of how do we translate a C (or C++) program that includes these characters? There are four competing models of interpretation (provided by Tom Plum):

MODEL #1: In phase 1, a program is translated (conceptually) into the minimal basic source character set of the host compiler; i.e. the minimal C/C++ ASCII characters, or a character set that contains a one-to-one mapping of those characters (e.g. an EBCDIC mapping). Any character not in the minimal basic source character set is converted into a specific ''canonical multibyte encoding'' which uses double-question-mark. Any double-question-mark code in the original source is preserved unchanged. (In all three models, it is an error, diagnostic-mandatory, if the source contains a double-question-mark code that stands for one of the minimal basic source characters. For example, there

is only one allowable encoding for TAB, or SPACE, or ''lowercase latin a''.) There are no shift sequences in the output of phase 1, because all non-basic characters have been replaced by double-question-mark codes. The lexical grammar for ''identifier'' doesn't need to refer to any characters that aren't in the minimal basic source character set, because all the extended characters are represented as double-question-mark codes.

MODEL #2: In phase 1, a program is translated (conceptually) into the (full) multibyte source character set of the host compiler. Any character not in the source character set is converted into a specific ''canonical multibyte encoding'' which uses double-question-mark. Any double-question-mark code that maps into a character in the (full) source character set is mapped into that character. Any redundant shift sequences are (at the proper time) eliminated. The lexical grammar for ''identifier'' includes a non-terminal for ''other source characters''.

MODEL #3: In phase 1, a program is translated (conceptually) into a widechar character set which is large enough to encode every 10646 character. (Or just say that the output of phase 1 actually IS 10646, conceptually.) (It is effectively a requirement of model 1, 2, or 3 that every character of the source character set of the host compiler must map into some 10646 character.) In model 3, there are no characters not in the (augmented) source character set, because 10646 encodes all of them. Any double-question-mark code in the physical source is translated into the corresponding widechar. There are no shift sequences in the output of phase 1, because the entire program consists at this point of widechars.

MODEL #4: All "??" sequences are translated, if possible, into a character in the source character set. All non-translatable sequences are left as "??" sequences. An identifier contains the usual characters plus any identified by the WG20 paper. The compiler vendor would know which of *their* characters corresponded to identifier characters AND they'd know which ''foreign'' characters were included by comparing ranges.

ISSUE: We need to resolve the conceptul model at the next meeting.

WHAT IS THE MAXIMUM LENGTH OF AN IDENTIFIER?

With encoding identifiers as "??U12345678", the programmer might be severely limited in the portable programs you could

write. This is because the linker may only support 31 characters in external identifiers. Even using the best encoding (i.e., variable length multibyte), this would limit the programmer to about 6 characters per extenal identifier. This isn't a problem for C++ since there is no translation limit for external identifiers. However, this is a problem for C.

ISSUE: What limit should be required for C translators?

HOW DO I WRITE A TEXT EDITOR FOR C PROGRAMS

There are three significant applications that characterized the C programming language: (1) the ''hello world'' program, (2) a text editor for writing C progams, (3) the UNIX operating system implemented in C. Although we have identified a syntax for supporting extended identifiers, we have no mechanism for handling the characters of the C source code.

ISSUE: What character I/O should we support? (See WG14/N533, ''Extended String Literals''.)