

c2z

Parser

## **Notice**

Information in this documentation is subject to change without notice.

This software is for private use. No commercial use is permitted without the express written permission of an authorized representative of the author of this software.

Other product names used herein are for identification purposes only, and may be trademarks of their respective companies.

**TCCS**

**Copyright 2015 - 2021**

**All Rights Reserved**

# Table of Contents

Notice.....	2
General Information.....	5
Need To Know.....	6
Command line execution.....	7
Data definitions.....	8
c2z Functions.....	9
c2z .....	11
#define.....	11
:.....	11
atoi.....	11
break.....	11
case.....	11
char.....	11
double.....	11
enum.....	11
exit.....	12
fclose.....	12
fflush.....	12
fgets.....	13
fopen.....	13
for.....	13
fprintf.....	13
fputs.....	13
free.....	13
gets.....	13
goto.....	13
if - else.....	13
int.....	14
isalnum.....	14
isalpha.....	14
isdigit.....	15
isspace.....	16
isupper.....	16
localtime.....	16
memcpy.....	16
memmove.....	17
printf.....	17
puts.....	17
scanf.....	17
sizeof.....	18
snprintf.....	18
streat.....	18
strcmp.....	18
strcpy.....	18
strlen.....	18
strncpy.....	18

strev.....	19
strset.....	19
switch.....	19
time.....	19
t_time.....	20
tolower.....	20
toupper.....	20
while.....	20
Math .....	21
Addition.....	21
Subtraction.....	21
Multiplication.....	21
Division.....	21
String .....	22
Example 1:.....	22
Example 2:.....	22
Array – Integer.....	23
Declaring Arrays.....	23
Initializing Arrays.....	23
Accessing Array Elements.....	23
.....	24
Array – Character.....	25
Structures.....	26
Includes.....	27
Non Processed Functions.....	28
Example Program.....	32
Programs.....	37
Functions.....	38
Test Programs.....	41
ctest_1.c.....	41
ctest_2.c .....	41
ctest_3.c .....	41
ctest_8.c.....	41
ctest_9.c.....	41
ctest_10.c.....	41
ctest_11.c.....	41
ctest_12.c.....	41
Internals.....	43

## General Information

The attempt of this program/document is to explain what I am experimenting with. There is a "GCC" for various IBM systems, but I found them difficult, if not impossible to use.

Several years ago, I help write a "P-code" Basic compiler/interpreter, So, why not a "C" Implementation parser!!!

I took the approach of a "front-end" to convert "C" to Z390 MLC assembler. Then feed the MLC file to the Z390 assembler.

The design of c2z is based on information found in the book, Compiler Design in C, by Allen I. Holub, 1990, Prentice Hall Software Series.

The parser program c2z (pronounced c to z) runs under Linux.

The following individuals have provided time,support and understanding on working on this project:

Abe Kornelis  
Melvyn Maltz  
James Cray

## Need To Know

Listed below are the differences/changes that will affect your C program converted to IBM assembler.

All variables are set to zero or blanks.

printf is mapped to console.

fprintf is mapped to a printer file.

Any MLC variable that starts with C370xxxx is an internal c2z variable.

C field names often exceed 8 characters. c2z has a translation table that takes the long C name and generates an internal parser field name that will not exceed 8 characters. These fields start with "C37xxxx" and used by the parser. Dump the stats.txt file to see this table.

## Command line execution

c2z pg-name flag

c2z            This is the parser.

Pg-name      The name of the file which contains the  
C source code. The .C extension is NOT  
required. Can be upper or lower case.  
ex:            c2 HELLO

flag          The only flag defined at this time is the -d. This is a debug  
flag. It has three levels, increasing in output as the level  
is increases.

-d1      Input lines only  
-d2      Adds function calls  
-d3      Adds subroutines calls

**Data definitions**

"C"	ASM	Type
int	PL6	I
char	C	C
double	D	D



## c2z Functions

Status of "C" functions

### **Name**

atoi  
break  
case  
char  
ctime  
double  
enum  
exit  
feof  
fclose  
fflush  
fgets  
fopen  
for  
fputs  
fprintf  
free  
gets  
goto  
if  
Int  
isalpha  
isdigit  
isspace  
isupper  
localtime  
memcpy  
memmove  
printf  
puts  
scanf  
sizeof  
snprintf  
sprintf  
strcat  
strchr

strchr  
strcmp  
strcpy  
strlen  
strncpy  
switch  
time  
time\_t  
tolower  
toupper  
while

## **Math**

addition  
subtract  
multiplication  
division

## **Subroutines**

## c2z

**#define**

:

**atoi**

**break**

**case**

**char**

Defines a character field. c2z converts to a DC CLx'xxxxxx'.

Ex:

```
char string[5];
```

Converts to:

```
STRING DC CL5'STRING'
```

**double**

**enum**

*enum* is the abbreviation for ENUMERATE, and we can use this keyword to declare and initialize a sequence of integer constants. Here's an example:

```
enum colors (red, yellow green, blue);
```

Here, *colors* is the name given to the set of constants - the name is optional. Now, if you don't assign a value to a constant, the default value for the first one in the list - *RED* in our case, has the value of 0. The rest of the undefined constants have a value 1 more than the one before, so in our case, *YELLOW* is 1, *GREEN* is 2 and *BLUE* is 3.

But you can assign values if you wanted to:

```
enum colors (red=1, yellow, green=6, blue);
```

Now RED=1, YELLOW=2, GREEN=6 and BLUE=7.

The main advantage of *enum* is that if you don't initialize your constants, each one would have a unique value. The first would be zero and the rest would then count upwards.

```
#include <stdio.h>

int main() {
    enum {RED=5, YELLOW, GREEN=4, BLUE};

    printf("RED = %d\n", RED);
    printf("YELLOW = %d\n", YELLOW);
    printf("GREEN = %d\n", GREEN);
    printf("BLUE = %d\n", BLUE);
    return 0;
}
```

This will produce following results

RED = 5

YELLOW = 6

GREEN = 4

BLUE = 5

## **exit**

Terminates program.

## **fclose**

Closes a stream. fclose closes the named stream.

```
fclose(flog);
```

## **fflush**

This is for software compatibility ONLY. Performs no activity in Z390.

## **fgets**

Gets a string from a stream.

**fgets** reads characters from stream into the string. The function stops reading when it reads either n -1 characters or a newline character, whichever comes first.

## **fopen**

Opens a stream.

Ex: `c_input = fopen(filename, "r");`

## **for**

## **fprintf**

(Mapped to a file name list.txt{z390})

```
fprintf(flog,"this a test to the printer\n");  
fprintf(flog,"My age is = %d\n", age);  
fprintf(flog,"My name is - %s\n", name);
```

## **fputs**

Outputs a string on a stream.

## **free**

C2z performs a free somewhat different than the standard. Free in c2z clears character fields to blanks and numeric fields to zeros.

## **gets**

## **goto**

## **if - else**

```
if(field1 operand field2)  
{
```

```

        . . . .
    }
else
    {
        . . . .
    }

```

field1 and field2 can be an numeric variable or a numeric constant.

Numeric variable

ex:     int ct

Numeric Constant

ex:     1,55,1234 . . . .

operand allowed:

==	field1 equal to field2
!=	field1 not equal field2
>	field1 greater than field2
<	field1 less than field2
>=	field1 greater than or equal to field2
<=	field1 less than or equal to field2

## int

Defines an integer field. c2z converts the variable to a PL4'0'.

Ex:     int x;

Converts to:

X     DC   PL4'0'

## isalnum

**isalnum** classifies ASCII-coded integer values by comparison. Isalnum returns nonzero if c is a a letter (A-Z or a-z) or a digit (0-9).

## isalpha

Test a single character inclusive A through Z. If returns nonzero (1) if the character is a letter (A-Z). Returns a zero (0) if not.

isalpha can be used in IF and WHILE loops.

Ex:     char ch;

```

strcpy(ch, "C")
if(isalpha(ch))
{
    printf("this is a character\n");
}

```

In the above example, `isalpha` will test `CH`. If it is a letter (A – Z), then it will return positive and the `printf` statement will be processed. If `CH` contains something other than A-Z, then it will return negative and the `printf` statement will not be processed.

`isalpha` can be used in a **NOT** condition.

```

Ex:   char ch;
      strcpy(ch, "C");
      if(!isalpha(CH))
      {
          printf("not a character C\n");
      }

```

in the above example, as long as `CH` does not equal `C`, then the `printf` loop will process.

## isdigit

Test a single character inclusive 0 through 9. If returns nonzero (1) if the character is a letter (0-9). Returns a zero (0) if not.

`isdigit` can be used in IF and WHILE loops.

```

Ex:   char ch;
      strcpy(ch, "1")
      if(isdigit(ch))
      {
          printf("this is a digit\n");
      }

```

In the above example, `isalpha` will test `CH`. If it is a digit (0 – 9), then it will return positive and the `printf` statement will be processed. If `CH` contains something other than 0-9, then it will return negative and the `printf` statement will not be processed.

`isdigit` can be used in a **NOT** condition.

```

Ex:   char ch;

```

```

strcpy(ch, "2");
if(!isdigit(CH))
{
    printf("not a digit\n");
}

```

in the above example, as long as CH does not equal C, then the printf loop will process.

## isspace

**isspace** returns nonzero (one) if c is a space, tab, carriage return, newline, vertical tab or formfeed.

## isupper

## localtime

To obtain either the current time or date, use the following command:

```
struct tm *local = localtime(&now);
```

It will return the following fields in char format:

```

local->tm_hour    /* get hours since midnight(0-23) */
local->tm_min     /* get minutes passed after the hour (0 – 59) */
local->tm_seconds /* get seconds passed after the minute (0 – 59) */
local->tm_day     /* day of the month (xx) */
local->tm_month   /* month (xx) */
local->tm_year    /* year (xxxx) */

```

## memcpy

Copies a given number of bytes from one string into another, truncating or padding as necessary.

```
Ex:    strncpy(astring, xstring,3);
```

In this example, 3 characters will be copied from xstring into astring starting at



position zero of pstring, and placed in astring starting at position 0.

Ex:     strncpy(astring, xstring+2, 4);

In this example, 4 characters of xstring starting at position 2 in xstring will be copied to astring starting at position 0.

## **memmove**

Copies a given number of bytes from one string into another, truncating or padding as necessary.

Ex:     strncpy(astring, xstring,3);

In this example, 3 characters will be copied from xstring into astring starting at position zero of pstring, and placed in astring starting at position 0.

Ex:     strncpy(astring, xstring+2, 4);

In this example, 4 characters of xstring starting at position 2 in xstring will be copied to astring starting at position 0.

## **printf**

```
printf("this is a test\n");  
printf("My name is - %s\n", name);  
printf("My age is - %d\n", age);      (raw number)  
printf("My age is - %.1d\n",age);    (one decimal position)  
printf("My age is - %.2d\n",age);    (two decimal position)
```

## **puts**

## **scanf**

```
scanf("%c",tt)  
    Accepts a single character from the console.
```

```
scanf("%s",tt)  
    Accepts a string from console. Stops at first space,tab or newline.
```

```
scanf("%d",x)  
    Accepts a integer from console.
```

## **sizeof**

## **snprintf**

## **strcat**

Appends one string to another.

Ex:     `strcat(pstring, xstring);`

String xstring will be appended to the end of pstring.

## **strcmp**

Compares one string to another.

Ex:     `int ret;`  
         `char fielda[10];`  
         `char fieldb[10]`  
         `ret = strcmp(fielda, fieldb);`

ret must be defined as an int.

## **strcpy**

Copies one string into another.

Ex:     `strcpy(xstring, "test of strcpy");`  
         `strcpy(xstring, any);`

xstring is the receiving field, and the data to the right of the comma is the sending field.

## **strlen**

Calculates the length of a string.

Ex:     `s = strlen(p_string);`

s will contain the length of p\_string. s must be defined as an int.

## **strncpy**

Copies a given number of bytes from one string into another, truncating or padding as

necessary.

Ex: `strncpy(astring, xstring, 3);`

In this example, 3 characters will be copied from xstring into astring starting at position zero of pstring, and placed in astring starting at position 0.

Ex: `strncpy(astring, xstring+2, 4);`

In this example, 4 characters of xstring starting at position 2 in xstring will be copied to astring starting at position 0.

## **strrev**

Reverses a string.

**strrev** changes all characters in a string to reverse order, except the terminating null character. The receiving field and the sending field both must be defined as character fields.

pp     DC   CL10

string DC   CL10'ABCDEFGHJI'

Ex:    `pp = strrev(string);`

## **strset**

Sets all characters in a string to a given character.

Ex:    `strset(p_string, '#');`

This will replace all characters in p\_string with '#'.

## **switch**

## **time**

Returns the current time, in seconds, elapsed since 00:00:00 GMT, January 1, 1970, and stores that value in the location pointed to by timer, provided that timer is not a null pointer.

Ex:

double seconds;

```
seconds = time(NULL);
```

## **t\_time**

## **tolower**

The *tolower()* function takes an uppercase alphabet and convert it to a lowercase character. If the arguments passed to the *tolower()* function is other than an uppercase alphabet, it returns the same character that is passed to the function.

## **toupper**

The *toupper()* function takes an lowercase alphabet and convert it to a uppercase character. If the arguments passed to the *toupper()* function is other than an lowercase alphabet, it returns the same character that is passed to the function.

## **while**

```
while(field1 operand field2)
{
    ....
}
```

field1 and field2 can be an numeric variable or a numeric constant.

Numeric variable

ex:     int ct

Numeric Constant

ex:     1,55,1234 . . . .

operand allowed:

==	field1 equal to field2
!=	field1 not equal field2
>	field1 greater than field2
<	field1 less than field2
>=	field1 greater than or equal to field2
<=	field1 less than or equal to field2

# Math

## ***Addition***

$x = 7;$   
 $x = ct;$   
 $x = x + 8;$   
 $x = x + ct;$   
 $x = 8 + x;$   
 $x = 8 + 8;$   
 $x = ct + d;$

## ***Subtraction***

$z = z - 8;$   
 $z = z - ct;$   
 $z = 10 - 2;$

## ***Multiplication***

$x = 8 * 2;$   
 $x = z * 2;$   
 $x = 10 * z;$   
 $x = y * z;$

## ***Division***

$x = 8 / 2;$   
 $x = z / 2;$   
 $x = 10 / z;$   
 $x = y / z;$

# String

## ***Example 1:***

```
int pi;  
char ch;  
  
pi = 3;  
  
ch = pstring[pi];
```

In this example, ch holds the char found in pstring at location pointed to by pi. Only **ONE** character at a time may be moved this way.

## ***Example 2:***

```
int pi;  
char ch;  
  
pi = 2;  
  
pstring[pi] = ch;
```

In this example, pstring[pi] is set to the character found in ch. Only **ONE** character a time may be moved this way.

## Array – Integer

### Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

```
type arrayName [ arraySize ];
```

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement –

```
double balance[10];
```

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

### Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows –

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array –

```
balance[4] = 50.0;
```

The above statement assigns the 5<sup>th</sup> element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above –

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

### Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

```
double salary = balance[9];
```

The above statement will take the 10<sup>th</sup> element from the array and assign the value to salary variable.



## **Array – Character**

## Structures

## Includes

time.h

## Non Processed Functions

`FILE *f_log;`

Z390 does not require this pointer. It can be left in the “C” program, but will not be processed by c2z.

`#include <xxxxx.h>`

System include header files are not processed as c2 uses the installed libraries of the underlying operating system. They can be included in the “C” program for compatibility but are NOT processed by c2z.

`malloc`

Z390 does not require this keyword. It can be left in the “C” program, but will not be processed by c2z.

## c2z Functions

Function	Passed	Compare	Return
isalnum	C370L1	C370L1A	C370ISAL
isdigit	C370L1	C370L1A	C370ISDG
isupper	C370L1	C370L1A	C370ISAL

NOTE: In all functions, return value is decimal value:

0	=	False
1	=	True

## c2z Parser Examples

### if

The following math operands are supported:

==  
!=  
<=  
>=  
<  
>

```
if(temp_byte[x] != 0)
if(pi < len)
```

### strcpy

strcpy(abc,"1234")	Variable - Literal
strcpy(abc,xyz)	Variable - Variable
strcpy(abc[x],xyz[z])	Array - Array

### strlen

xyz = strlen(abc);	Variable
xyz = strlen(array[ii]);	Array

### while

The following math operands are supported:

==  
!=  
<=  
>=  
<

```
while(ch != '\0')
```

## MATH

int x;	Integer set to zero (default)
int x = 2;	Integer set to 2

```
ch = p_string[pi]
token[t1] = ch
in_stack[ndx][0] = '\0'
iv_stack[ndx] = rdp_set();
```

```
temp_byte[x] = array1[z];
```

Array - Array

## Example Program

Below is a sample “Hello World” program.

```
/******  
*                                           *  
* hello.c                               *  
* Sample demo hello world program       *  
*****/
```

“C” source code:

```
*  
* This program was generated by c2z parser.  
* Generated code is for the z390 MLC.  
* Copyright (c) TCCS 2015 - 2016  
*  
* This is a modified MVC2 macro. Modified for baseless code  
* Modified by TCCS 2016  
* Performs a MVC operation, BUT using the Source Length NOT Target Length  
* MVC2 BUFFER,=C'Message Text' Should move 12 characters  
*  
*BUFFER DS CL133  
*  
MACRO  
&LAB MVC2 &TARGET,&SOURCE  
LARL R8,&SOURCE  
&LAB CLC 0(0,0),R8  
ORG *-6  
LARL R9,&TARGET  
LA 0,R9.(0)  
ORG *-4  
DC AL1(X'D2',L'&SOURCE-1),AL4(X'90008000')  
MEND  
*  
hello SUBENTRY  
JLU START /* Ln#- 0 R5 parm lg */  
*****  
* open/close macro code goes here *  
*****  
*  
*  
C370END DS 0H  
SUBEXIT  
*****  
* *  
* STATIC STORAGE AREA FOR CSECT - NO BASE REG REQUIRED. *  
* - LARL ADDRESSING IS USED *  
* *  
*****  
*****  
* *  
* CODE AREA FOR CSECT -DOESN'T REQUIRE BASE REGISTER COVERAGE *  
* -USE RELATIVE BRANCHES HERE *  
* -DON'T CODE LITERALS HERE *  
* *  
*****  
START DS 0H
```



```

XR      0,0
LARL    R1,L10
LARL    R2,C370WTOB
MV10    MVST  R2,R1
        JO     MV10
BRASL   C370PRTR,WTOALPHA
C370EXIT DS    0H
        SUBEXIT
*
WTOALPHA DS    0H
        LARL   R9,C370WTO
        WTO    MF=(E,0(R9))
        LARL   R9,C370WTOB
        LARL   R8,C370WTOB
        XC     0(78,R9),0(R8)
        BR     C370PRTR
*
WTOMVC   DS    0H
        LARL   R9,C370L10
        LARL   R8,C370B10
        MVC    0(10,R9),0(R8)
        LARL   R9,C370EDN
        LARL   R8,C370N3
        CP     0(4,R9),0(4,R8)
        JLH    PRT2
        LARL   R9,C370DEC
        LARL   R8,C370ZERO
        CP     0(4,R9),0(4,R8)
        JLE    PRT10
        LARL   R9,C370DEC
        LARL   R8,C370ONE
        CP     0(4,R9),0(4,R8)
        JLE    PRT11
        LARL   R9,C370DEC
        LARL   R8,C370TWO
        CP     0(4,R9),0(4,R8)
        JLE    PRT12
        JLU    PRT99
        /* Ln#- 15 br zero dec */
PRT10    DS    0H
        /* Ln#- 15 zero dec */
        LARL   R9,C370PN2
        LARL   R8,C370EDN
        ZAP    0(4,R9),0(4,R8)
        LARL   R9,C370TDW2
        LARL   R8,C370EDW2
        MVC    0(4,R9),0(R8)
        LARL   R9,C370TDW2
        LARL   R8,C370PN2
        ED     0(4,R9),2(R8)
        LARL   R9,C370L10
        LARL   R8,C370TDW2
        MVC    0(4,R9),0(R8)
        JLU    PRT99
PRT11    DS    0H
        /* Ln#- 15 one dec printf */
        LARL   R9,C370PN2
        LARL   R8,C370EDN
        ZAP    0(4,R9),0(4,R8)
        LARL   R9,C370TD1D
        LARL   R8,C370ED1D
        MVC    0(10,R9),0(R8)
        LARL   R9,C370TD1D
        LARL   R8,C370PN2
        ED     0(10,R9),0(R8)
        LARL   R9,C370L10
        LARL   R8,C370TD1D
        MVC    0(9,R9),2(R8)
        JLU    PRT99

```

PRT12	DS	0H	/* Ln#- 15 two dec printf */
	LARL	R9,C370PN2	
	LARL	R8,C370EDN	
	ZAP	0(4,R9),0(4,R8)	
	JLU	PRT99	
PRT2	DS	0H	
	LARL	R9,C370EDN	
	LARL	R8,C370N5	
	CP	0(4,R9),0(4,R8)	
	JLH	PRT3	
	LARL	R9,C370DEC	
	LARL	R8,C370ZERO	
	CP	0(4,R9),0(4,R8)	
	JLE	PRT20	/* Ln#- 15 br zero dec */
	LARL	R9,C370DEC	
	LARL	R8,C370ONE	
	CP	0(4,R9),0(4,R8)	
	JLE	PRT21	/* Ln#- 15 br one dec */
	LARL	R9,C370DEC	
	LARL	R8,C370TWO	
	CP	0(4,R9),0(4,R8)	
	JLE	PRT22	/* Ln#- 15 br two dec */
	JLU	PRT99	
PRT20	DS	0H	/* Ln#- 15 zero dec printf */
	LARL	R9,C370PN3	
	LARL	R8,C370EDN	
	ZAP	0(4,R9),0(4,R8)	
	LARL	R9,C370TDW3	
	LARL	R8,C370EDW3	
	MVC	0(6,R9),0(R8)	
	LARL	R9,C370TDW3	
	LARL	R8,C370PN3	
	ED	0(7,R9),1(R8)	
	LARL	R9,C370L10	
	LARL	R8,C370TDW3	
	MVC	0(6,R9),0(R8)	
	JLU	PRT99	
PRT21	DS	0H	/* Ln#- 15 one dec printf */
	LARL	R9,C370PN2	
	LARL	R8,C370EDN	
	ZAP	0(4,R9),0(4,R8)	
	LARL	R9,C370TD1D	
	LARL	R8,C370ED1D	
	MVC	0(10,R9),0(R8)	
	LARL	R9,C370TD1D	
	LARL	R8,C370PN2	
	ED	0(10,R9),0(R8)	
	LARL	R9,C370L10	
	LARL	R8,C370TD1D	
	MVC	0(9,R9),2(R8)	
	JLU	PRT99	
PRT22	DS	0H	/* Ln#- 15 two dec printf */
	JLU	PRT99	
PRT3	DS	0H	
	LARL	R9,C370DEC	
	LARL	R8,C370ZERO	
	CP	0(4,R9),0(4,R8)	
	JLE	PRT30	/* Ln#- 15 br zero dec */
	LARL	R9,C370DEC	
	LARL	R8,C370ONE	
	CP	0(4,R9),0(4,R8)	
	JLE	PRT31	/* Ln#- 15 br one dec */
	LARL	R9,C370DEC	
	LARL	R8,C370TWO	
	CP	0(4,R9),0(4,R8)	
	JLE	PRT32	/* Ln#- 15 br two dec */

```

PRT30      JLU      PRT99
           DS       0H                      /* Ln#- 15 zero dec printf */
           LARL     R9,C370TDW4
           LARL     R8,C370EDW4
           MVC      0(10,R9),0(R8)
           LARL     R9,C370TDW4
           LARL     R8,C370EDN
           ED       3(7,R9),0(R8)
           LARL     R9,C370L10
           LARL     R8,C370TDW4
           MVC      0(7,R9),3(R8)
PRT31      JLU      PRT99
           DS       0H                      /* Ln#- 15 one dec printf */
           LARL     R9,C370PN2
           LARL     R8,C370EDN
           ZAP      0(4,R9),0(4,R8)
           LARL     R9,C370TD1D
           LARL     R8,C370ED1D
           MVC      0(10,R9),0(R8)
           LARL     R9,C370TD1D
           LARL     R8,C370PN2
           ED       0(10,R9),0(R8)
           LARL     R9,C370L10
           LARL     R8,C370TD1D
           MVC      0(10,R9),0(R8)
PRT32      JLU      PRT99
           DS       0H                      /* Ln#- 15 two dec printf */
PRT99      JLU      PRT99
           DS       0H
           BR       C370PRTR
           EQUREGS
C370PRTR   EQU      R11                      /* c2 gen variable */
C370LNK    EQU      R10                      /* c2 gen variable */
*
*
* Character Literals
*
*           DS       0H
*
* Math Literals
*
*
* C37F1     DC       P'0000000'              /* c */
* NULL     DC       P'-000001'             /* NULL */
* argc     DC       P'0000000'             /* argc */
*           DS       0H
*
* Global Variables
*
* L10       DC       C'Hello z390 World',X'0'
*           DS       0H
*
*
* C37F2     DS       F'0'                   /* STRLEN */
*
* argv     DC       CL32' '                 /* argv */
*
* Local Variables
*
*           DS       0H
*
*
*
* c2z Parser Variables

```

```

*
C370WTO DC AL2 (C370WTOE-*,0) /* c2z gen variable */
C370WTOB DC CL78' ' /* c2z gen variable */
C370WTOE EQU * /* c2z gen variable */
C370EDN DC P'0000000' /* c2z gen variable */
C370PN2 DS P'0000000' /* c2z gen variable */
C370PN3 DS P'0000000' /* c2z gen variable */
DS 0H /* c2z gen variable */
C370L8 DC C' ' /* c2z gen variable */
C370B8 DC C' ' /* c2z gen variable */
DS 0H /* c2z gen variable */
C370ZERO DC P'0000000' /* c2z gen variable */
DS 0H /* c2z gen variable */
C370ONE DC P'0000001' /* c2z gen variable */
DS 0H /* c2z gen variable */
C370TWO DC P'0000002' /* c2z gen variable */
C370EDW2 DC X'40202120' /* c2z gen variable */
C370TDW2 DS C' ' /* c2z gen variable */
C370EDW3 DC X'402020202120' /* c2z gen variable */
C370TDW3 DS C' ' /* c2z gen variable */
C370EDW4 DC XL10'40202020202021202020' /* c2z gen variable */
C370TDW4 DS CL10 /* c2z gen variable */
C370N3 DC P'0000999' /* c2z gen variable */
C370N5 DC P'0099999' /* c2z gen variable */
DS 0H /* c2z gen variable */
DS 0H /* c2z gen variable */
C370DEC DC P'0000000' /* c2z gen variable */
DS 0H /* c2z gen variable */
C370PER DC C'.' /* c2z gen variable */
DS 0H /* c2z gen variable */
C370TD1D DS CL10 /* c2z gen variable */
DS 0H /* c2z gen variable */
C370ED1D DC XL10'402020206B2120204B20' /* c2z gen variable */
C370B10 DC C' ' /* c2z gen variable */
C370L10 DS CL10 /* c2z gen variable */
END

```

## Programs

c2z.c

c2z\_char.c

c2z\_debug.c

c2z\_double.c

c2z\_eoj.c

c2z\_error.c

c2z\_files.c

c2z\_for.c

c2z\_if.c

c2z\_int.c

c2z\_macro.c

c2z\_math.c

c2z\_mem.c

c2z\_misc.c

c2z\_paramaters.c

c2z\_print.c

c2z\_string.c

c2z\_time.c

c2z\_utility.c

c2z\_while.c

prototyp.h

## Functions

Function	Program
<code>int main(argc, argv[]);</code>	<code>c2.c</code>
<code>void a_bort(int,int);</code> <code>void a_warn(int,int);</code>	<code>c2_error.c</code> <code>c2_error.c</code>
<code>void c2_eoj(void);</code> <code>void c2_while(void);</code>	<code>c2_eoj.c</code> <code>c2_while.c</code>

```
/*          c2z_if.c          */
void c2_if(void);
void c2_case(void);
void c2_switch(void);
void c2_case_end(void);
void c2_break(void);
void c2_case_default(void);
```

```
/*          c2z_string.c      */
void c2_strcpy(void);
void c2_strcat(void);
void c2_strlen(void);
void c2_strchr(void);
void c2_strrchr(void);
void c2_strcmp(void);
void c2_strncpy(void);
void c2_strset(void);
```

```
/*          c2z_time.c        */
void c2_ctime(void);
void c2_time(void);
void c2_compute_time(void);
void c2_localtime(void);
```

```
/*          c2z_utility.c     */
void write_remark();
void write_short();
void write_variable();
void check_blank(void);
void check_length(void);
void check_continuation(void);
void check_semi(void);
void change_case(void);
```

```

/*          c2z_files.c          */
void c2_open(void);
void c2_close(void);
void c2_scan_fopen(void);
void c2_fgets(void);
void c2_scan_fgets(void);
void c2_fputs(void);
void c2_scan_fputs(void);
void c2_scan_feof(void);

/*          c2z_for.c          */
void c2_for(void);

/*          c2z_math.c          */
void c2_math(void);
void c2_plus(void);
void c2_minus(void);
void c2_atoi(void);

/*          c2z_print.c          */
void c2_fprintf(void);
void c2_printf(void);

/*          c2z_misc.c          */
void c2_regs(void);
void c2_func_call(void);
void c2_func_end(void);
void c2_func_sub(void);
void c2_goto(void);
void c2_goto_label(void);
void c2_isalpha(void);
void c2_isdigit(void);
void c2_exit(void);
void c2_define(void);
void c2_main(void);

/*          c2z_mem.c          */
void c2_mempy(void);
void c2_memmove(void);

/*          c2z_int.c          */
void c2_int(void);
void c2_int_1(void);
void c2_int_2(void);

/*          c2z_double.c          */
void c2_double(void);

/*          c2z_char.c          */
void c2_char(void);

/*          c2_debug.c          */
void c2_debug(void);

/*          c2z_paramaters.c          */

```

```
void c2_parm_ct(void);
```

```
/*          c2z_macro.c          */  
void c2_sizeof(void);
```



## Test Programs

The following test programs can be executed by running in the Z390 the batch job stream ASMLG. This will compile, link and execute the test program.

### **ctest\_1.c**

Simple hello world program.

### **ctest\_2.c**

example of int and char defines.

### **ctest\_3.c**

math addition examples

### **ctest\_8.c**

for / next examples

### **ctest\_9.c**

while examples

### **ctest\_10.c**

strcpy, strcat, strlen examples

### **ctest\_11.c**

atoi examples

### **ctest\_12.c**

int and char array examples

```

/* usage counters */
int var_use[24];

/* ***** */
* 1 =      asmct                2 =      printf          *
* 3 =      fprintf              4 =      branch          *
* 5 =      fopen                6 =      fclose          *
* 7 =      if                   8 =      strcpy          *
* 9 =      isalpha              10 =     isdigit          *
* 11 =     isspace              12 =     isalnum          *
* 13 =     atoi                 14 =     free            *
* 15 =     strcat               16 =     strlen          *
* 17 =     define               18 =     fgets           *
* 19 =     unsigned             20 =     isupper          *
***** */

/* compiler generated fields usage counter */
int work_use_ct[80];
/* ***** */
* 1 =      C370ISAL             2 =      C370L1          *
* 3 =      C370L2               4 =      C370L3          *
* 5 =      C370L4               6 =      C370L5          *
* 7 =      C370L6               8 =      C370L7          *
* 9 =      C370L8               10 =     C370L80         *
* 11 =     C370MTOT             12 =     C370MT1         *
* 13 =     C370MT2             14 =     C370MT3         *
* 15 =     C370MT4             16 =     C370MT5         *
* 17 =     C370PTOT            18 =     C370PT1         *
* 19 =     C370PT2            20 =     C370PT3         *
* 21 =     C370PT4            22 =     C370PT5         *
* 23 =     C370CLCT            24 =                      *
* 25 =     C370EDW2            26 =     C370EDW3         *
* 27 =     C370EDW4            28 =                      *
* 29 =     C370PN2             30 =     C370PN3         *
* 31 =     C370ISDG            32 =     C370ZERO         *
* 33 =     C370ONE             34 =     C370TWO         *
* 35 =     C370UCA             36 =     C370UCZ         *
* 37 =     C370LCA             38 =     C370LCZ         *
* 39 =     C370NL1             40 =     C370DEND         *
* 41 =     C370IENT            42 =     REMAINDER        *
* 43 =     C370ISOR            44 =     C370MLT1         *
* 45 =     C370MLT2            46 =     C370MLT3         *
* 47 =     C370STRG            48 =     C370U           *
* 49 =     C370NWK1            50 =     C370NWK2         *
* 51 =     C370EDN             52 =     C370B1          *
* 53 =     C370B8              54 =     C370TDW2         *
* 55 =     C370TDW3            56 =     C370TDW4         *
* 57 =     C370N3              58 =     C370N5          *
* 59 =     C370L1A             60 =     C370NL1A         *
* 61 =     C370LWQ             62 =     C370LZER         *
* 63 =     C370LNIN            64 =     FW00XX004        *
* 65 =     C370NWK3            66 =     C370PDBL         *
* 67 =     C370DEC             68 =     C370TD1D         *
* 69 =     C370ED1D            70 =     C370PNID         *
* 71 =     C370L10             72 =     C370XXX         *
* 73 =     C370LPCT            74 =     C370EOF         *
* 75 =     C370PER             76 =     C370B10         *
* 77 =     C370ECB             78 =     C370NWK3         *
* 79 =     C370FONE            *
***** */

```

## Internals

These entries are stored in the v\_variable table.

Variable type

A	=	Arrays – Character
C	=	Character
D	=	Double
G	=	Array - Integer
I	=	Integer
M	=	Entry in w_variable for reference only

For v\_type **A** and **G**, the following fields should always be mapped to the following:

field5	=	v_dsect
field6	=	v_label
field7	=	v_table
field8	=	v_label
field9	=	v_sv_reg
field10	=	v_wk_reg