c113c

a Programming Language

Rachel Powers rachel.powers@student.nmt.edu
Leon Rogge leonjoaquim.rogge@student.nmt.edu
Marek Listy marek.listy@gmail.com

Draft Version 0.1, January 28, 2021.

Language Reference Manual

Abstract

c113c (pronounced "See 113 See", short for CSE 113 Compiler) is a subset of the ANSI C Programming Language c113c is a tiny language intended to be implemented in a compiler construction class.

New Mexico Institute of Mining and Technology
Department of Computer Science and Engineering
Socorro, NM 87801 USA

Contents

1. Introduction

c113c is a subset of C. c113c is intended to correspond roughly to the subset of C that would be covered in a CS1 class such as NMT's CS 113 course.

The facilities that c113c supports are just barely interesting enough to write some non-trivial computations in it.

c113c programs are legal C programs with a .c file extension. A program begins with a main() procedure. A "Hello world" program looks like:

```
#include

int main() {
 printf("Hello, world");
 return 0;
}
```

The c113c include facility is restricted to only those built-in system includes used in CSE 113, which are faked in c113c. C113c supports a small subset of the functionality of a small subset of C's includes, including stdio.h (printf), time.h (currenttime), and math.h (rand). While the full C versions of these libraries support many many functions, and even types, c113c will be minimalist. For example, instead of defining 25 public symbols in stdio.h, c113c will have as few as possible, possibly only one or two. So far, only printf().

C uses variable initializers. c113c does not do initializers.

```
int c, p, j = 1; /* legal in C, not in c113c */
```

C features many basic types. c113c supports:

```
char
int
float64
```

c113c has only one type of loop, the while loop. Curly braces are required.

Conditionals use syntax similar to loops. Curly braces are required. An else branch is optional.

```
if (x < 0) {
 ...
}
```

Else branches require curly braces, unless they are (chained) if statements. Due to Go's wimpy semi-colon insertion that is not as good as Unicon's semi-colon insertion rule, chained elses have to be on the line with the curly brace that closes the preceding then-part or else-part.

```
if (x < 0) {
 ...
} else if (x < 10) {
 ...


} else {
 ...
}
```

c113c does not do switch statements.

C supports creation of new types via a struct. c113c has structs.

C has pointers, but no pointer arithmetic. c113c should support just enough pointers to support homework assignments in CSE 113. Linked lists. Thus, pointers to structs.

C has arrays. c113c has one-dimensional arrays only.

When in doubt about c113c features, refer to the C language specification. I will add notes below as needed. The easiest way to get out of having to implement something is to ask about it and negotiate.

2. Lexical Rules

The lexical rules of c113c start with: the lexical rules of C. c113c may simplify and reduce the lexical rules of C a bit.

2.1 Whitespace and Comments

C has no specific symbol for a single whitespace. The same is true for c113c.

C uses a set of characters to describe different white spaces.

' ' space

'\t'   horizontal tab

'\n' newline

'\v' vertical tab

'\f'   feed

'\r'   carriage return

c113c only implements ' ' and '\n'.

In C comments are text placed between the delimiters /* and */. Comments can also use // to comment a single line. c113c behaves the same as C in respects to comments.

Example:

//single line comment

/* this is a

multiple line

comment */

2.2 Reserved Words

C has 33 reserved words listed below.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void

continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

c113c only includes:

Reserved Word	**Description**	**Example**

if	Executes the body of the "if" statement given condition returns true.	if (1 < 3) {
		// code to be executed
		}

else	Follows the "if" statement. If the condition of the "if" statement is false, execute "else" statement.	if (1 > 3) {
		/*code to be executed if condition true */
		}
		else {
		/* code to be executed if condition false */
		}

while	A loop that runs as long as the condition remains true.	while (1 == 1) {
		/* code to be executed if condition true */
		}

for	A loop that can set to run a specific amount of times. It depends on an init to initialize a variable, a condition and an increment.	for (int i = 1; i <= 10; i++) {
		/* code to be executed 10 times*/
		}

break	A statement used to exit the nearest loop or control structure when it is encountered.	while (1 == 1) {
		    break;
		} //exits to loop immediately

return	A return statement returns a value or pass control to another function.	int addOne(int n) {

		return n + 1;
		}
		int main() {
		int n = 0;
		printf(“%d”, addOne(n));
		}

void	Void is a return type, it shows that the function does not return a value.	void printmsg( ) {
		printf(“Hello”);
		}
		int main ( ) {
		printmsg( );
		return 0;
		}

char	A char is a data type which holds one character. Can be used with an array to hold multiple characters.	char bigS = 'S';
double	A double is a data type used to store decimals with double precision.	double y = 4244.546;
float	A float is a data type used to store decimals with single precision.	float x = 10.327000;

int	A int is a data type which is any whole number from zero to both positive or negative values. Has no decimals.	int x = 1000;
		int y = -478;

switch	A switch statement allows for a variable to be tested against many conditions.	switch (x)
		{
		case 1: //run code if x = 1
		break;
		case 2: //run code if x = 2
		break;
		default:
		}

case	Used in a switch statement which acts like multiple conditionals.	Refer to switch.
default	Used in a switch as the default condition if no other case has been met.	Refer to switch.

2.3 Operators

Arithmetic Operators

Binary Operators

Operator	Description	Example A = 10, B = 400
+	Add two operands	A + B = 410
-	Subtract Two operands	A - B = -390
*	Multiply two operands	A * B = 4000
/	Divide numerator by de-numerator.	B / A = 40
%	Modulo Operator, return the remainder after an integer devision	A % B = 0

Unary Operators

Operator	Description	Example A = 10, B = 400
++	Increment operand in place by one (can be post or prefixed)	A++ = 11
--	Decrement operand in place by one (can be post or prefixed)	A-- = 9
-	Flips the sign of a number	-A = -10

Relational Operators

Binary Operators

Operator	Description	Example A = 10, B = 400
==	Test equality	(A == B) // false
!=	Test inequality	(A != B) // true
>	If the left is greater than the right operand	(A > B) // false
<	If the left is less than the right operand	(A < B) // true
>=	If the left is greater than or equal to the right operand	(A >= B) // false
<=	If the left is less than or equal to the right operand	(A <= B) // true

Logical Operators

Binary Operators

Operator	Description	Example A = 1, B = 0
&&	Logical AND, if both operands are non-zero return true	(A && B) // false
||	Logical OR, if either operand is non-zero return true. Short circuits if the left operand is non-zero	(A || B) // true

Unary Operators

Operator	Description	Example A = 1, B = 0
!	Logical NOT, reverses the truth of a statement	!(A && B) // true

Bitwise Operators

Binary Operators

Operator	Description	Example A = 0b 1100 0010 , B = 0b 0001 0011
&	Bitwise AND	A & B = 0b 0000 0010
|	Bitwise OR	A | B = 0b 1101 0011
^	Bitwise XOR	A ^ B = 0b 1100 0001
~	Bitwise One’s Complement, flipps bits	~A = -0b 00111101
<<	Bitwise Left shift, zerofills	B << 2 = 0b 0100 1100
>>	Bitwise Right shift	B >> 2 = 0b 0000 0100

Assignment Operators

Binary Operators

Operator	Description	Example A = 10, B = 400
=	Assign value from right side statement to memory of left side	C = A + B // C will have the value of 410
+=	In place addition	C += A // is equivalent to C = C + A
-=	In place subtraction	C -= A // is equivalent to C = C - A
*=	In place multiplication	C *= A // is equivalent to C = C * A

/=	In place division	C /= A // is equivalent to C = C / A
%=	In place modulo division	C %= A // is equivalent to C = C % A
<<=	In place bitwise left shift	C <<= 2 // is equivalent to C = C << 2
>>=	In place bitwise right shift	C >>= 2 // is equivalent to C = C >> 2
&=	In place bitwise AND	C &= A // is equivalent to C = C & A
|=	In place bitwise OR	C |= A // is equivalent to C = C | A
^=	In place bitwise XOR	C ^= A // is equivalent to C = C ^ A

Other Operators

Unary Operators

Operator	**Description**	**Example**
&	Reference a variable returning the address of its memory	&A // the address of A
*****	De-reference a variable returning the value as the memory location.	*A // interpret A as a pointer and dereference.
sizeof()	Return the size of the memory in bytes	sizeof(A) // if A is an int is 4
(type)	Cast Operator, interpret the variable as another datatype	(float) A // A as a float

Ternary Operator

Operator	**Description**	**Example**

<condition> ? <true statement> : <false statement>	Conditional Expression.	If Condition is true ? then value X : otherwise Y

Operator Precedence

Operators have priority in the following order

Category	Operator	Associativity
1. Postfix	() [] -> . ++ - -	Left to right
2. Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
3. Multiplicative	* / %	Left to right
4. Additive	+ -	Left to right
5. Shift	<< >>	Left to right
6. Relational	< <= > >=	Left to right
7. Equality	== !=	Left to right
8. Bitwise AND	&	Left to right
9. Bitwise XOR	^	Left to right
10. Bitwise OR	|	Left to right
11. Logical AND	&&	Left to right
12. Logical OR	||	Left to right
13. Conditional	?:	Right to left

14. Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
15. Comma	,	Left to right

2.4 Literals

Literals	Rules and other notes	Examples
integer	Suffix: Unsigned (u or U), Long(l or L), Prefix: hexadecimal (0x or 0X), Octal (0) Possible Chars: 0-9, a-fA-F (hex), 0-8(octal)	212, 212u 0xFF4L, 0XEf4, 071
Floating Point	Decimal form: must include zero or one decimal point (.) and integer value on at least one side Exponent form: needs integer followed by exponent symbol (e or E) then by exponent integer Suffix : long double (l or L)	3.14, 3. , .14 ,3 314159E-5 3.14657l
Character	Enclosed in single quotes (‘ ’) Include plain characters (ASCII) , escape sequences, universal characters. Universal characters: defined as ‘\u’ followed with valid character values Escape sequences: require backslash (\ ) before required characters. An escaped Octal sequence (1-3 digits) is equivalent to a character, based on a character set An escaped hexadecimal sequence is also equivalent to a character based on a set. There is no limit on the amount of digits, yet an error will occur when greater than a character set	‘l’ , ‘j’ ‘\u02C0’ ‘\n’, \t’ , ‘\\’ ‘\101’ would be ASCII character ‘A’ ‘\41’ would be ASCII character ‘A’
String Literals	Enclosed in double quotes (“ ”) Include plain characters (ASCII), escape sequences, and universal characters	“Hello World” “Hello \nWorld”

String cont.	Strings are stored as arrays of characters, without size attribute If string is too long to fit in one line, a backslash (\) can split it into seperate lines Ended by a null termination character (\0), A string cannot contain unescaped double quotes	“This line is too long and cannot fit into \ one single line hence the break” “ He said \”Hi there!\” to the man.”

In CSE 113, and gcc in general, ASCII is the default character set.
Cse 113 does not include octal integers, exponential floats ,universal characters, and escaped octal or hexadecimal sequences.

2.5 Punctuation

These are all the punctuation characters in C.
! " # $ % & ' ( ) * +, - . / : ; ? @ [\ ] ^ _ ` { | } ~

c113c only includes:
! " ' () * + , - ; {} | [ ] \ ^ % &
Most of these forms of punctuation already have descriptions in the upper part of this document.

Punctuation	Description	Example
"	Is used to define a strings literal. Often used in print statements.	printf(“Hello”);

( )	Used by functions to declare values, dictates order in math equations, and holds conditionals for loop and if/else statements.	int main (){
		}
		if(conditional){
		}

{}	Dictates the start and end of loops, if/else statements, and functions.	if(conditional){ // start
		//code
		} //end

,	Used to separate expressions. Can also be used as an operator in C, but not in c113c.	int a, b, c;
		int add(int n, int x) {
		    return n + x;
		}

;	The semicolon dictates the end of a statement.	int x = 1;
[ ]	Brackets are used to index an array.	double array[10];
\	Used to in an addition to a letter to describe a whitespace. Also called an escape character.	\n new line \t horizontal tab
'	Used to define a single character.	char a = 'a';
:	In c113c you see colons in switch statements. There are other uses to colons in C such as being used to describe bitfields.	unsigned short num: 4;

2.6 Identifiers

Identifiers in c113c can only be made from upper and lowercase alphanumerics (a-z A-Z 0-9) and the underscore (_) they can contain no whitespace and must not start with a numeric digit.

Identifiers must also not be a reserved word

/ [a-zA-Z][a-zA-Z0-9_]* /

3. Syntax

3.1 Function Syntax

Functions in c113c follow the following format:
<return type> <Identifier function_name> ([<type> <identifier parameter_name>]*,) { <function body> }

3.2 Control Structures

```
If (statement) {}

If (statement) {} else {}

If (statement) {} else if (statement) {} else {}

while (statement) {}

for ( init; condition; increment) {}

switch(integer){
case literal:
  statements;
…
default:
  statements;
}
```

3.3 Structures

Structs in C and c113c behave the same.

```
struct <structure_name> {
            <data>
};
```

3.4 Declaration Syntax

type identifier;

type identifier = literal;

4. Data Types

4.1 Numbers

All data types listed are used by c113c and are described in the reserved words section.
char, short, int, long, float, double

4.2 Strings

char* array, null terminated ‘\0’

4.3 Arrays

As listed in the introduction C has multidimensional arrays whereas c113c single dimensional arrays. For example: int num[100];
Arrays are constructed as pointers of a specific type with a fixed memory width with sequential memory allocated either on the stack or heap. The reference pointer is to the first sequential location.

5. Library Functions

C includes a large host of functions, however, c113c only contains a small subset of functions listed below:

Function	Library	Use

printf	stdio.h	Prints to stdout
sprintf	stdio.h	Sends formatted output to a string
fopen	stdio.h	Opens the filename listed
fclose	stdio.h	Closes the file which was opened
malloc	stdlib.h	Allocates memory and returns a pointer
realloc	stdlib.h	Resize memory block pointed to by pointer which was previously allocated by malloc.
strlen	string.h	Finds Length of the string as unsigned integer
strcpy	string.h	Copies one string to destination
strcmp	string.h	Compares two strings, return result as integer
strtok	string.h	Splits a string into tokens using a delimiter, consuming the string.
sqrt	math.h	Takes single double argument and return the square root as a double
cos	math.h	Takes double argument in radians and returns cosine value as a double
pow	math.h	Takes two doubles, base and exponent. Returns base to the power of exponent ($base^{exponent}$)
sin	math.h	Takes double argument in radians and returns sine value as a double
rand	stdlib.h	Returns a random integer value between o and RAND_MAX

Summary

Sure, c113c may be a toy language designed for a compiler class. Even with only this much, it may provide a convenient notation for a lot of simple programming tasks such as those faced by students in CSE 113.