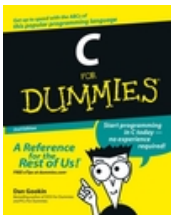
dummies
A Wiley Brand

Looking at the C Language



RELATED BOOK
C For Dummies, 2nd Edition

By **Dan Gookin**

Each program must have a starting point. When you run a program, DOS sends it off on its way — like launching a ship. As its last dock-master duty, DOS hurls the microprocessor headlong into the program. The microprocessor then takes the program's helm at that specific starting point.

In all C programs, the starting point is the **main()** function. Every C program has one, even GOODBYE.C (shown in Figure 1). The **main()** function is the engine that makes the program work, which displays the message on the screen.

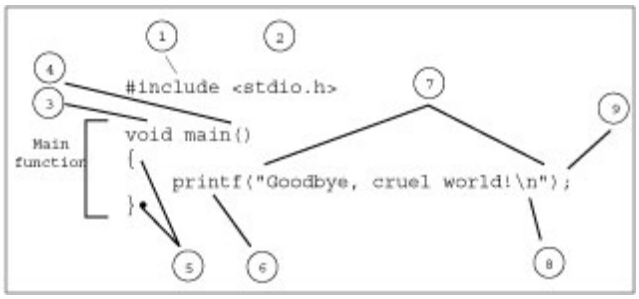


Figure 1: GOODBYE.C and its pieces and parts.

Other C programs may carry out other tasks in their **main()** function. But whatever's there, it's the first instruction given to the computer when the program runs.

- **main()** is the name given to the first (or primary) function in every C program. C programs can have other functions, but **main()** is the first one.

- It's a common convention to follow a C language function name with parentheses, as in **main()**. It doesn't mean anything. Everyone does it, and it's included here so that you don't freak when you see it elsewhere.

- In Borland C++, you may have seen the error message say "in function main." This message refers to the main function — the **void main()** thing that contains the C language instructions you've been writing.

- A function is a machine — it's a set of instructions that does something. C programs can have many functions in them, though the **main** function is the first function in a C program. It's required.

REMEMBER

Function. Get used to that word.

Pieces' parts

Here are some interesting pieces of the C program shown in Figure 1:

1. #include is known as a preprocessor directive, which sounds impressive, and it may not be the correct term, but you're not required to memorize it anyhow. What it does is tell the compiler to "include" another program or file along with your source code, which generally avoids a lot of little, annoying errors that would otherwise occur.

2. <stdio.h> is a filename hugged by angle brackets (which is the C language's attempt to force you to use all sorts of brackets and whatnot). The whole statement #include <stdio.h> tells the compiler to use the file STDIO.H, which contains standard I/O, or input/output, commands required by most C programs.

3. void main identifies the name of the function main. The void identifies the type of function or what the function produces. In the case of main, it doesn't produce anything, and the C term for that is "void."

4. Two empty parentheses follow the function name. Sometimes, there may be items in these parentheses.

5. The curly brackets or braces enclose the function, hugging in tight all its parts. Everything between { and } is part of the function main() in Figure 1.

6. printf is a C language instruction, part of the programming language that eventually tells the computer what to do.

---

7. Belonging to printf are more parentheses. In this case, the parentheses enclose text, or a "string" of text. Everything between the double quotes (") is part of printf's text string.

8. An interesting part of the text string is n. That's the backslash character and a little n. What it represents is the character produced by pressing the Enter key. What it does is to end the text string with a "new line."

9. Finally, the printf line, or statement, ends with a semicolon. The semicolon is how the C compiler knows when one statement ends and another begins — like a period at the end of a sentence. Even though printf is the only instruction in this program, the semicolon is still required.

• Text in a program is referred to as a *string.* For example, "la-de-da" is a string of text. The string is enclosed by double quotes.

• The C language is composed of keywords that appear in statements. The statements end in semicolons, just as sentences in English end in periods.)

The C language itself — the keywords

The C language is really rather brief. There are only 33 *keywords* in C. If only French were that easy! Table 1 shows the keywords that make up the C language.

Table 1: C Language Keywords

asm	enum	signed
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while

Not bad, eh? But these aren't all the words you find in the C language. Other words or instructions are called *functions.* These include jewels like **printf** and several dozen other common functions that assist the basic C language keywords in creating programs.

If you're using DOS, additional functions specific to DOS are piled on top of the standard C armada of functions. And if you get into Windows, you find hoards of Windows-specific functions that bring C's full vocabulary into the hundreds. And no, you don't really have to memorize any of them. This is why all C compilers come with a language reference, which you'll undoubtedly keep close to your PC's glowing bosom.

REMEMBER

Languages are more than a collection of words. They also involve grammar, or properly sticking together the words so that understandable ideas are conveyed. This concept is completely beyond the grasp of the modern legal community.

In addition to grammar, languages require rules, exceptions, jots and tittles, and all sorts of fun and havoc. Programming languages are similar to spoken language in that they have various parts and lots of rules.

- You will never be required to memorize the 33 keywords.
- In fact, of the 33 keywords, you may end up using only half on a regular basis.
- Some of the keywords are real words! Others are abbreviations or combinations of two or more words. Still others are cryptograms of the programmer's girlfriends' names.
- Each of the keywords has its own set of problems. You don't just use the keyword **else**, for example; you must use it *in context.*
- Functions like **printf** require a set of parentheses and lots of stuff inside the parentheses. (Don't fret over this right now; just nod your head and smile in agreement, "Yes, **printf** does require lots of stuff.")
- By the way, the fact that **printf** is a C function and not a keyword is why the **#include <stdio.h>** thing is required at the beginning of a program. The STDIO.H file contains the instructions telling the compiler what exactly **printf** is and does. If you edit out the

#include <stdio.h> line, the compiler produces a funky “I don’t know that **printf** thing” type of error.

Cheat Sheet

C For Dummies Cheat Sheet

From **C For Dummies, 2nd Edition**

By **Dan Gookin**

The C programming language is fast and versatile. You can use just 32 keywords and some fairly intuitive symbols to do comparisons and conversions. Then you get to numeric data and math symbols, which are pretty much as you expect as well.

C Language Comparison Symbols

If you're writing programs in C, you need to use comparison symbols. The symbols C uses, their meanings, and examples are shown in the following table:

Symbol	Meaning or Pronunciation	"True" Comparison Examples
<	Less than	1 < 5 8 < 9
==	Equal to	5 == 5 0 == 0
>	Greater than	8 > 5 10 > 0
<=	Less than or equal to	4 <= 5 8 <= 8
>=	Greater than or equal to	9 >= 5 2 >= 2
!=	Not equal to	1 != 0 4 != 3.99

C Language Comparisons and Their Opposites

If you're programming in C — or any other language — you need to use If/Else statements. The comparison symbols you need if you're working with C and the Else statements they generate are shown in the following table:

If Comparison	Else Statement Executed By This Condition
<	>= (Greater than or equal to)
==	!= (Not equal to)
>	<= (Less than or equal to)
<=	> (Greater than)
>=	< (Less than)
!=	== (Equal to)

C Language Conversion Characters

When programming in C, you use conversion characters — the percent sign and a letter, for the most part — as placeholders for variables you want to display. The following table shows the conversion characters and what they display:

Conversion Character	Displays Argument (Variable's Contents) As
%c	Single character
%d	Signed decimal integer (int)
%e	Signed floating-point value in E notation
%f	Signed floating-point value (float)
%g	Signed value in %e or %f format, whichever is shorter

%i	Signed decimal integer (int)
%o	Unsigned octal (base 8) integer (int)
%s	String of text
%u	Unsigned decimal integer (int)
%x	Unsigned hexadecimal (base 16) integer (int)
%%	(percent character)

C Language Escape Sequences

Programming in C is fast — all you have to do is type a short sequence of keystrokes — generally just two — to get a tab, a new line, a question mark, and more. The following table shows the sequences you need to accomplish a variety of tasks:

Sequence	Represents
a	The speaker beeping
b	Backspace (move the cursor back, no erase)
f	Form feed (eject printer page; ankh character on the screen)
n	Newline, like pressing the Enter key
r	Carriage return (moves the cursor to the beginning of the line)
t	Tab
v	Vertical tab (moves the cursor down a line)
\	The backslash character
’	The apostrophe
”	The double-quote character

?	The question mark
	The “null” byte (backslash-zero)
x*nnn*	A character value in hexadecimal (base 16)
X*nnn*	A character value in hexadecimal (base 16)

C Language Keywords

The C programming language has just 32 keywords for you to build robust programs. With only 32 keywords, they all fit nicely into a short table. Use them wisely and well.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	static	void
default	goto	sizeof	volatile
do	if	signed	While

C Language Numeric Data Types

When programming with C, keywords and variables go together like the 4th of July and fireworks, although with a bit less drama. The following table shows C keywords, their variable types, and their ranges:

Keyword	*Variable Type*	*Range*
`char`	Character (or string)	-128 to 127
`int`	Integer	-32,768 to 32,767
`short` `short int`	Short integer	-32,768 to 32,767
`long`	Long integer	-2,147,483,648 to 2,147,483,647
`unsigned char`	Unsigned character	0 to 255
`unsigned int`	Unsigned integer	0 to 65,535
`unsigned short`	Unsigned short integer	0 to 65,535
`unsigned long`	Unsigned long integer	0 to 4,294,967,295
`float`	Single-precision floating point (accurate to 7 digits)	$\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
`double`	Double-precision floating point (accurate to 15 digits)	$\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$

C Language Mathematical Symbols

Programming math functions with C is fairly straightforward: a plus sign works like any sixth-grader knows it should and does addition. The mathematical symbols and the function they serve in C are shown in the following table:

Operator or Symbol	*What You Expected*	*As Pronounced By Sixth-Graders*	*Task*
+	+	"Plus"	Addition
-	-	"Minus"	Subtraction
*	x	"Times"	Multiplication
/	÷	"Divided by"	Division
%	☂	??	Modulus