C

C/C++ for Scientists and Engineers

Presented by:

Jim Polzin

e-mail: james.polzin@normandale.edu
otto: http://otto.normandale.edu

TABLE OF CONTENTS

C OVERVIEW

Goals

- speed
- portability
- allow access to features of the architecture
- speed

C

- fast executables
- allows high-level structure without losing access to machine features
- many popular languages such as C++ , Java, Perl use C syntax/C as a basis
- generally a compiled language
- reasonably portable
- very available and popular

BASIC C PROGRAM STRUCTURE

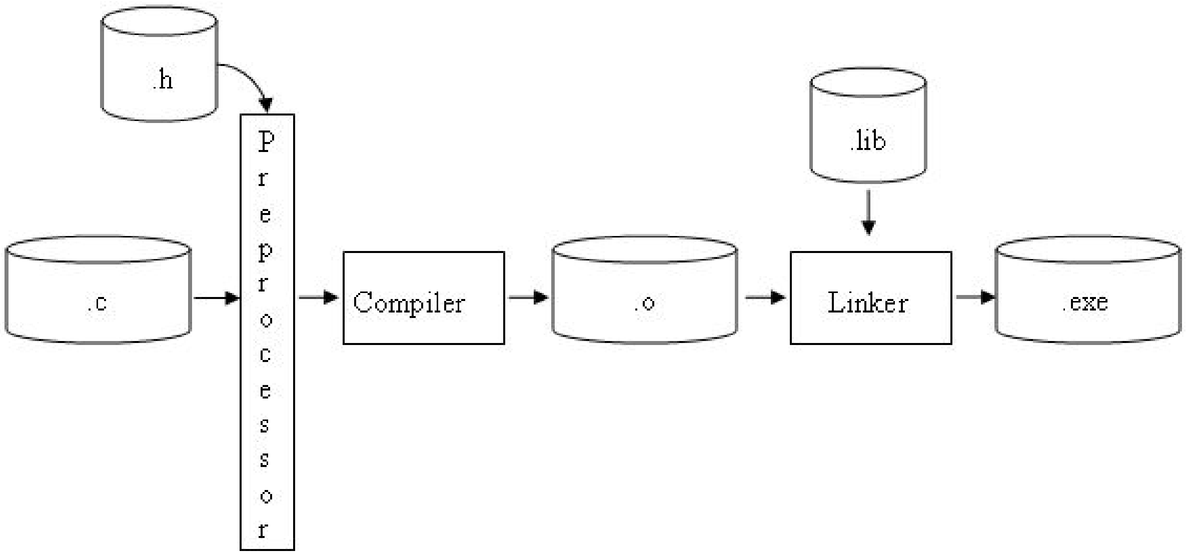
- The function main( ) is found in every C program and is where every C program begins execution.
- C uses braces { } to delimit the start/end of a function and to group sets of statements together into what C calls a block of code.
- Semicolons are used to terminate each C statement.
- Groups of instructions can be gathered together and named for ease of use and ease of programming. These “modules” are called functions in C.

Ex:

```
/*     FILE: first.c     */

#include <stdio.h>

int main( )
{
  printf("Hello world! \n");

  return 0;
}


/*    OUTPUT: first.c

          Hello world!

*/
```

COMPILING AND LINKING

- Producing an executable file from C source code involves a two step process, compiling and linking.
- The compiler translates the C code into machine code, the linker combines the new machine code with code for existing routines from available libraries and adds some startup code.
- The end result is a file full of machine instructions that are executable on the target machine.



Ex:

gcc first.c

- compile and link to a.exe

gcc -c first.c

- compile only, stop at object module

gcc -lm first.c

- link in math libraries

FUNDAMENTAL DATA TYPES

- there are three basic types of data, integer, floating-point, and character
- character data type is really a small integer
- signed and unsigned integer types available

Type	Size	Min	Max
char	1 byte	0 / -128	255 / 127
short	2 bytes	-32768	32767
int	2,4 bytes	-2147483648	2147483647
long	4 bytes	-2147483648	2147483647
long long	8 bytes	2^{63} ~ $-9x10^{18}$	$2^{63}-1$ ~ $9x10^{18}$
float	4 bytes ~ 7 digits	$\pm 1.0x10^{-37}$	$\pm 3.4x10^{+38}$
double	8 bytes ~ 14 digits	$\pm 1.0x10^{-307}$	$\pm 1.8x10^{+308}$
long double	12 bytes ~ 20 digits	$\pm 1.0x10^{-4931}$	$\pm 1.0x10^{+4932}$

Ex:

```
/*     FILE: unsigned.c     */

/*
   Illustration of the unsigned keyword.  Allows
   recovering the use of the lead bit for magnitude.
*/
#include <stdio.h>

int main( )
{
  unsigned int x;

  x = 3333222111;

  printf("Unsigned x = %u\n", x);

  printf("Signed x = %d\n", x);

  return 0;
}


/*    OUTPUT: unsigned.c

          Unsigned x = 3333222111
          Signed x = -961745185

*/
```

COMMENTS

- Block-style comments /* … */   Everything between the opening /* and the first */ is a comment.
- Comment-to-end-of-line: //   Everything from the // to the end of the line is a comment.
- Nesting of block-style comments doesn't work.
- Note: Some older compilers may not recognize the // comment indicator.

Ex:

```
/*     FILE: example.c     */

#include <stdio.h>

/* C-style comments can span several lines
   ...where they are terminated by:
*/

int main( )
{
  printf("Here's a program\n");

  return 0;
}


/*    OUTPUT: example.c

          Here's a program

*/
```

IDENTIFIERS

- C identifiers must follow these rules:
    - C is case sensitive
    - may consist of letters, digits, and the underscore
    - first character must be a letter, (could be underscore but this is discouraged)
    - no length limit but only the first 31-63 may be significant

KEYWORDS

auto	extern	short	while
break	float	**signed**	*_Alignas*
case	for	sizeof	*_Alignof*
char	goto	static	*_Bool*
const	if	struct	*_Complex*
continue	*inline*	switch	*_Generic*
default	int	typedef	*_Imaginary*
do	long	union	*_Noreturn*
double	register	unsigned	*_Static_assert*
else	*restrict*	void	*#_Thread_local*
enum	return	**volatile**	

BASIC INPUT AND OUTPUT

- The basic I/O functions are printf( ) and scanf( ).
- printf( ) and scanf( ) are very generic. They always are processing text on one end. They get all their information about the data type they are to print or scan from the conversion specifiers.
- printf( ) always is producing text output from any number of internal data formats, i.e. int, float, char, double. The job of the conversion specifiers is to tell printf( ) how big a piece of data it's getting and how to interpret the internal representation.
- scanf( ) always receives a text representation of some data and must produce an internal representation. It is the conversion specifiers job to tell scanf( ) how to interpret the text and what internal representation to produce.
- <u>printf( ) tips and warnings</u>:
  - Make sure your conversion specifiers match your data values in number, type and order.
  - Use %f for both float and double.
  - Everything you put in the format string prints exactly as it appears, except conversion specifiers and escape sequences.
- <u>scanf( ) tips and warnings</u>:
  - Make sure your conversion specifiers match your data values in number, type and order.
  - As a general rule, scan only one value with each scanf( ) call, unless you really know what you are doing.
  - Use %f for float, %lf for double and %Lf for long double.
  - Don't forget the &, except with strings. {Someday you'll know why that is, and it will make sense.}
  - For %c every character you type is a candidate, even <return>. Placing a space in front of the %c in the format string will cause scanf( ) to skip whitespace characters.
  - scanf( ) is NOT without it's problems. However, it provides an easy way to get text input into a program and has some very handy conversion capabilities.

CONVERSION SPECIFIERS

printf( )

%d	signed decimal int
%hd	signed short decimal integer
%ld	signed long decimal integer
%lld	signed long long decimal integer
%u	unsigned decimal int
%lu	unsigned long decimal int
%llu	unsigned long long decimal int
%o	unsigned octal int
%x	unsigned hexadecimal int with lowercase
%X	unsigned hexadecimal int with uppercase
%f	float or double [-]dddd.dddd.
%e	float or double of the form [-]d.dddd e[+/-]ddd
%g	either e or f form, whichever is shorter
%E	same as e; with E for exponent
%G	same as g; with E for exponent if e format used
%Lf,	
%Le,	
%Lg	long double
%c	single character
%s	string
%p	pointer

scanf( )

%d	signed decimal int
%hd	signed short decimal integer
%ld	signed long decimal integer
%u	unsigned decimal int
%lu	unsigned long decimal int
%o	unsigned octal int
%x	unsigned hexadecimal int
%f	float
%lf	double **NOTE:** double & float are distinct for scanf !!!!
%LF	long double
%c	single character
%s	string

ESCAPE SEQUENCES

- Certain characters are difficult to place in C code so an escape code or escape sequence is used to encode these characters.
- These escape sequences all begin with a backslash '\' and cause the encoded character to be placed into the program.

Escape	value
\n	newline
\t	tab
\f	formfeed
\a	alarm
\b	backspace
\r	carriage return
\v	vertical tab

Ex:

```
/*     FILE: print.c     */

/*
   Illustration of printf( ) and conversion specifiers.
*/
#include <stdio.h>

int main( )
{
  int x = 12;
  float y = 3.75;

  printf("%d", x);

  printf("\nx = %d\n", x);

  printf("y = %f\n", y);

  return 0;
}


/*    OUTPUT: print.c

          12
          x = 12
          y = 3.750000

*/
```

Ex:

```
/*     FILE: scan.c     */

/*
   Illustration of scanf( ).
*/
#include <stdio.h>

int main( )
{
  int x;
  float y;

  printf("x = %d\n", x);

  printf("y = %f\n", y);

  printf("Enter an integer value for x: ");
  scanf("%d", &x);

  printf("Enter a floating-point value for y: ");
  scanf("%f", &y);

  printf("x = %d\n", x);

  printf("y = %f\n", y);

  return 0;
}


/*    OUTPUT: scan.c

          x = 4206596
          y = 0.000000
          Enter an integer value for x: 7
          Enter a floating-point value for y: 3.3
          x = 7
          y = 3.300000

*/
```

Ex:

```
/*     FILE: scan2.c     */

/*
   Illustration of scanf( ) with characters and characters
   are integers.
*/
#include <stdio.h>

int main( )
{
  char c;

  printf("Enter a character: ");
  scanf("%c", &c);

  printf("c = %c\n", c);

  printf("c = %d\n", c);

  return 0;
}


/*    OUTPUT: scan2.c

          Enter a character: A
          c = A
          c = 65

*/
```

Ex:

```
/*     FILE: scan3.c     */

/*
   Illustration of interpretation caused by conversion specifiers.
*/
#include <stdio.h>

int main( )
{
  char c;
  int x;

  printf("Enter a character: ");
  scanf("%c", &c);

  printf("c = %c\n", c);

  printf("c = %d\n", c);

  printf("Enter an integer: ");
  scanf("%d", &x);

  printf("x = %d\n", x);

  printf("x = %c\n", x);

  return 0;
}


/*    OUTPUT: scan3.c

          Enter a character: 6
          c = 6
          c = 54
          Enter an integer: 6
          x = 6
          x = _

*/
```

OPERATORS

Arithmetic operators:

* / %	multiplication/division/modulus
+ −	addition/subtraction
+ −	positive/negative sign (unary)
++ − −	increment/decrement (unary)

Logical operators:

&&	AND
||	OR
!	NOT (unary)

Relational operators:

< <= > >=	less than, less than or equal to, greater than, greater than or equal to
= = !=	equal to and not equal to

Bit operators:

<< >>	left and right bit shift
&	bitwise AND
|	bitwise OR
^	bitwise exclusive or XOR
~	bitwise NOT (unary)

Assignment operators:

= += −= *= /= %= &= ^= |= <<= >>=

Address/Pointer operators:

&	address of (unary)
*	dereference (unary)

Structure operators:

.	structure member acccess
−>	member access thru a structure pointer

Other operators:

( )	function call
[ ]	array access
(*type*)	type cast (unary)
sizeof	data object size in bytes (unary)
?:	conditional operator
,	comma operator

OPERATOR PRECEDENCE

- The C compiler determines the order of operations in a C statement by operator precedence.
- Operator Precedence is the ranking of operators in C. The higher the rank the sooner the operator is evaluated.
- Parentheses can be used to override operator precedence.
- There are many kinds of operators but all operators are ranked via operator precedence.
- In the case of operators with the same rank, associativity is used and the operators are evaluated left-to-right or right-to-left.
- Operator precedence and associativity are detailed in the Operator Precedence Chart in the appendix, on the following page, and on pg. 53 in the K&R book

OPERATOR PRECEDENCE CHART

Operators	Associativity
( ) [ ] –> .	left to right *{( ) function call}*
! ~ ++ – – + – * & (*type*) sizeof	right to left *{All Unary}*
* / %	left to right
+ –	left to right
<< >>	left to right
< <= > >=	left to right
= = !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += –= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right

ARITHMETIC OPERATORS

- Arithmetic operators are the symbols used by C to indicate when an arithmetic operation is desired.
- Arithmetic operators follow the same precedence rules we learned as kids. Multiplication & division before addition and subtraction. In case of a tie evaluate left-to-right. { Look at a precedence chart and see if this is true. }
- The modulus operator, %, is an additional arithmetic operator. It produces the remainder of integer division and ranks at the same level as division in the precedence chart.
- The increment, ++, and decrement, --, operators are basically a shorthand notation for increasing or decreasing the value of a variable by one.

Ex:

```
/*     FILE: arith_1.c     */

/* Arithmetic operators */

#include <stdio.h>

int main( )
{
  int first, second, sum;

  first = 11;
  second = 12;

  sum = first + second;
  printf("sum = %d\n", sum);

  sum = first - second;
  printf("sum = %d\n", sum);

  sum = first * second;
  printf("sum = %d\n", sum);

  sum = first / second;
  printf("sum = %d\n", sum);

  return 0;
}


/*    OUTPUT: arith_1.c

          sum = 23
          sum = -1
          sum = 132
          sum = 0

*/
```

Ex:
```
/*     FILE: arith_2.c     */

/* Arithmetic operators with nicer output */

#include <stdio.h>

int main( )
{
  int first, second, sum;

  first = 11;
  second = 12;

  sum = first + second;
  printf("%d + %d = %d\n", first, second, sum);

  sum = first - second;
  printf("%d - %d = %d\n", first, second, sum);

  sum = first * second;
  printf("%d * %d = %d\n", first, second, sum);

  sum = first / second;
  printf("%d / %d = %d\n", first, second, sum);

  return 0;
}


/*    OUTPUT: arith_2.c

          11 + 12 = 23
          11 - 12 = -1
          11 * 12 = 132
          11 / 12 = 0

*/
```

Ex:

```
/*     FILE: arith_3.c     */

/* More arithmetic operators with nicer output */

#include <stdio.h>

int main( )
{
  int first, second, sum;

  first = 11;
  second = 12;

  sum = first + second;
  printf("%d + %d = %d\n", first, second, sum);

  sum = first - second;
  printf("%d - %d = %d\n", first, second, sum);

  sum = second - first;
  printf("%d - %d = %d\n", second, first, sum);

  sum = first * second;
  printf("%d * %d = %d\n", first, second, sum);

  sum = first / second;
  printf("%d / %d = %d\n", first, second, sum);

  return 0;
}


/*    OUTPUT: arith_3.c

          11 + 12 = 23
          11 - 12 = -1
          12 - 11 = 1
          11 * 12 = 132
          11 / 12 = 0

*/
```

Ex:

```
/*     FILE: arith_4.c     */

/* Arithmetic operators with floating-point data */

#include <stdio.h>

int main( )
{
  float first, second, sum;

  first = 11;
  second = 12;

  sum = first + second;
  printf("%f + %f = %f\n", first, second, sum);

  sum = first - second;
  printf("%f - %f = %f\n", first, second, sum);

  sum = second - first;
  printf("%f - %f = %f\n", second, first, sum);

  sum = first * second;
  printf("%f * %f = %f\n", first, second, sum);

  sum = first / second;
  printf("%f / %f = %f\n", first, second, sum);

  return 0;
}


/*    OUTPUT: arith_4.c

          11.000000 + 12.000000 = 23.000000
          11.000000 - 12.000000 = -1.000000
          12.000000 - 11.000000 = 1.000000
          11.000000 * 12.000000 = 132.000000
          11.000000 / 12.000000 = 0.916667

*/
```

Ex:

```
/*     FILE: arith_5.c     */

/* More arithmetic operators with floating-point data */

#include <stdio.h>

int main( )
{
  float first, second, sum;

  first = 1.35;
  second = 2.75;

  sum = first + second;
  printf("%f + %f = %f\n", first, second, sum);

  sum = first - second;
  printf("%f - %f = %f\n", first, second, sum);

  sum = second - first;
  printf("%f - %f = %f\n", second, first, sum);

  sum = first * second;
  printf("%f * %f = %f\n", first, second, sum);

  sum = first / second;
  printf("%f / %f = %f\n", first, second, sum);

  return 0;
}


/*    OUTPUT: arith_5.c

          1.350000 + 2.750000 = 4.100000
          1.350000 - 2.750000 = -1.400000
          2.750000 - 1.350000 = 1.400000
          1.350000 * 2.750000 = 3.712500
          1.350000 / 2.750000 = 0.490909

*/
```

Ex:

```
/*     FILE: arith_6.c     */

/* Precedence of operators */

#include <stdio.h>

int main( )
{
  int first, second, sum;

  first = 10;
  second = 12;

  sum = first + second / 3;
  printf("%d + %d / 3 = %d\n", first, second, sum);

  return 0;
}


/*    OUTPUT: arith_6.c

          10 + 12 / 3 = 14

*/
```

Ex:
```
/*     FILE: arith_7.c     */

/* Parentheses override precedence of operators */

#include <stdio.h>

int main( )
{
  int first, second, sum;

  first = 10;
  second = 12;

  sum = (first + second) / 3;
  printf("(%d + %d) / 3 = %d\n", first, second, sum);

  return 0;
}


/*    OUTPUT: arith_7.c

          (10 + 12) / 3 = 7

*/
```

Ex:
```
/*     FILE: computation.c     */

/* Computes the cost per sq inch of pizza
   -- inspired by pizza.c example in C
      Primer Plus by Prata                */

#include <stdio.h>

int main( )
{
  int diameter, radius, area, price, pricePerInch;

  printf("What is the price of your pizza: ");
  scanf("%d", &price);

  printf("What is the diameter of your pizza: ");
  scanf("%d", &diameter);


  radius = diameter/2;
  area = 3.14159 * radius * radius;
  pricePerInch = price/area;

  printf("Pizza analysis:\n");
  printf("    diameter = %d\n", diameter);
  printf("      radius = %d\n", radius);
  printf("        area = %d\n", area);
  printf("       price = %d per sq. inch\n", pricePerInch);

  return 0;
}


/*    OUTPUT: computation.c

          What is the price of your pizza: 10.50
          What is the diameter of your pizza:
          Pizza analysis:
              diameter = 4206596
                radius = 2103298
                  area = -2147483648
                 price = 0 per sq. inch


          What is the price of your pizza: 10
          What is the diameter of your pizza: 14
          Pizza analysis:
              diameter = 14
                radius = 7
                  area = 153
                 price = 0 per sq. inch

*/
```

Ex:

```
/*     FILE: computation2.c     */

/* Computes the cost per sq inch of pizza

   Uses a float for price, to get dollars
   and cents.
                                          */

#include <stdio.h>

int main( )
{
  int diameter, radius, area, pricePerInch;
  float price;

  printf("What is the price of your pizza: ");
  scanf("%f", &price);

  printf("What is the diameter of your pizza: ");
  scanf("%d", &diameter);


  radius = diameter/2;
  area = 3.14159 * radius * radius;
  pricePerInch = price/area;

  printf("Pizza analysis:\n");
  printf("    diameter = %d\n", diameter);
  printf("      radius = %d\n", radius);
  printf("        area = %d\n", area);
  printf("       price = %d per sq. inch\n", pricePerInch);

  return 0;
}


/*    OUTPUT: computation2.c

          What is the price of your pizza: 10.50
          What is the diameter of your pizza: 14
          Pizza analysis:
              diameter = 14
                radius = 7
                  area = 153
                 price = 0 per sq. inch

*/
```

Ex:

```
/*     FILE: computation3.c     */

/* Computes the cost per sq inch of pizza

   More floating-point.
                                           */

#include <stdio.h>

int main( )
{
  int diameter;
  float price, radius, area, pricePerInch;

  printf("What is the price of your pizza: ");
  scanf("%f", &price);

  printf("What is the diameter of your pizza: ");
  scanf("%d", &diameter);


  radius = diameter/2;
  area = 3.14159 * radius * radius;
  pricePerInch = price/area;

  printf("Pizza analysis:\n");
  printf("    diameter = %d\n", diameter);
  printf("      radius = %f\n", radius);
  printf("        area = %f\n", area);
  printf("       price = %.2f per sq. inch\n", pricePerInch);

  return 0;
}


/*    OUTPUT: computation3.c

          What is the price of your pizza: 10.50
          What is the diameter of your pizza: 14
          Pizza analysis:
              diameter = 14
                radius = 7.000000
                  area = 153.937912
                 price = 0.07 per sq. inch


          What is the price of your pizza: 15.50
          What is the diameter of your pizza: 18
          Pizza analysis:
              diameter = 18
                radius = 9.000000
                  area = 254.468796
                 price = 0.06 per sq. inch


          What is the price of your pizza: 15.50
          What is the diameter of your pizza: 19
          Pizza analysis:
              diameter = 19
                radius = 9.000000
                  area = 254.468796
                 price = 0.06 per sq. inch

*/
```

Ex:

```
/*     FILE: computation4.c     */

/* Computes the cost per sq inch of pizza

   A type cast.
                                           */

#include <stdio.h>

#define PI 3.14159

int main( )
{
  int diameter;
  float price, radius, area, pricePerInch;

  printf("What is the price of your pizza: ");
  scanf("%f", &price);

  printf("What is the diameter of your pizza: ");
  scanf("%d", &diameter);


  radius = (float)diameter/2;
  area = PI * radius * radius;
  pricePerInch = price/area;

  printf("Pizza analysis:\n");
  printf("    diameter = %d\n", diameter);
  printf("      radius = %f\n", radius);
  printf("        area = %f\n", area);
  printf("       price = %.2f per sq. inch\n", pricePerInch);

  return 0;
}


/*    OUTPUT: computation4.c

          What is the price of your pizza: 15.50
          What is the diameter of your pizza: 18
          Pizza analysis:
              diameter = 18
                radius = 9.000000
                  area = 254.468796
                 price = 0.06 per sq. inch



          What is the price of your pizza: 15.50
          What is the diameter of your pizza: 19
          Pizza analysis:
              diameter = 19
                radius = 9.500000
                  area = 283.528503
                 price = 0.05 per sq. inch

*/
```

INCREMENT ++/DECREMENT -- OPERATORS

- C has two specialized operators for incrementing and decrementing the value of a variable.

++	- will increase a variables value by “one”
--	- will decrease a variables value by “one”

- Both operators can be written in both prefix and postfix notation. Each has implications as to when the actual increment or decrement takes place. Fortunately the implications are reasonable. Prefix notation causes the increment/decrement to occur “before” the value of the variable is supplied to an expression. Postfix notation causes the increment/decrement to occur “after” the value of the variable is supplied to an expression. In all cases the variables value is increased/decreased by “one”

Ex:

```
/*     FILE: incDec.c     */

/* Example of increment & decrement, postfix and prefix. */

#include <stdio.h>

int main( )
{
  int i =7;

  printf("i = %d\n", i++);
  printf("After postfix ++, i = %d\n", i);

  printf("i = %d\n", ++i);
  printf("After prefix ++, i = %d\n", i);

  printf("i = %d\n", i--);
  printf("After postfix --, i = %d\n", i);

  printf("i = %d\n", --i);
  printf("After prefix --, i = %d\n", i);

  return 0;
}

/*    OUTPUT: incDec.c

          i = 7
          After postfix ++, i = 8

          i = 9
          After prefix ++, i = 9

          i = 9
          After postfix --, i = 8

          i = 7
          After prefix --, i = 7

*/
```

INDEX