Getting Started

C Language Features

- Compiled
- Hi/Low Level
- Efficient
- Portable
- Flexible
- Modular

Minimal C Program

program.c

```
main()
{

}
```

```
main(){}
```

```
main
(
)
{
}
```

- Filename ends with ".c"
- White space independent
- Token, not column oriented

Sample Program 1

```
 1  /*  This program prints a
 2      welcome message three times */
 3
 4  #include    <stdio.h>
 5
 6  main()
 7  {
 8      int     count;  /* counter variable */
 9
10      count = 1;
11      while  (count <= 3) {
12          printf("Welcome to the course!\n");
13          count = count + 1;
14          }
15  }
```

NOTE: The line numbers shown above are for reference purposes only.

Output:

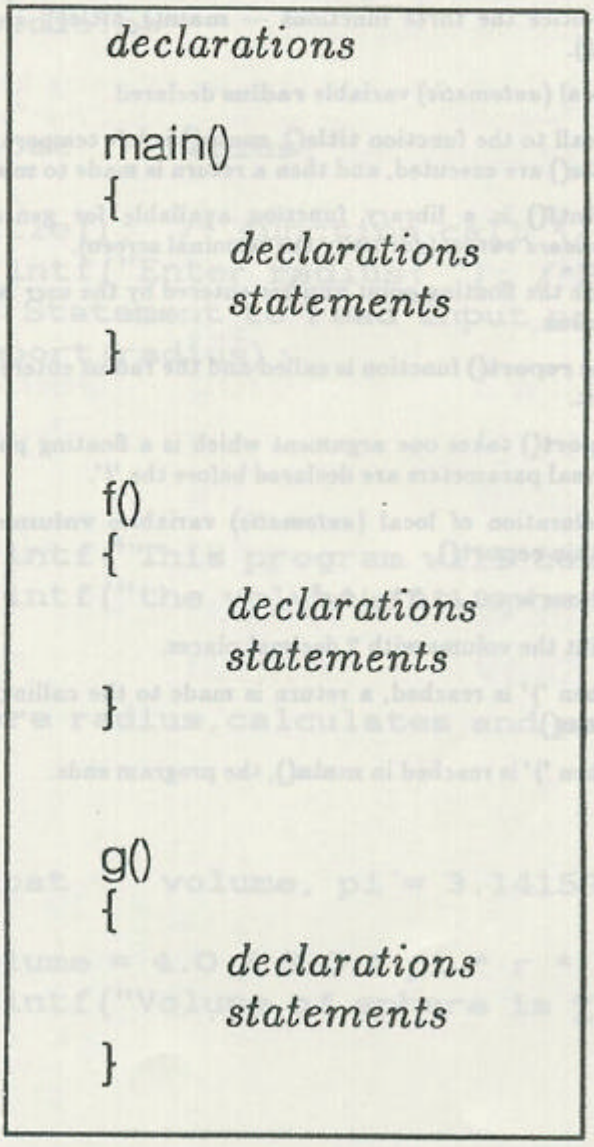
```
Welcome to the course!
Welcome to the course!
Welcome to the course!
```

Comments

- Important for readability, documentation
- Begin with /* and end with */
- May be anywhere in program, may span lines
- Can not nest

```
/*  This is a comment */

/*  This is a comment
that spans several
lines  */

/*  Another
    style of
    commenting
*/

/*  and yet       */
/*  another style */


/* This is a /* common */ error. */
```

Generic C Program



Sample Program 2

```
1   /* Program to calculate the volume of a sphere. */
2   #include   <stdio.h>
3   main()
4   {
5               float    radius;
6
7               title();  /* function call */
8               printf("Enter radius: "); /*Prompt user*/
9               /* Statement to read input not shown */
10              report(radius);
11  }
12
13  title()
14  {
15              printf("This program will compute ");
16              printf("the volume of a sphere.\n");
17  }
18
19  /* Given sphere radius,calculates and prints volume */
20  report(r)
21  float    r;
22  {
23                 float    volume, pi = 3.14159;
24
25                 volume = 4.0 / 3.0 * pi * r * r * r;
26                 printf("Volume of sphere is %.2f\n",volume)
27  }
```

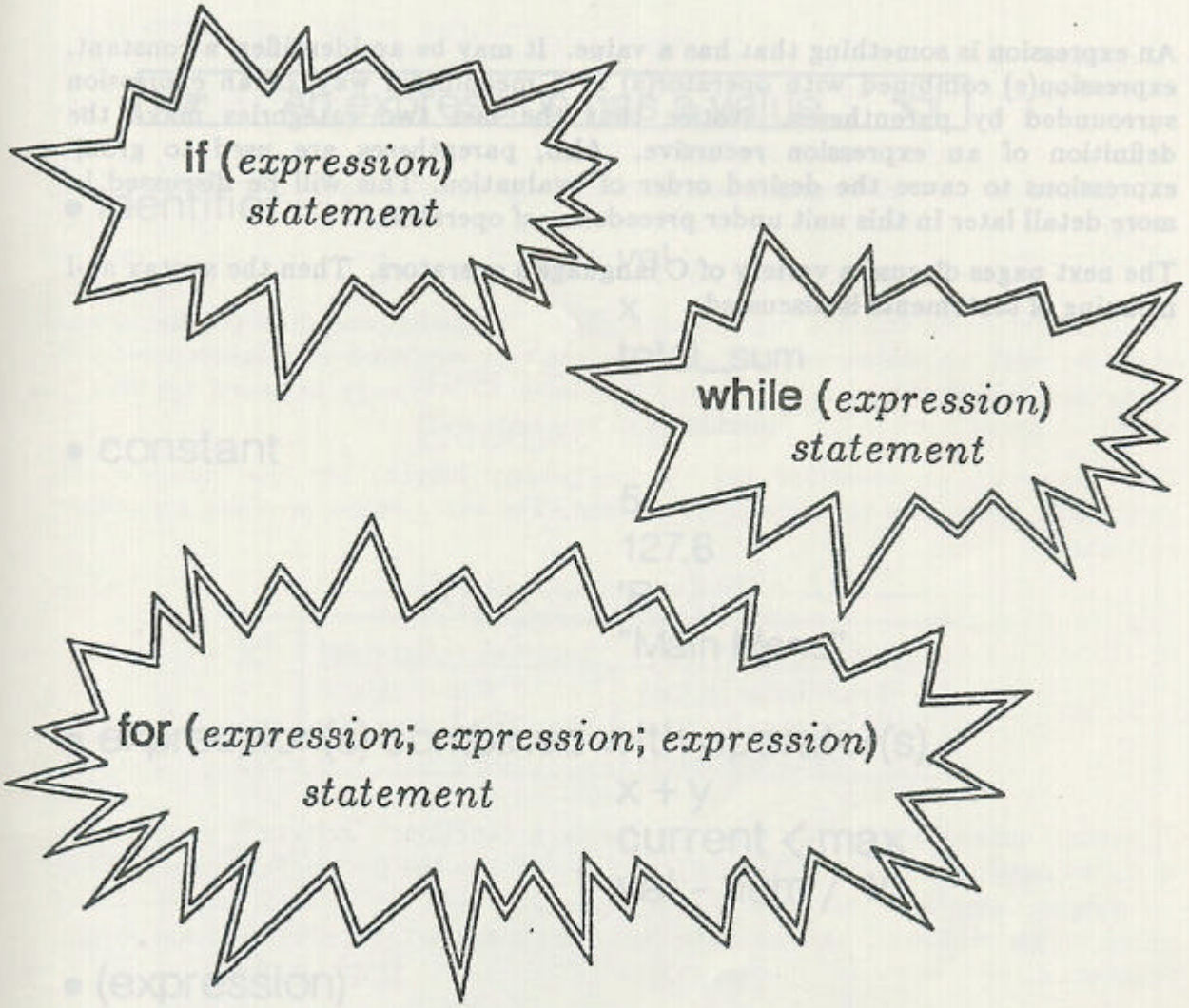
Output:

```
This program will compute the volume of a sphere.
Enter radius: 57.5
Volume of sphere is 796327.63
```

C Language Keywords

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

Expressions, Operators, and Statements



What is an expression?

What is a statement?

Expressions

☞ An expression has a value ☜

- identifier

  val
  x
  total_sum

- constant

  5
  127.6
  'B'
  "Main Menu"

- expression(s) combined with operator(s)

  x + y
  current < max
  val - num / 10

- (expression)

  (x + y)
  (current < max)

Identifiers

- Letters, digits, underscores
- Cannot begin with a digit
- Cannot be a keyword
- Significant Characters (ANSI Standard):

Internal: 31
External: 6

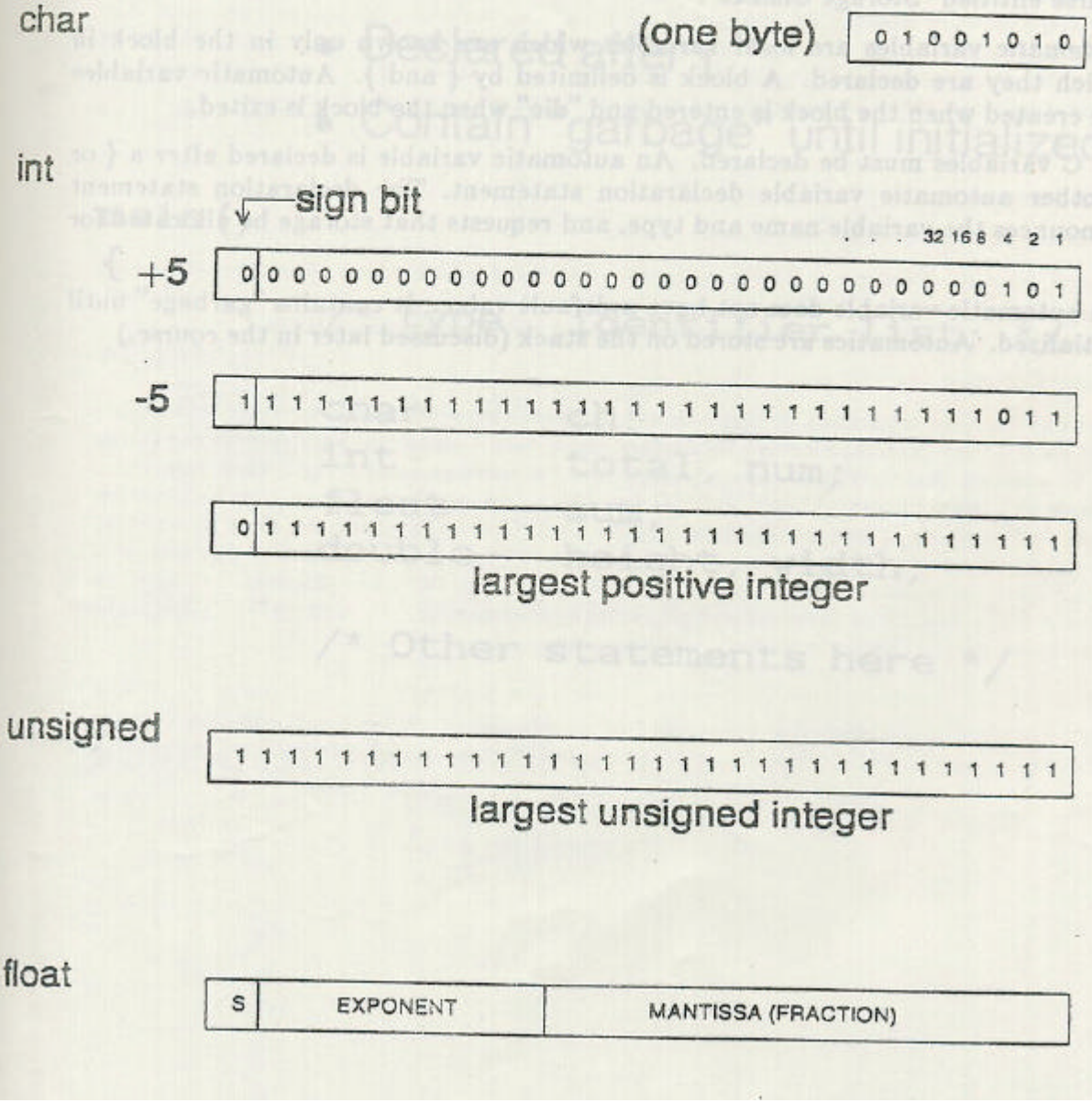
Valid or Invalid?	
x	num_parts
6x	part5
VAL	table.index
PrevAmt	$pay
employee_name	employee_pay

Arithmetic Data Types

		Typical Sizes	
		Word Size	
		16-bit	32-bit
char	*1 byte*	8	8
int	*integer; wordsize*	16	32
unsigned		16	32
short		16	16
long		32	32
float	*single precision*	32	32
double	*double precision*	64	64

Storage of Arithmetic Data Types

Hypothetical Machine



Declaration of Automatic Variables

- All C variables must be declared
- Automatic variables:
  - Local - block scope
  - Declared after {
  - Contain "garbage" until initialized

```
main()
{
        /* type   identifier list; */

        char     ch;
        int      total, num;
        float    sum;
        double   height, width;

        /* Other statements here */

}
```

Constants

•Integer

Decimal	55	100	255
Octal	067	0144	0377
Hex	0x37	0x67	0Xff
Explicit Long	55L	100l	255L

•Floating Point

5.32	5.	.75
25e-3		.240E100

•Character

Printable	'a'	'B'	'+'	'5'
Special	'\n'			newline
	'\t'			horizontal tab
	'\b'			backspace
	'\0'			null byte
Bit pattern	'\007'	'\012'		'\377'

•String Literal - delimited by '\0'

"Please enter your first name"
"Annual Report"

ASCII Table - Portable 7 bit code

Binary	Oct	Hex	Dec	Ch	Binary	Oct	Hex	Dec	Ch	Binary	Oct	Hex	Dec	Ch
00000000	0000	0x00	0	nul	00101011	0053	0x2b	43	+	01010110	0126	0x56	86	V
00000001	0001	0x01	1	soh	00101100	0054	0x2c	44	,	01010111	0127	0x57	87	W
00000010	0002	0x02	2	stx	00101101	0055	0x2d	45	-	01011000	0130	0x58	88	X
00000011	0003	0x03	3	etx	00101110	0056	0x2e	46	.	01011001	0131	0x59	89	Y
00000100	0004	0x04	4	eot	00101111	0057	0x2f	47	/	01011010	0132	0x5a	90	Z
00000101	0005	0x05	5	enq	00110000	0060	0x30	48	0	01011011	0133	0x5b	91	[
00000110	0006	0x06	6	ack	00110001	0061	0x31	49	1	01011100	0134	0x5c	92	\
00000111	0007	0x07	7	bel	00110010	0062	0x32	50	2	01011101	0135	0x5d	93	]
00001000	0010	0x08	8	bs	00110011	0063	0x33	51	3	01011110	0136	0x5e	94	^
00001001	0011	0x09	9	ht	00110100	0064	0x34	52	4	01011111	0137	0x5f	95	_
00001010	0012	0x0a	10	nl	00110101	0065	0x35	53	5	01100000	0140	0x60	96	`
00001011	0013	0x0b	11	vt	00110110	0066	0x36	54	6	01100001	0141	0x61	97	a
00001100	0014	0x0c	12	np	00110111	0067	0x37	55	7	01100010	0142	0x62	98	b
00001101	0015	0x0d	13	cr	00111000	0070	0x38	56	8	01100011	0143	0x63	99	c
00001110	0016	0x0e	14	so	00111001	0071	0x39	57	9	01100100	0144	0x64	100	d
00001111	0017	0x0f	15	si	00111010	0072	0x3a	58	:	01100101	0145	0x65	101	e
00010000	0020	0x10	16	dle	00111011	0073	0x3b	59	;	01100110	0146	0x66	102	f
00010001	0021	0x11	17	dcl	00111100	0074	0x3c	60	<	01100111	0147	0x67	103	g
00010010	0022	0x12	18	dc2	00111101	0075	0x3d	61	=	01101000	0150	0x68	104	h
00010011	0023	0x13	19	dc3	00111110	0076	0x3e	62	>	01101001	0151	0x69	105	i
00010100	0024	0x14	20	dc4	00111111	0077	0x3f	63	?	01101010	0152	0x6a	106	j
00010101	0025	0x15	21	nak	01000000	0100	0x40	64	@	01101011	0153	0x6b	107	k
00010110	0026	0x16	22	syn	01000001	0101	0x41	65	A	01101100	0154	0x6c	108	l
00010111	0027	0x17	23	etb	01000010	0102	0x42	66	B	01101101	0155	0x6d	109	m
00011000	0030	0x18	24	can	01000011	0103	0x43	67	C	01101110	0156	0x6e	110	n
00011001	0031	0x19	25	em	01000100	0104	0x44	68	D	01101111	0157	0x6f	111	o
00011010	0032	0x1a	26	sub	01000101	0105	0x45	69	E	01110000	0160	0x70	112	p
00011011	0033	0x1b	27	esc	01000110	0106	0x46	70	F	01110001	0161	0x71	113	q
00011100	0034	0x1c	28	fs	01000111	0107	0x47	71	G	01110010	0162	0x72	114	r
00011101	0035	0x1d	29	gs	01001000	0110	0x48	72	H	01110011	0163	0x73	115	s
00011110	0036	0x1e	30	rs	01001001	0111	0x49	73	I	01110100	0164	0x74	116	t
00011111	0037	0x1f	31	us	01001010	0112	0x4a	74	J	01110101	0165	0x75	117	u
00100000	0040	0x20	32		01001011	0113	0x4b	75	K	01110110	0166	0x76	118	v
00100001	0041	0x21	33	!	01001100	0114	0x4c	76	L	01110111	0167	0x77	119	w
00100010	0042	0x22	34	"	01001101	0115	0x4d	77	M	01111000	0170	0x78	120	x
00100011	0043	0x23	35	#	01001110	0116	0x4e	78	N	01111001	0171	0x79	121	y
00100100	0044	0x24	36	$	01001111	0117	0x4f	79	O	01111010	0172	0x7a	122	z
00100101	0045	0x25	37	%	01010000	0120	0x50	80	P	01111011	0173	0x7b	123	{
00100110	0046	0x26	38	&	01010001	0121	0x51	81	Q	01111100	0174	0x7c	124	!
00100111	0047	0x27	39	'	01010010	0122	0x52	82	R	01111101	0175	0x7d	125	}
00101000	0050	0x28	40	(	01010011	0123	0x53	83	S	01111110	0176	0x7e	126	~
00101001	0051	0x29	41	)	01010100	0124	0x54	84	T	01111111	0177	0x7f	127	del
00101010	0052	0x2a	42	*	01010101	0125	0x55	85	U					

Exercise

Arithmetic Operators

		Examples
Additive		
+	add	x + y
–	subtract	num – 20
Multiplicative		
*	multiply	200 * val
/	divide	10/3 10.0/3.0
%	remainder	clock % 60
Unary Minus		
–		–x –(prev + current)

Relational Operators

<	less than	x < max
<=	less than or equal to	x <= max
>	greater than	x > max
>=	greater than or equal to	x >= max
==	equal to	x == max
!=	not equal to	x != max

☞ 0 is false, nonzero is true ☜

☞Value of relational expression is 0(false) or 1(true)☜

Logical Operators

Logical and

```
(x >= 1) && (x <= 500)
```

Logical or

```
(input == 'q') || (input == 'Q')
```

- left to right evaluation
- execution stops when truth value determined

Logical negation

```
!(x > y && y > z)
```

☞ 0 is false, nonzero is true ☜

☞ Value of logical expression is 0(false) or 1(true) ☜

Assignment Operator =

lvalue = *expression*

count = 1 /* valid */

(a – b) = 6 /* invalid */

```
1  main()
2  {
3      int    quantity;
4      float  total, price = 150.00;
5
6      quantity = 4;
7      total = quantity * price;
8      ...
9  }
```

op= Assignment Operators

- Concise and Efficient

expr1 op= expr2	*expr1 = expr1 op (expr2)*
x += 50	x = x + 50
x −=50	x = x − 50
x *= 50	x = x * 50
x /= 50	x = x / 50
x %= 50	x = x % 50
x *= a + b	x = x * (a + b)

Conversion by assignment

1. float or double = char or int

```
int_var = 27;
float_var = int_var;  /* float_var now 27.0 */
```

2. char or int = float or double

```
/* fraction is truncated */
/* data is lost if won't fit */

float_var = 27.9;
int_var = float_var;  /* int_var now 27 */
```

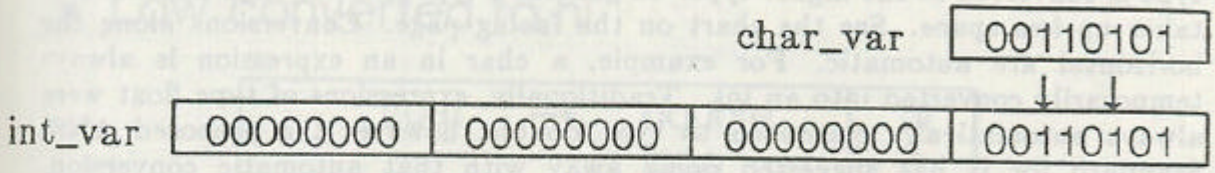
3. small = big

```
/*  data may be  lost */

char _var = int_var;
short_var = long_var;
```

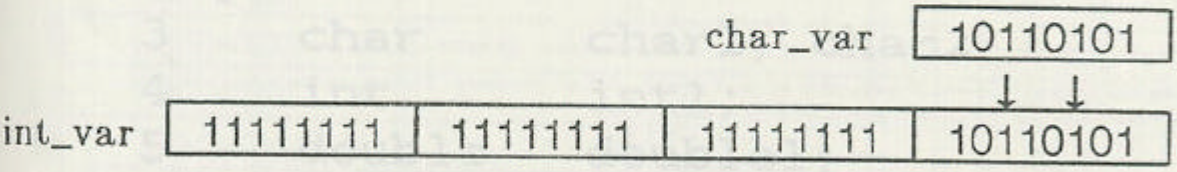
(Continued on next page)

Conversion by assignment, continued

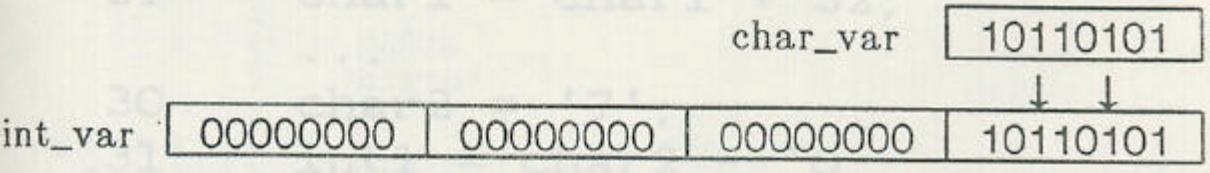
4. big = small



some machines sign extend:



others do not:



Automatic Conversions

- Mixed types OK (e.g. int * float + double)
- Low converted to hi

	float	→ double	hi
		long	↑
		unsigned	↑
	char, short	→ int	low

```
 1 main()
 2 {
 3     char      char1, char2;
 4     int       int1;
 5     double    double1;
       ...
10     double1 = double1 / int1;
       ...
20     char1 = 'J';
21     char1 = char1 + 32;
       ...
30     char2 = '7';
31     int1 = char2 - '0';
       ...
50 }
```

OPERATOR PRECEDENCE

TYPE	OPERATOR	ASSOCIATIVITY
PRIMARY	() [] -> .	L TO R
UNARY	! ~ - ++ -- (TYPE) * & sizeof	R TO L
MULTIPLICATIVE	* / %	L TO R
ADDITIVE	+ -	L TO R
SHIFT	<< >>	L TO R
RELATIONAL	< <= > >=	L TO R
EQUALITY	== !=	L TO R
BITWISE	&	L TO R
BITWISE	^	L TO R
BITWISE	|	L TO R
LOGICAL	&&	L TO R
LOGICAL	||	L TO R
CONDITIONAL	?:	R TO L
ASSIGNMENT	= op=	R TO L
COMMA	,	L TO R

- Parentheses - clarity, change precedence

	Value Assigned		Value Assigned
x = 5 + 3 * 2	_____	x = 12 / 2 * 3	_____
x = (5 + 3) * 2	_____	x = 12 / (2 * 3)	_____
a = b = 0	_____	done = (x<y && y<z)	_____

Statements

☞ A statement is an action. ☜

- simple

  expression;

```
count = count + 1;
temp = num;
;
```

- block

  {
  optional declarations
  statements
  }

```
{
temp = y;
y = x;
x = temp;
}
```

- flow control

  if, while, switch, ...

Expressions and Statements Summary

☞ An expression has a value ☜

- *constant*
- *identifier*
- *expression(s) combined with operator(s)*
- *(expression)*

☞ A statement is an action. ☜

• simple	*expression*;

• block	{
	optional declarations
	statements
	}

• flow control	if, while, switch ...

Basic I/O

printf(), getchar(), putchar()

Input / Output

- C does not have built-in I/O statements
- Standard I/O Library
- #include <stdio.h>

	Write	Read
formatted	printf()	*
character	putchar()	getchar()
line	*	*
record	*	*

* Will be covered in another unit.

Formatted Output

NAME

printf

SYNOPSIS

```
#include <stdio.h>
int  printf(format[,arg] ... )
```

DESCRIPTION

Prints output to standard stream *stdout* according to *format* which may contain literal characters and *conversion characters*.

```
%d   decimal integer
%f   floating point
```

EXAMPLES

```
printf("Please enter name: ");

printf("\n\t** DAILY REPORT **\n\n");

printf("Number in Stock: %d\n",num);

printf("Price: %f\n",price);

printf("  Total: $%.2f\n",count*price);
```

Character I/O

NAME	getchar	putchar
SYNOPSIS	#include <stdio.h> int getchar()	#include <stdio.h> int putchar(c) int c;
DESCRIPTION	Reads and returns next character from *standard input.*	Writes character *c* to *standard output.*
DIAGNOSTICS	Returns EOF* at end-of-file or error.	Returns value written if successful, else EOF*.

EXAMPLES

```
 1  #include <stdio.h>
 2  main()
 3  {  int  c;
 4
 5     c = getchar();   /*  read character */
 6     putchar(c);   /* write character */
 7     putchar('M');
 8     putchar('\t');
 9     putchar('\007');
10  }
```

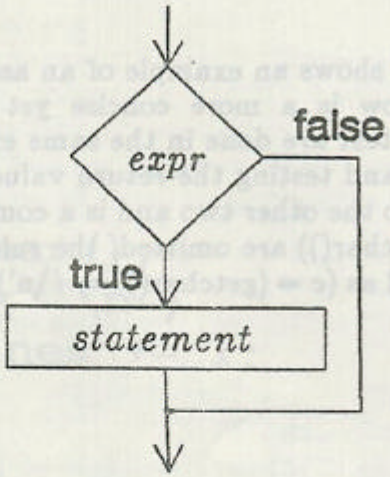
* EOF is defined in stdio.h; typical UNIX System value is -1

Basic Flow Control Statements

if, if-else, while

if statement

if (*expression*)
    statement




```
if ( x > largest )
    largest = x;
```

```
if (val < min || val > max) {
    printf("Value %d outside range.\n",val);
    printf("Range is %d-%d\n",min,max);
    }
```

if statement, continued

```
c = getchar();
if (c == '\n')
    lines += 1;
```

Equivalent:

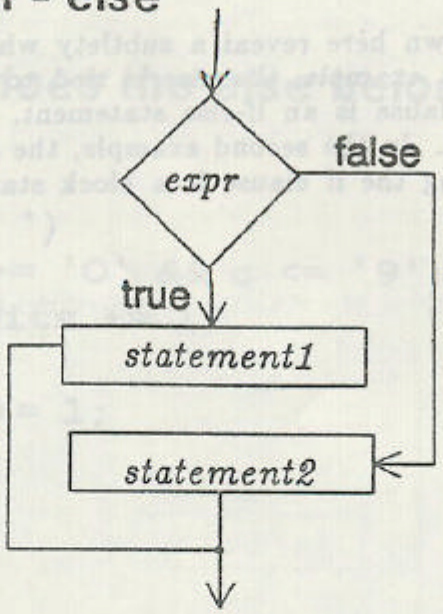
```
if ( (c = getchar()) == '\n')
    lines += 1;
```

Wrong - Why?

```
if ( c =(getchar() == '\n'))
    lines += 1;
```

if - else

```
if (expression)
      statement1
else
      statement2
```




```
if ( quantity < 1000 )
   printf("Stock low\n");
else
   printf("Stock OK\n");
```

```
if ( x != 0)
   y = y/x;
else {
   printf("Error--");
   printf("Divide by 0\n");
   y = 0;
   }
```

```
if ( hour >= 8 && hour < 17 )
   rate *= 1.02;
else if ( hour >= 17 && hour < 23 )
      rate *= 1.01;
```

Dangling else problem

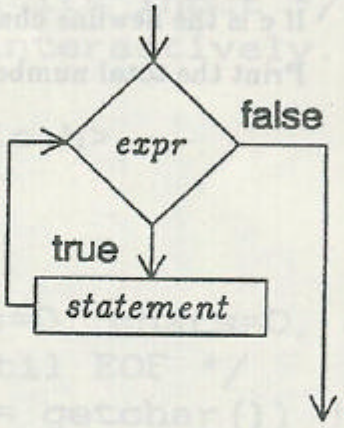
To which if does the else belong?

```
if (c > ' ')
   if (c >= '0' && c <= '9')
      digits += 1;
else
   count += 1;
```

```
if (c > ' ')
   {
   if (c >= '0' && c <= '9')
      digits += 1;
   }
else
   count += 1;
```

while statement

while (*expression*)
statement




```
count = 1;
while (count < 100) {
      statement
      statement
      count += 1;
      }
```

```
while ((c = getchar()) != '\n') {
      statement
      statement
      statement
      }
```

Sample Program

```
 1  /* This program counts lines */
 2  /*  and chars of its input */
 3  /* May be used interactively. */
 4
 5  #include    <stdio.h>
 6
 7  main()
 8  {
 9        int   lines=0, chars=0, c;
10        /* Read until EOF */
11        while ((c = getchar()) != EOF){
12              chars += 1;
13              if ( c == '\n')
14                    lines += 1;
15              }
16        printf("%d lines\n",lines);
17        printf("%d characters\n",chars);
18  }
```

Exercise
What's Wrong Here?

```
a = 0;
while ( a < 10 )
        a = a + 1;
        b = b + a;
```

```
a = 0;
while ( a < 10 ) ;
        {
        a = a + 1;
        b = b + a;
        }
```

```
/* Discard new-line characters */
input = getchar();
while (input = '\n') {
        input = getchar();
        c = c + 1;
        }
```

C Compiler

- Source file name ends with ".c"
- Compiler creates executable file, error messages
- UNIX System compiler - cc(1)

On a UNIX System:

```
$ vi prog.c          Create program with an editor

$ cc prog.c          Compile program

$ a.out              Execute program
```

To change name of executable file:

```
$ mv a.out newname
$ newname
        - OR -
$ cc file.c -o newname
$ newname
```