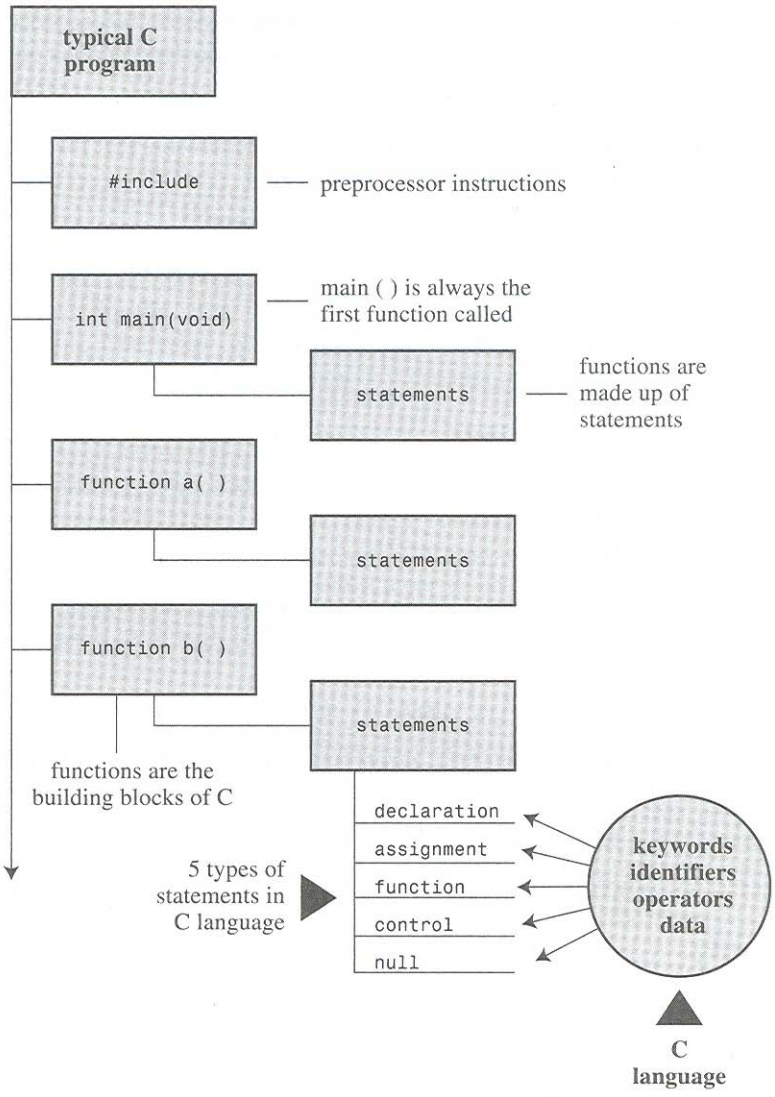
FIGURE 2.1
Anatomy of a C program.



Hello.c

```
/*
gcc -Wall %1.c -o %1.exe          Everything between /*
                                  and */ is a comment
   */
#include <stdio.h>                Preprocessor statement
int main(void){                   Main function, begin
     printf("\nHello world");     function call to screen
     return(0);                   value returned to OS
}                                 end
```

The *function prototype* for printf in <stdio.h>:

```
int printf(const char * restrict, ...)
```

fahrenheit.c

```
/*  Program to Convert Fahrenheit to Celsius Degrees
       and write them out to the screen and a file

    Modified from the listing in Kernighan & Ritchie p. 9
Preprocessor Statement    */
#include <stdio.h>
/* File Function, pointer to file */
FILE *kp;
int main(void)
{
/* File open, name & write */
        kp=fopen("fahrenheit.dat","w");
/* int is a keyword and a variable type */
        int fahr, celsius;
        int lower, upper, step;

        lower = 0;    /* lower limit of temperature table */
        upper = 110;  /* upper limit of temperature table */
        step = 5;     /* step size */

        fahr = lower;
/* Same as R */
        while (fahr <= upper)
/* Begin Loop */
        {
                celsius = 5*(fahr-32)/9;
                printf("%6d %6d\n",fahr,celsius);
                fprintf(kp,"%6d %6d\n",fahr,celsius);
                fahr = fahr + step;
        }
/* end loop */
/* file close */
        fclose(kp);
        return(0);
}
```

Fahrenheit_real.c

```
/*  Program to Convert Fahrenheit to Celsius Degrees
      and write them out to the screen and a file
    Modified from the listing in Kernighan & Ritchie p. 9

Preprocessor Statement    */
#include <stdio.h>
/* File Function, pointer to file */
FILE *kp;
int main(void)
{
/* File open, name & write */
        kp=fopen("fahrenheit.dat","w");
/* switch to real numbers */
        double fahr, celsius;
        double lower, upper, step;

        lower = 0;    /* lower limit of temperature table */
        upper = 110;  /* upper limit of temperature table */
        step = 5;     /* step size */

        fahr = lower;
        while (fahr <= upper)
        {
                celsius = 5*(fahr-32)/9;
/* Print real numbers */
                printf("%10.0f %10.4f\n",fahr,celsius);
                fprintf(kp,"%10.0f %10.4f\n",fahr,celsius);
                fahr = fahr + step;
        }
        fclose(kp);
        return(0);
        }
}
```

Table 2-1	C Language Keywords		
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Appendix E: C Language Variable Types

Type	*Value Range*			*Comments*
char	–128	to	127	
unsigned char	0	to	255	
int	–32,768	to	32,767	16-bit
	–2,147,483,648	to	2,147,483,647	32-bit
unsigned int	0	to	65,535	16-bit
	0	to	4,294,967,295	32-bit
short int	–32,768	to	32,767	
unsigned short int	0	to	65,535	
long int	–2,147,483,648	to	2,147,483,647	
unsigned long int	0	to	4,294,967,295	
float	1.17×10^{-38}	to	3.40×10^{38}	6-digit precision
double	2.22×10^{-308}	to	1.79×10^{308}	15-digit precision

Some compilers treat the `char` type as signed, and others treat it as unsigned. To be sure, use the `signed` or `unsigned` prefix if you absolutely need a signed or unsigned `char` variable.

The size of the basic integer depends on the microprocessor. For most modern microprocessors, a 32-bit width is used. Older computers used only a 16-bit width. To ensure a 16-bit value, define your `int`s as `short`. To ensure the 32-bit value, use `long int`s.

TABLE 4.3 Conversion Specifiers and the Resulting Printed Output

Conversion Specification	Output
%a	Floating-point number, hexadecimal digits and p-notation (C99).
%A	Floating-point number, hexadecimal digits and P-notation (C99).
%c	Single character.
%d	Signed decimal integer.
%e	Floating-point number, e-notation.
%E	Floating-point number, E-notation.
%f	Floating-point number, decimal notation.
%g	Use %f or %e, depending on value. The %e style is used if the exponent is less than -4 or greater than or equal to the precision.
%G	Use %f or %E, depending on value. The %E style is used if the exponent is less than -4 or greater than or equal to the precision.
%i	Signed decimal integer (same as %d).
%o	Unsigned octal integer.
%p	A pointer.
%s	Character string.
%u	Unsigned decimal integer.
%x	Unsigned hexadecimal integer, using hex digits 0f.
%X	Unsigned hexadecimal integer, using hex digits 0F.
%%	Print a percent sign.

Vector.c

```
/*
Illustrates the use of vectors
   */
#include <stdio.h>
int main(void){
/* declare an array of length 4 */
        double numlist[4];
        int i;
/* set the values  */
        numlist[0] = 3;
        numlist[1] = 4;
        numlist[2] = 5;
        numlist[3] = 10;

/* Simple Write to Screen*/

        printf("\n numlist[0] == %f", numlist[0]);
        printf("\n numlist[1] == %f", numlist[1]);
        printf("\n numlist[2] == %f", numlist[2]);
        printf("\n numlist[3] == %f", numlist[3]);

/* While Loop */
/* you need to initialize i */

        i=0;
        while(i < 4)
        {
                printf("\n %10.4f",numlist[i]);
/* old fashioned increment */
                i = i + 1;
        }

/* While Loop With ++ increment operator -- ++i same as i=i+1 */
        i=0;
        while(i < 4)
        {
                printf("\n Nerd %10.4f",numlist[i]);
                ++i;
        }
        printf("\n\n\n");

/* For Loop With ++ increment operator */
/* for loop -- for(initialize; test; update) */
/* loop begins with { */
        for(i=0;i<4;i++)
        {
                printf("Nerd For %10.4f\n",numlist[i]);
        }
/* end for loop */
                printf("\n\nWhy will numlist[4] give an error?");

        return(0);
}
```

Random_test.c

```
/* Illustrates Pseudo-Random Number Generation
     in the Range of 0 to 1
     srand() and rand() are part of the
     general utilities defined in stdlib.h
     srand() with an integer argument sets the
     random number seed and rand() draws a
     pseudo-random number between 1 and RAND_MAX
 */
#include <stdlib.h>
#include <stdio.h>

int main(void){

        double temp;
        int i;
        srand(17);
/* check operating system limit value, %u = unsigned integer */
         printf("RAND_MAX=%u\n", RAND_MAX);
        for(i=0;i<10;i++){
/* convert RAND_MAX into double*/
                temp = rand()/((double)RAND_MAX + 1);
                printf("rand = %f\n",temp);
        }
         return(0);
}
```

Array.c

```
/*
Illustrates the use of both an array/matrix and a vector
  to store data.
   */
#include <stdlib.h>
#include <stdio.h>

int main(void){
/*  This somewhat awkward syntax is a matrix in C */
/*  C is organized around vectors so that matrices need not be used*/
        double randMat[3][3];
        double temp[9];
        int i, j, k;
        srand(16);
        printf("RAND_MAX=%10d\n", RAND_MAX);
/* Store random numbers in a matrix and a vector -- print out matrix  */
        k = 0;
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        randMat[i][j] = rand()/((double)RAND_MAX + 1);
                        printf("\nrandMat[%d][%d] = %f", i,j,randMat[i][j]);
                        temp[i+j*3] = randMat[i][j];
                        k = k + 1;
                }
        }
/* Write total number of Random numbers drawn */
        printf("\n\nNumber of Random Numbers Drawn %7d",k);

/* Print out Vector  */

        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        printf("\ntemp %d %d = %f", i,j,temp[i+j*3]);
                }
        }

        return(0);

}
```

How temp[] is Storing the Data:

$$\begin{bmatrix} i=0,\ j=0 \\ i=1,\ j=0 \\ i=2,\ j=0 \\ i=0,\ j=1 \\ i=1,\ j=1 \\ i=2,\ j=1 \\ i=0,\ j=2 \\ i=1,\ j=2 \\ i=2,\ j=2 \end{bmatrix}$$

Pointer_Simple.c

```
/* Pointer_simple.c -- Program illustrates fact that C works with both
 * the address of a variable and the value of the variable at the
 * address.  */

#include <stdio.h>

int main(void)
{
/* Declare a character, an integer, and a double
 * Note that none are initialized so that there is no
 * "value" at the address of the variables! */
        char c;
        int i;
        double x;
/* %p = hex, %c = character */
        printf("c: address=%p, content=%10c\n", &c, c);
        printf("i: address=%p, content=%10d\n", &i, i);
        printf("x: address=%p, content=%10.4f\n", &x, x);
/* Now, set the character, integer, and double variables
 * to specific values and write them out again */
        c = 'A';
        i = 8;
        x = 123.45;
        printf("c: address=%p, content=%10c\n", &c, c);
        printf("i: address=%p, content=%10d\n", &i, i);
        printf("x: address=%p, content=%10.4f\n", &x, x);
         return(0);
}
```

Pointer_declaring.c

```
/* Pointer_declaring.c -- Program shows how pointers are declared and
 * used to point to memory locations   */

#include <stdio.h>

int main(void)
{
/* Declare a character, an integer, and a double variable,
 * and a charter pointer, an integer pointer, and a double pointer
 */
         char c, *ptr_c;
         int i, *ptr_i;
         double x, *ptr_x;
/* Now, set the character, integer, and double variables
 * to specific values and write them out */
         c = 'A';
         i = 8;
         x = 123.45;
         printf("c: address=%p, content=%10c\n", &c, c);
         printf("i: address=%p, content=%10d\n", &i, i);
         printf("x: address=%p, content=%10.4f\n", &x, x);
/* Now, aim the pointers at the memory locations */
         ptr_c = &c;
         ptr_i = &i;
         ptr_x = &x;
/* Write them out again using the pointers -- Note that a pointer is
 * pointing to an address but it also has an address!!! */
         printf(" The address of ptr_c is %p, and the content is %p\n", &ptr_c, ptr_c);
/* Write out "dereferenced" pointer = value of variable */
         printf(" Dereferenced pointer *ptr_c => %c, c = %c\n", *ptr_c, c);

         printf(" The address of ptr_i is %p, and the content is %p\n", &ptr_i, ptr_i);
         printf(" Dereferenced pointer *ptr_i => %10d, i = %10d\n", *ptr_i, i);

         printf(" The address of ptr_x is %p, and the content is %p\n", &ptr_x, ptr_x);
         printf(" Dereferenced pointer *ptr_x => %10.4f, x = %10.4f\n", *ptr_x, x);

        return(0);
}
```

Control.c

```
/*
Illustrates various if-then-else branching
methods and the power function in C
   */
#include <stdio.h>
/* Miscellaneous math functions */
#include <math.h>
int main(void){

        int a=4;
        int b=10;
        double temp[10];
        double xa, xi;
        int i, j;
/* start, if-then-else branch structure */
        if(b == a){
                printf("\nb is exactly equal to a");
                j = 1;
        }
/* If not exactly equal, goes to else if */
        else if(b < a){
                printf("\nb is less than a");
                j = 2;
        }
/* If not less than, drops to else */
        else {
                printf("\na is less than b");
                j = 3;
         }
/* This is an error catch in case a > b  */
         if(j == 3){
             while(a != b){
                 a=a+1;
                 printf("\na equals %i", a);
             }
        }
/* Do type conversion -- transfer integer "a" to double "xa" */
        xa = a;
        for(i=0;i<10;i++){
/* Do type conversion -- transfer integer "i" to double "xi" */
                xi = i;
/* Power function -- xa**xi -- Part of the Math.h functions -- types
         are double so temp[i] must also be double */
                temp[i] = pow(xa,xi);
        }

        for(i=0;i<10;i++){
                printf("\nElement %i in temp = %15.3f", i, temp[i]);
        }
return(0);
}
```