More operator and control flow

Functions

- Introduction
- Arguments and Parameters
- Call by Value
- Real World Application: Computing Resistance
- The Scope of Variables

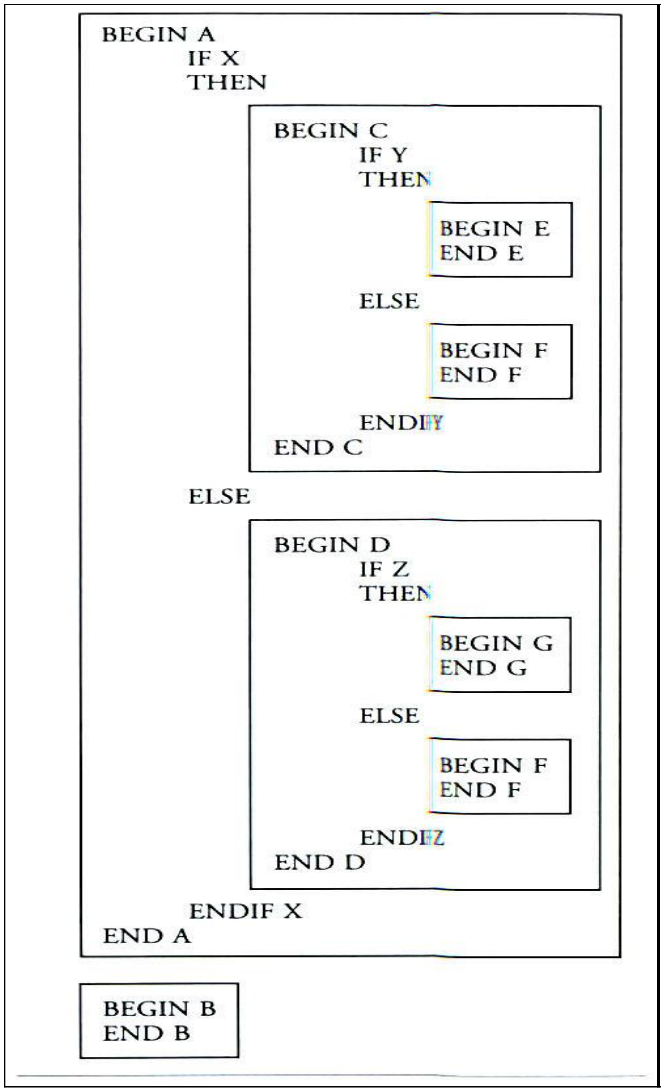
functions

- main()
  - must be present
  - where execution begins
  - program terminates when main ends
- other functions

Necessity for function

1. Divide the program into small sections, everyone works on one part

2. repetition



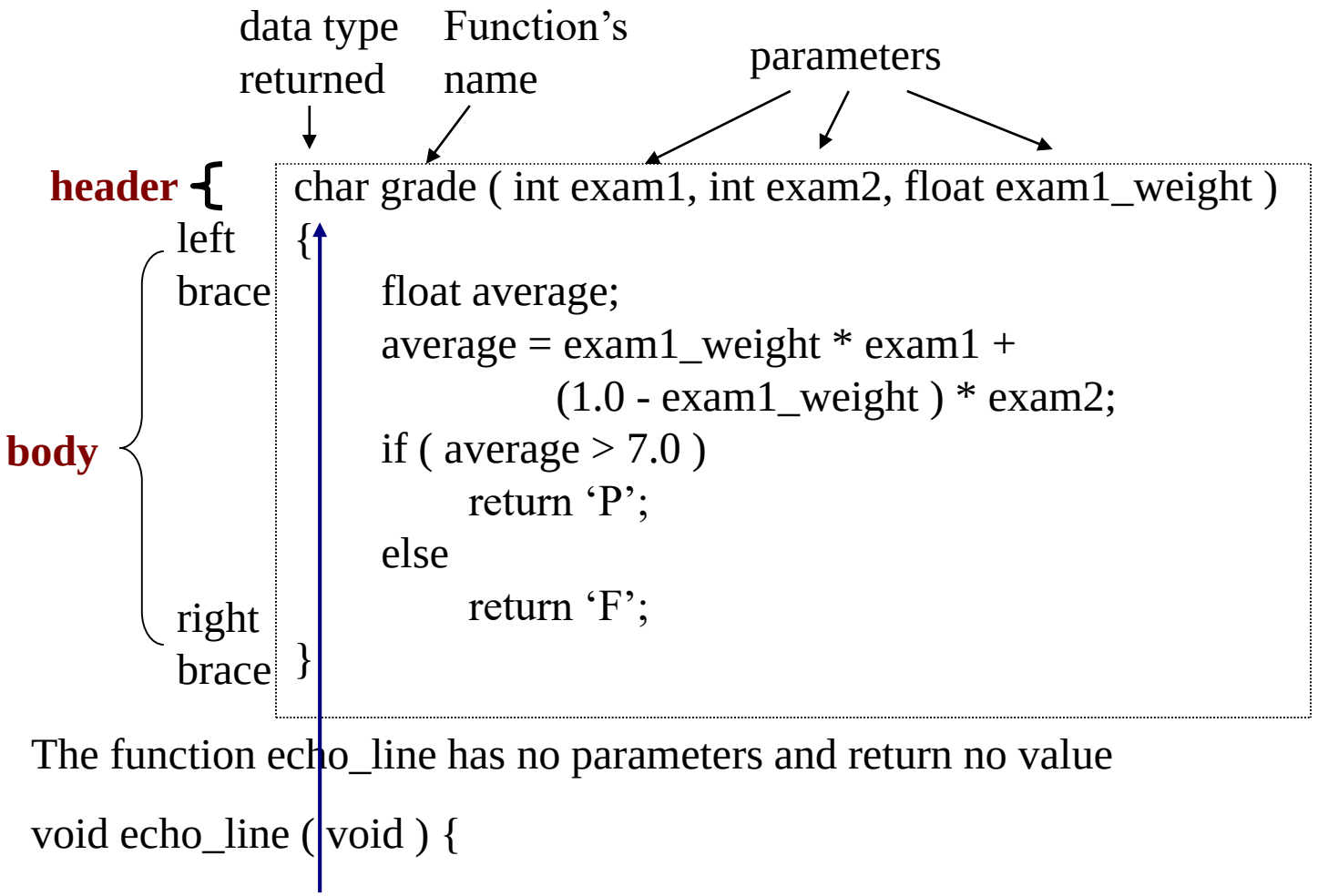
FUNCTION

- Header
  - int gamma(int x,float y,double z)
  - float alpha(char u)
  - char abc(char w)
  - void beta(float t,float u)
  - float prin(void)
  - spd(float x,char y) ⟶ int spd(float x,char y)
- Body

  {

  …

  }



The function echo_line has no parameters and return no value

```
void echo_line ( void ) {

}
```

Optional, if ignored, return as integer

return statement

- main() calls (invokes) a function *f*
- function *f* runs and then returns to main
- statement:
  - return;
  - return x;
  - return (exprn);
  - return (12 + 6 +fun1(m));
  - return x1,x2;
    - returns x2
    - a function may contain several return statements

Declaration and definition

- Declaration
  - char grade(int a,int b, int c);
- Definition

```
char grade(int a,int b,int c)
{
…

}
```

- Each function's definition must be in its own file ???

Main with returned status

```
main()
{
…
return EXIT_SUCCESS;
}
```

status may be used by operating system

```
#include <stdio.h>
char grade ( int exam1, int exam2, float exam1_weight );/* function- */
main ( ) {
   int ans, mid_term, final;                          /*declaration */
   float weight;
   char letter_grade;
   …
   letter_grade = grade ( mid_term, final, weight ); /* invoke function */
   }
   char grade (  int x1, int x2, float x3 )
   {

   ...
   return;
   }
```

Note the ;

Note: NO ;

- In a declaration, the names that follow the data types of the parameters are optional and are ignored by the compiler. The names need not be the same as the names of the parameters.
  In a definition, the parameters must always be named.

  ```
  char grade (int, int, float);
  ```

  or

  ```
  char grade ( int mid, int fin, float wt );
  ```

  serves as a declaration of grade.
  Caution:
- Arguments and parameters of functions must match The compiler can use a function declaration to check for matches between arguments and parameters of functions and issue appropriate warning and error messages if it detects problems.

  ```
  char grade ( int, int, int );/*error: Type mismatch in redeclaration of ‘grade’*/
  ```
- Every function except **main** should be declared because, if a function is invoked without being declared, the system will assume (possibly incorrectly) that it returns an int, and no checking for matches of parameters and arguments will occur.

- The main Function

  Every program must contain a function called **main** where execution of the program begins.

  The function **main** has no declaration, and its type is implementation-dependent.

```
main ( ) {        /* every implementation will support */
        …
}
void main ( ) {/* may not be supported by a particular implementation */
        …
}
```

  by default, main return an int. The return statement

```
return status;
```

  status → { EXIT_SUCCESS, EXIT_FAILURE } → defined in stdlib.h

- Functions in Sources Files

  In this chapter, we assume that all of the functions that make up a program are in one file.

Caution:
One function's definition cannot occur in another function's body.

```
void fun1 ( int I ) {
        …
        int fun2 ( char c ) { /* Illegal */
                …
        }
}
```

- Functions and Program Design
  Advantages:
  - by decomposing a program into functions, we can divide the work among several programmers. Then it is even possible for the programmers to write the functions at different time.
  - We can test one function separately from the rest;
  - We can change one function without changing other parts of the program;
  - We can make the program more readable by delegation intricate or otherwise specialized tasks to the appropriate functions.

Rewrite the bar graph program of chapter 3.

```
#include <stdio.h>
#include <stdlib.h>
int echo_line ( void );
void print_stars ( int val );
main ( ) {
        int value;                          /* number of stars - volume */
        while ( echo_line ( ) != EOF ) { /* get name and print in standard output */
                scanf ( "%d", &value );
                print_stars ( value );      /* print stars - print volume */
        }
        return EXIT_SUCCESS;
}
int echo_line ( void ) {
        char c;
        if ( scanf ( " %c", &c ) == EOF ) /* reading from standard input */
                return EOF;
```

```
        for ( ; ; ) {
                putchar ( c );
                if ( c == '\n' )
                        return c;
                c = getchar ( );        /* reading from standard input */
        }
}
void print_stars ( int vol ) {
        int I;
        for ( I = 1; I <= vol; I++)
                putchar ( '*' );
        printf ( "\n\n" );
}
```

Exercises

- (T/F) If C program has main and f1 functions, it is possible that f1 invokes main
- (T/F) An argument and its corresponding parameter must have the same name
- (T/F) Parameters are declared in functions header
- (T/F) A function's header and body must be defined within the same file
- (T/F) A function may return more than one value at a time
- Is the following syntax correct?
  - return;
- What's wrong, if any:

```
void fun1(int, float)
{
…
}
```

Exercises (cont.)

- If the function h is called as
  h((a,b,c));
  how many arguments are passed to h?

- Is following syntactically correct?
  return valu1,valu2;

↓Arguments and Parameters

A function is invoked with zero or more arguments.

```
print_stars ( value ); /* main ( ) invokes print_stars with one argument. */
while ( echo_line( ) != EOF ) {          /* with no argument */
```

Functions can be invoked in two different ways.

```
        print_stars ( value );          /* no return value, it is simply named */
```

and

```
        if(echo_line( ) ! = EOF)        /* has return value */
```

or

```
        I = echo_line( );               /* has return value, I is int */
```

- Matching Parameters with Arguments
  A function's parameters should match the function's arguments in number and data type, although arguments and parameters may have different names.

```
int echo_line ( void );
char grade ( int, int, float );
int rval, garbage, mid, final;
float weight;
char letter_grade;
...

 rval = echo_line ( );                                  /* right! */
 rval = echo_line ( garbage ); /* error, argument and parameter not match */
 letter_grade = grade ( mid, final, weight );     /* right */
 letter_grade = grade ( mid, weight, final );     /* error, data type not match */
 letter_grade = grade ( mid, final );  /* error, number not match  */
```

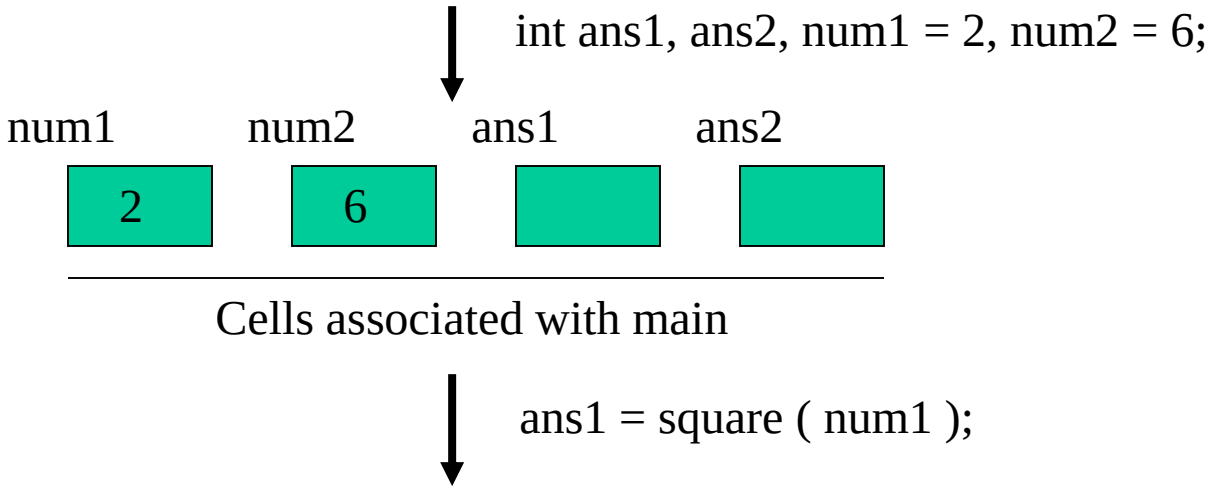
- Order of Evaluation of Arguments
  C guarantee that all the arguments passed to a function will be evaluated before control passes to the function. C does not guarantee the order of evaluation of the arguments.

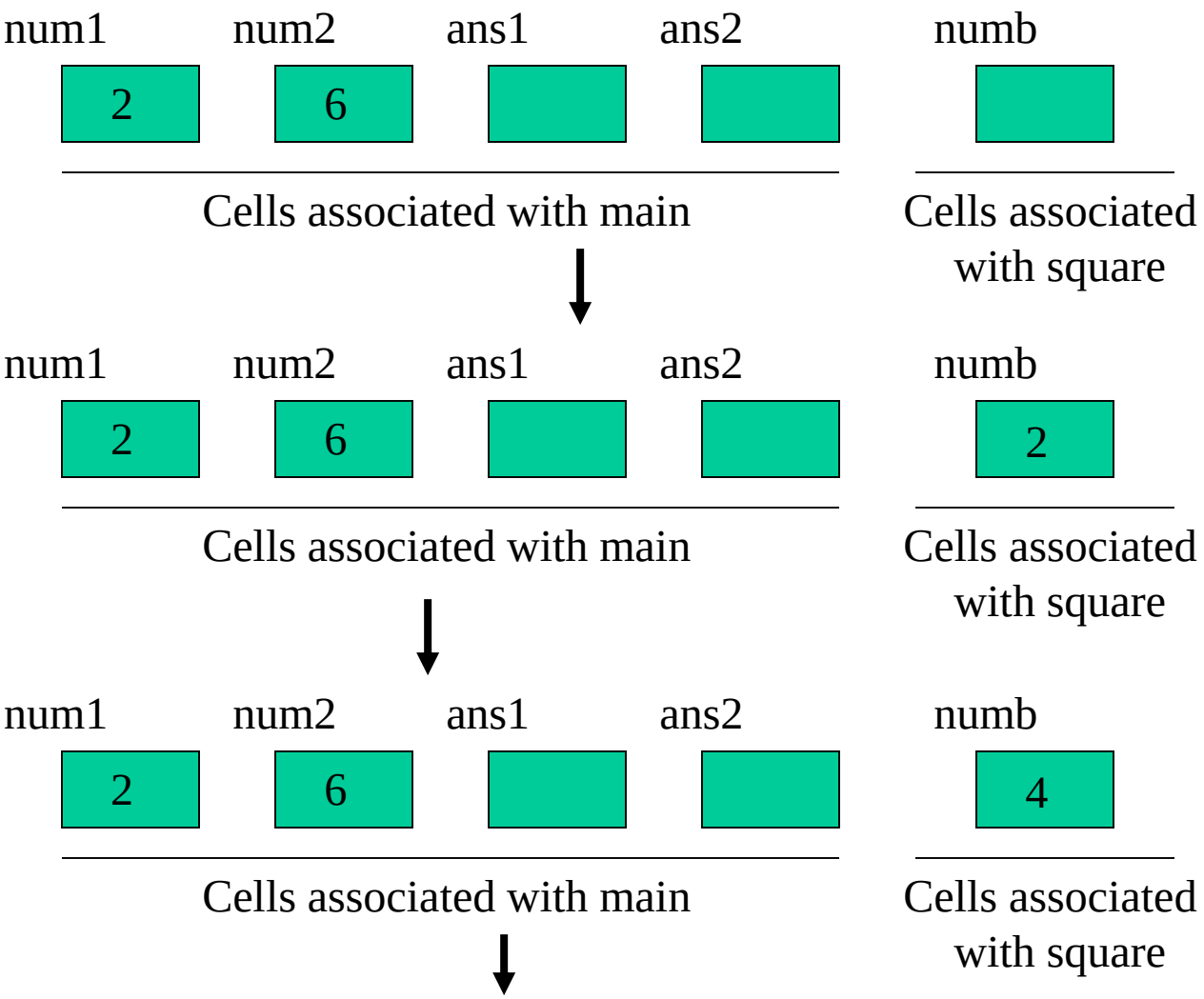
↓Call by Value

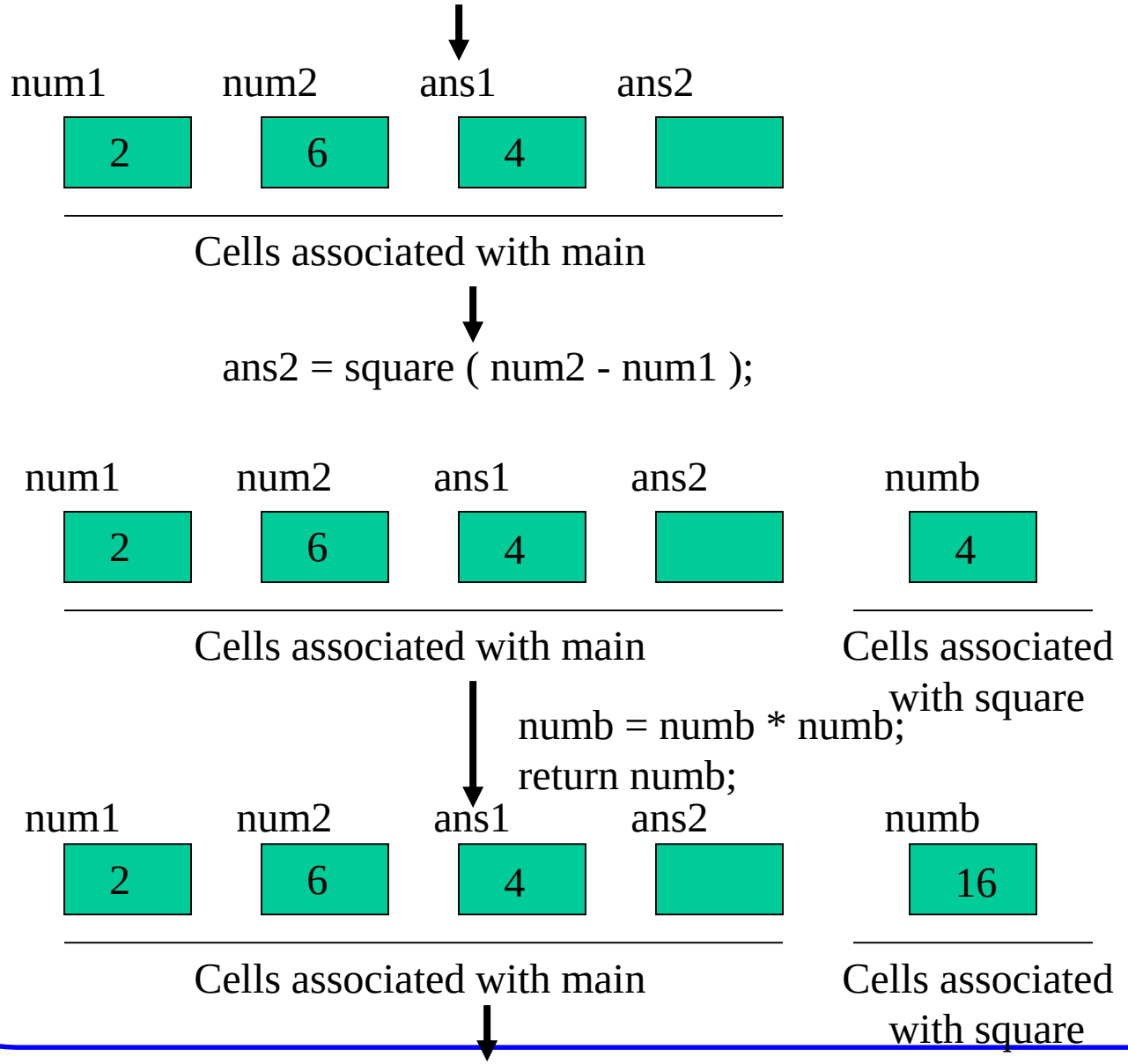
Call by value - the method of passing an argument to an invoked function by making a copy of the expression's value, storing it in a temporary cell, and making the corresponding parameter this cell's identifier.

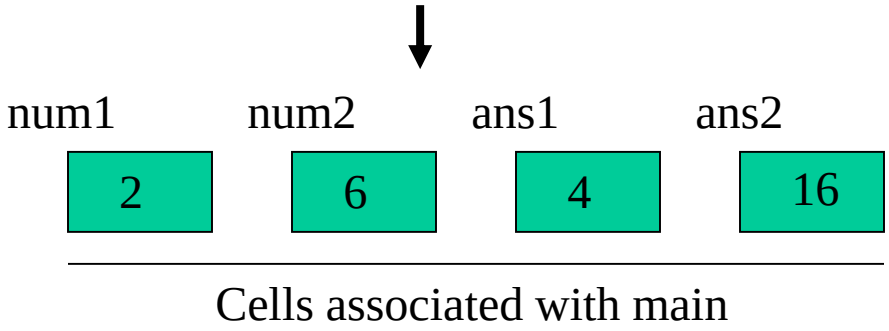
```
int square( int num ); /* declare the function square before main that invokes it */
main ( ) {
```

```
        int ans1, ans2, num1 = 2, num2 = 6;
        ans1 = square( num1);          /*invokes square and return 4, ans1 = 4 */
        ans2 = square(num2 - num1 ); /* invokes square and return 16, ans2 = 16*/
}
int square ( int numb ) {  /* The parameter numb becomes the identifier of the
                              special cell used to hold a copy of square's argument*/
        numb = numb * numb;
        return numb;
}
```




num1
num2
ans1
ans2
numb
2
6
Cells associated with main
Cells associated with square
num1
num2
ans1
ans2
numb
2
6
2
Cells associated with main
Cells associated with square
num1
num2
ans1
ans2
numb
2
6
4
Cells associated with main
Cells associated with square


num1
num2
ans1
ans2
2
6
4
Cells associated with main
ans2 = square (num2 - num1);
num1
num2
ans1
ans2
numb
2
6
4
4
Cells associated with main
Cells associated with square
numb = numb * numb;
return numb;
num1
num2
ans1
ans2
numb
2
6
4
16
Cells associated with main
Cells associated with square



Note: Call by value lets us pass arguments to a function with a guarantee that the original arguments will remain unchanged, no matter what the invoked function does to the copies.

↓Computing Resistance

♦ ***Problem***

Given single-character codes for the colored bands that mark a resistor, computer its resistance in ohms. The color codes are as follows:

Color	Code
Black	0
Brown	1

Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

If the integer codes of the bands are (in order) color1, color2, and color3, the resistance in ohms is

$$\mathbf{resistance = (10 * color1 + colo2) * 10^{color3}}$$

♦ ***Sample Input/Output***

The colored bands are coded as follows:

Color	Code
---	---

Black----------> B
Brown----------> N
Red----------> R
Orange----------> O
Yellow----------> Y
Green----------> G
Blue----------> E
Violet----------> V
Gray----------> A
White----------> W
Enter 3 codes: ERO
Resistance in Ohms: 62000.000000

- ***Solution***

  1, Code colors as single characters;
  2, From the single characters we can find the color code;
  3, Find the resistance.

- ***C Implementation***

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
void print_codes (void);  /* menu of codes */
double decode_char ( char code );
main ( ) {
        double resistance;
        double color1, color2, color3;
        char code1, code2, code3;
        print_codes ( );        /* Print codes and prompt for user input. */
        printf ( “Enter three codes: “);
        code1 = getchar ( );
        code2 = getchar ( ) ;
        code3 = getchar ( );
        /* Decode each character code */
        color1 = decode_char ( code1);
        color2 = decode_char ( code2 );
        color3 = decode_char ( code3 );
```

```
        if ( color1 == -999.0 || color2 == -999.0 || color3 == -999.0 )
                printf ( "\n\n\tBad code - cannot compute resistance\n");
        else {
                resistanc = ( 10.0 * color1 + color2 ) * pow ( 10.0, color3 );
                printf ( "\n\n\tResistance in ohms: \t%f\n", resistanc );
        }
        return EXIT_SUCCESS;
}
void print_codes ( void ) {
        printf ( " Color \t\t\tCode\n\t");
        printf ( "-----\t\t\t----\n\n");
        printf ( "\t\n" );
        printf ( "\tBlack-----------------> B\n" );
        printf ( "\t Brown-----------------> N\n" );
        printf ( "\t Red-------------------> R\n" );
        printf ( "\t Orange----------------> O\n" );
        printf ( "\t Yellow----------------> Y\n" );
        printf ( "\t Green-----------------> G\n" );
```

```
        printf ( “\t Blue------------------> E\n” );
        printf ( “\t Violet----------------> V\n” );
        printf ( “\t Gray------------------> A\n” );
        printf ( “\t White-----------------> W\n” );
}
double decode_char ( char ) {
        switch ( char ) {
                case ‘B’:
                        return 0.0;
                case ‘N’:
                        return 1.0;
                case ‘R’:
                        return 2.0;
                case ‘O’:
                        return 3.0;
                case ‘Y’:
                        return 4.0;
                case ‘G’:
                        return 5.0;
```

```
            case 'E':
                return 6.0;
            case 'V':
                return 7.0;
            case 'A':
                return 8.0;
            case 'W':
                return 9.0;
            default:
                return -999.0;    /* illegal code */
        }
    }
```

↓The Scope of Variables

Every variable has a **scope --** the region of the program in which it is **visible**.

- Variables Local to a Function
  The variables that we have been defining are **local** to the functions in which they reside.

```
void func1 (void ) {
      /* This I and j are local to func1. They cannot be seen outside this function */
      int I, j;
}
void func2 ( void ) {
      /* This I and j are local to func2. They cannot be seen outside this function */
      int I, j;
}
```

In above functions, we have four distinct variables.

Caution:

It is illegal to define two variables that have the same scope and the same name.