CONTROL STRUCTURES

simple if:

```
if (boolean exp) {
   statements // body
}
```

if-then-else:

```
if (boolean exp) {
   statements // true part
} else {
   statements // false part
}
```

multi-way if:

```
if (boolean exp) {
   statements // part 1
} else if (boolean exp) {
   statements // part 2
} else if (boolean exp) {
   statements // part 3
… // and so on
} else { // an else part is
            optional
   statements // else part
}
```

while loop (pre-test):

```
while (boolean exp) {
   statements // body
}
```

do-while loop (post-test):

```
do {
   statements // body
} while (boolean exp);
```

for loop:

```
for (exp1; exp2; exp3) {
   statements // body
}
```

is equivalent to:

```
exp1;
while (exp2) {
   same statements
   exp3;
}
```

break statement:

```
break;
```

– causes an exit from the enclosing loop.

continue statement:

```
continue;
```

– sends control back to the top of the enclosing loop.

CALL BY …

```
int sample (int a, int &b, int c[]);
```

"a" is call-by-value (just like a regular variable but given a value when the function is called).
"b" is call-by-reference. all operations on "b" actually operate on the variable supplied.
"c" is call-by-reference. all operations on "c" actually operate on the array supplied.

ARRAYS

```
// sample declaration
int a[4] = {a1, 2, 4, 12 };

// typical use
sum = 0;
for (i = 0; i < array_size; i++) {
   sum += a [i];
}

// typical function
void write_array(int a[], int n) {
   int i;
   for (i = 0; i < n; i++) {
      cout << a[i] << endl;
   }
}
```

BISECTION SEARCH

to find the correct x, first pick an x that is too low (low) and an x that is too high (high)

loop

pick mid = (low + high) / 2

if low and high are so close that mid must be within the desired tolerance of the correct x, mid is the answer

try out mid

if the result is exactly what we're looking for (or is close enough) mid is the answer.

```
if mid is too low
   low = mid
else
   high = mid
```

end loop

SIMULATION

`x = rand () /(RAND_MAX + 1.0);`
produces a random value between zero and 1 (0 <= value < 1)

MODEL PROGRAM

```
#include <iostream.h>
int add(int x, int y) {
    int result;
    result = x + y;
    return result;
}
int main() {
    int a, b, c;
    cout << "Enter two values: ";
    cin >> a >> b;
    c = add(a, b);
    cout << "The answer is " << c << endl;
    return 0;
}
```

EXPRESSIONS

<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to
!=	is not equal to
||	OR (either side is true)
&&	AND (both sides are true)
!	NOT (changes true to false, false to true)
%	modulus (gives remainder from division)

X++ means "use value of X, then increment X"
X-- means "use value of X, then decrement X"
++X means "increment X, then use new value"
--X means "decrement X, then use new value"

X += Y is equivalent to X = X + (Y)
(same idea for -=, *=. and /=)

STRINGS

string = a character array with '\0' at the end of the data.
typical loop:

```
for (i = 0; str[i] != '\0'; i++) {
}
```

STRUCTURES

```
struct Point {
    double lat, lng;
};  // *** note the semi-colon
Point point;          // declares variable
Point list[10];       // declares array
point.lat. = 45;      // access lat component
list[n].lng = 130;    // access lng component of
                      // array element n
```

INPUT (use iostream.h, fstream.h)

```
ifstream xin;       // declares an input stream object (for reading files)
xin.open(string);// attaches the input stream object to the file specified
xin >> resetiosflags(ios::ws);       // turns off  whitespace skipping when reading characters
xin >> setw(size_of_char_array) >> string;          // prevents overflow when reading strings
xin.fail()// returns true if the stream is in the failed state (something's gone wrong)
xin.eof()           // returns true if the program has tried to read past the end of the file
xin.clear()         // resets the failure flag
xin.ignore(count, ch);      // discards input characters until  "count" characters have been discarded
                            // or  character "ch" has been discarded (whichever comes first)
xin.get(array, size, ch);  // copies input characters into character array "array" until "size – 1"
                            // characters have been copied or character "ch" is seen.  in the second
                            // case character "ch" is not read.
xin.get();          // gets the next input character
xin.getline(array, size);  // copies the next line of input (everything up to the next '\n') into the
                            // character array supplied.  the '\n' which ends the input process is
                            // discarded.  at most "size – 1" characters will get read (if the line is
                            // too long to be read, the extra characters are just left in the pipeline)
```

OUTPUT (use iostream.h, fstream.h, iomanip.h)

```
ofstream xout;   // declares an output stream object (for reading files)
xout << setiosflags(ios::fixed|ios::showpoint);  // forces use of non-scientific notation and the display of zeroes
                                                 // to the right of the decimal point. this remains in effect until changed.
xout << setprecision(value);        // selects the number of digits to be displayed to the right of the decimal point
                                    // when outputting double values.  the choice remains in effect until changed.
xout << setw (value) << …           // indicates that the next value output is to occupy the specified number of
                                    // columns.  a one shot deal – affects only the next value output
xout << setfill(ch);          // selects the fill character to be used when padding output values to a specified width.
                              // the choice remains in effect until changed.
```

LIBRARY FUNCTIONS

```
double fabs(double x);   returns the absolute value of "x", for real numbers
int abs(int x);     returns the absolute value of "x", for integers
double log (double x)     natural log (log base e)
double log10(double x)  log base 10
double exp(double x)     returns "e" to power of  "x"
double sqrt(double x)     returns the square root of "x"
double pow (double x, double y);            returns  "x" to the power of "y"
double sin(double x);     returns the sine of "x" (note:  "x" is in radians)
double cos(double x);     returns the cosine of "x" (note:  "x" is in radians)
double asin(double x);   returns the inverse sin of "x" (in radians)
double acos(double x);  returns the inverse cosine of "x" (in radians)
double sinh(double x);   hyperbolic sin
double cosh(double x);  hyperbolic cosine
int strlen(const char str[]);          returns length of string (not counting the termination)
void strcpy(char tostr[], const char fromstr[]);       copies "fromstr" into "tostr"
int strcmp(const char str1[], const char str2[]);      compares "str1" to "str2"
        if str1 > str2, returns a value greater than zero
        if str1 == str2, returns zero
        if str1 < str2, returns a value less than zero
int stricmp(const char str1[], const char str2[]);     same as "strcmp" but ignores case
double atof(const char str[]);       converts a string to a floating point value
int atoi(const char str[]); converts a string to an integer value
int rand(void);    returns a "randomly" chosen value between 0 and RAND_MAX
bool isalnum(char ch);   returns true if the character is alphanumeric (letter or digit)
bool isdigit(char ch);       returns true if the character is a digit (one of '0' through '9')
bool isalpha(char ch);     returns true if the character is a letter
char toupper(char ch);   returns the uppercase equivalent of a character
char tolower(char ch);   returns the lowercase equivalent of a character
```