```c
#include <stdio.h>
#define DIVISOR 2.0

/* const float DIVISOR = 2.0 */
int sum, a, b;
float ave;
char name[15];

int main() {
   printf("What is your name? ");
   scanf("%s", name);

   printf("%s, give me two numbers: ", name);
   scanf("%d %d", &a, &b);
   ave = (a + b) / DIVISOR;

   printf("The average of %d and %d is %0.2f.\n", a, b, ave);

   return 0;
}
```

Compiling (using gcc)

- To generate an executable file
  - gcc sample.c -o sample
- To preprocess
  - gcc -E sample.c -o sample.i
- To generate an assembly language
  - gcc -S sample.c

INPUT, PROCESSING, OUTPUT statements

- A simple C program – see *p. 12* (modified - EAAp12.c)
- Remarks:
  - Use of symbolic constants (#define vs. const declaration)
  - Coding style:
    - Use of horizontal/vertical space to reflect block-structure
    - Indention & bracing style (1TBS, Allman)
    - Documentation
- Use of prompt messages
- scanf, assignment, printf statements

Non-executable statements

Pre-processor commands *(p. 8)*

- **#include < ... >**

  e.g. #include <stdio.h>

- **#define ...** *(p. 15)*

  e.g. #define DIVISOR 2.0

Declaration statements

- variable declaration *(p. 5)*

  <data type> variable(s);

- constant declaration

  const <data type> variable = <constant value>;

Input statement (overview)

scanf(*format string***, *argument list*);**

Format string – contains format codes:

%c – char type
%d, %i – int type
%f – floating point number
%s – character string

Argument list

- list of variables separated with comma,
- variable is preceded by & (except those of string type)

Matching – BOTH format codes and argument list should match consistently with each other

Header file - stdio.h

Output statement (overview)

printf(*format string*, *argument list*);

Format string – contains other characters + format codes:
Argument list
- list of variables &/or expressions separated with comma
- OPTIONAL component

Matching – BOTH format codes and argument list should match consistently with each other

Header file - stdio.h

Processing statement (overview)

identifier = *expression*;

Assignment, NOT equality
Type matching
- implicit conversion, type casts

Mathematical expression
- integer division

Increment/decrement operators
Shortcut operators
Rule of precedence *(p. 24)*
- overriding through the use of parenthesis

True or False boolean values as Integers
Nested statements

Increment/Decrement Operators

Consider:

```
count = count + 1;
```

Alternative syntax (using the Increment operator):

```
count++;
```

or

```
++count;
```

NOTE: if embedded within another expression --
e.g. Assigment statement, relational expression, etc
--placement of ++ matters.

Increment Operator

Compare: (assume X is initially 7)

Y = X++; vs. Y = ++X;

Analysis: (after executing):

X = ?, Y = ? vs. Y = ?, X = ?

Notice:

A nested statement: 2 operations in 1.

The longer way:

Y = X; vs. ++X;

X++; Y = X;

Decrement Operator

Consider:

count = count – 1;

Alternative syntax:

count--; or --count;

Analyze:

Y = X--; vs. Y = --X;

Shortcut Operators

Consider:

<var> = <var> <OPERATOR> <constant>;

Alternative:

<var> <OPERATOR>= <constant>;

Example:

n = n * 2; or n *= 2;

Other operators:

/=, -= , +=

Nested statements:

- Increment/decrement within an assignment statement
- Assignment statement within another assignment statement
  E.g. a = b = c = d = e;
- Increment/decrement within a relational expression...
- Increment/Decrement within printf..
- Assignment within a relatonal expression...
- etc..

Rule of precedence

- +, -, *, /, %
  - Highest: *, /, %
  - Lowest: +, -
- To override this rule, or clarify your intent, use parenthesis operators.
- Compare: code optimization & code readability
- Detailed discussion, p. 24

Another Example

Swapping Without using a Temporary Variable *(p. 144)*

Swap values of a & b, w/o a temporary variable:

```
a = a + b;
b = a - b;
a = a - b;
```

Let sum = a + b

b' = sum – b
= (a + b) – b
= a

a' = sum – b'
= (a + b) - a
= b

Program Control Flow

3 Control Flows

Sequential – (default) statements are executed in the order that they are written

Conditional – a statement(or group of statements) may or may not be executed at all

Iterative – a statement(or group of statements) is executed repeatedly

Conditional Statements

Boolean condition is at the heart of every conditional/iterative statement

Condition is either true or false

- true (numerical: non-zero)
- false (numerical: zero)

Simple to complex expressions

- use of logical & relational operators
- rule of precedence applies
- TIP: Parenthesize subexpressions to clarify intent

Code block

- sequence of statements grouped together by curly braces
- treated as a unit

Conditional Statements

Two-way selection

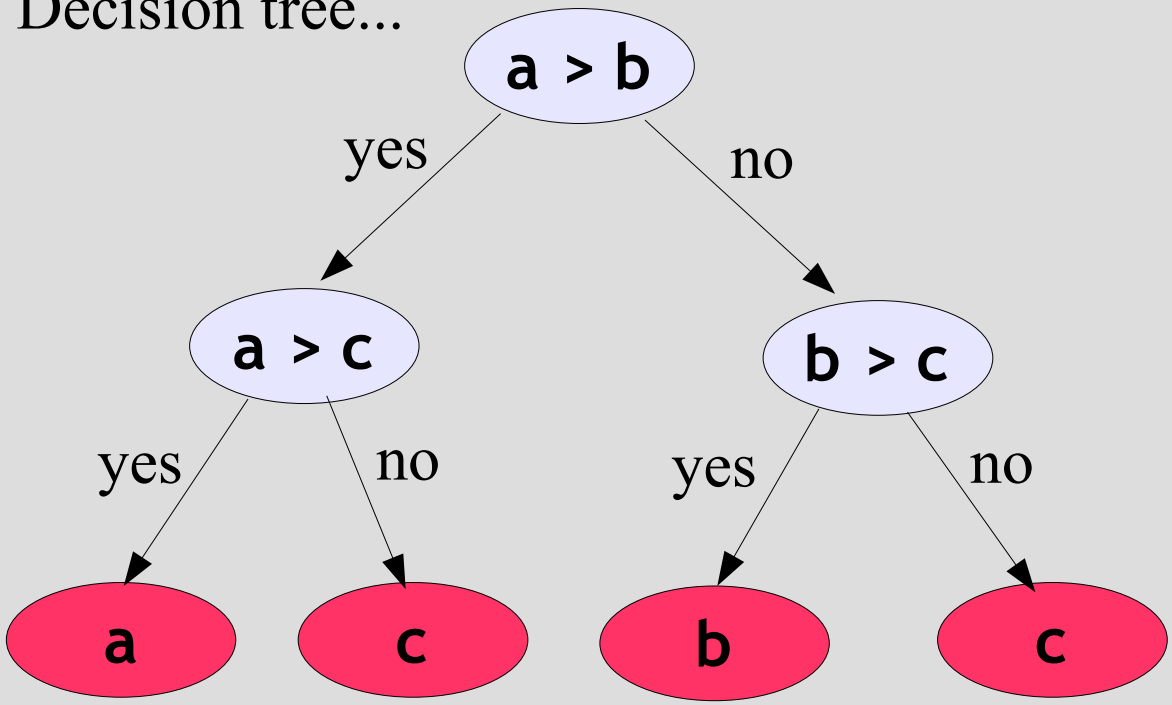
```
if condition
    statement1;
else
    statement2;
```

- nature of the condition
- else part is optional; if-else pairing
- statement could be a code block *(def'n)*
- nested statement, *ladderized* if-else
- ternary expression

Conditional Statements - sample program

Finding the maximum of three numbers *(p. 146 – sol'n modified)*

Decision tree...




```
if (a > b) {
   if (a > c)
      max = a;
   else
      max = c;
} else {
   if (b > c)
      max = b;
   else
      max = c;
}
```

Two-way selection...

CODE OPTIMIZATION

Ternary operator (?) - conditional expression, p. 32

```
if expr1
   expr2;
else
   expr3;
```

```
expr1 ? expr2 : expr3;
```

```
if (x > y)
   z = x;
else
   z = y;
```

```
z = (x > y) ? x : y ;
```

Conditional Statements

Multi-way selection

```
switch expression {
  case const1 : statement1;
                break;
  case const2 : statement2;
                break;
  ...
  case constn : statement3;
                              break;
  default : statementn+1;
}
```

- 1 out of n+1 statements
- OPTIONAL default stmt
- nature of comparison
- use of break,
- nature of the statements
- relates to *ladderized* if-else statement

Multi-way selection...

SAMPLE PROGRAM #1

Ladderized if-else VS switch statement

```
if (num == 1)
   printf("one\n");
else if (num == 2)
   printf("two\n");
else if (num == 3)
   printf("three\n");
else if (num == 4)
   printf("four\n");
else if (num == 5)
   printf("five");
else
   printf("%d", num);
```

```
switch (num) {
   case 1 : printf("one\n");
            break;
   case 2 : printf("two\n");
            break;
   case 3 : printf("three\n");
            break;
   case 4 : printf("four\n");
            break;
   case 5 : printf("five\n");
            break;
   default: printf("%d", num);
}
```

Multi-way selection...

SAMPLE PROGRAM #2

```
if ((num == 1) || (num == 3) || (num == 5))
   printf("odd\n");
else if ((num == 2) || (num == 4))
   printf("even\n");
else
   printf("%d", num);
```

```
switch (num) {
   case 1 :
   case 3 :
   case 5 : printf("odd\n");
            break;
   case 2 :
   case 4 : printf("even\n");
            break;
   default : printf("%d", num);
}
```

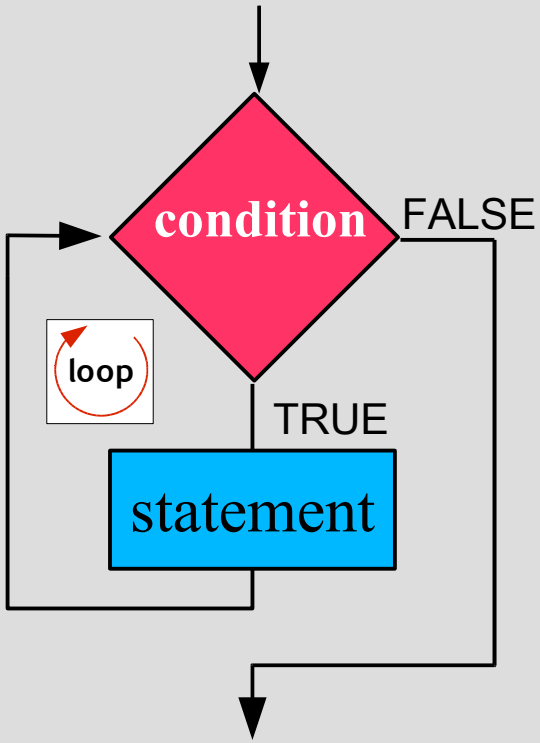
Multi-way selection...

MORE SAMPLE PROGRAMS

Pages 33-40

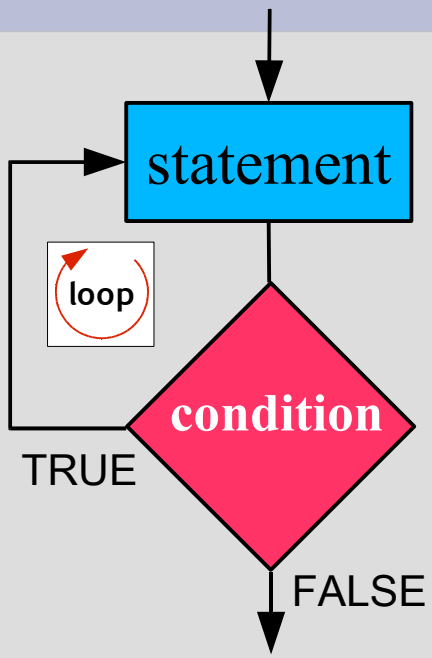
Iterative Statements

Test-before



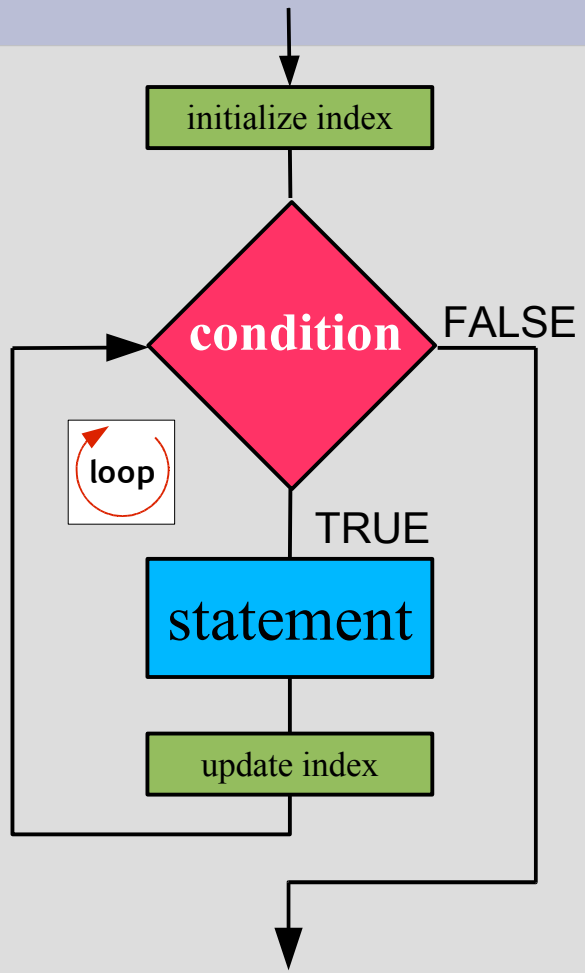

```
while condition
  statement;
```

Test-after




```
do
  statement;
while condition;
```

Indexed




```
for (initialization; condition; update)
  statement;
```

```c
i = 10;
while (i >= 2) {
  printf("%d\n", i);
  i -= 2;
}
```

```c
i = 10;
do  {
  printf("%d\n", i);
  i -= 2;
} while (i >= 2);
```

```c
for (i = 10; i >= 2; i-=2) {
  printf("%d\n", i);
}
```

```c
i = 10;
while (i > 0) {
   printf("%d\n", i);
   i -= 2;
}
```

```c
i = 12;
do  {
   i -= 2;
   printf("%d\n", i);
} while (i > 2);
```

```c
for (i=10; i > 1; i-=2)
   printf("%d\n", i);
```

Variations in Loops

```
for (i=0, j=1; i<3; i++, j+=2)
   printf("%d\n", i*j);

for (i=0; i != 7;)
   scanf("%d", &i);

for (;;) {
   scanf("%d", &x);
   if (x == 7)
     break;
}
```

Variations in Loops

```
i = 1;
        while (++i <= 4)
         printf("%d", i);

                -----
i = 1;
        while (i++ <= 4)
         printf("%d", i);
```

Variations in Loops

```
while (1) {
        scanf("%d", &x);
        if (x==7)
            break;
}
```

Use **continue** to ignore succeeding statements in the loop body.

```
i = 0;
while (i < 5) {
   i++;

   if (i > 2)
      break;

   k = 0;
   do {
      k++;
      if (k > 3)
         continue;

      printf("%d\n", i * k);
   } while (k < 10);         }
```