W2

CHAPTER 3 READ

Problem Solving

→ **Short-hand:** `val = val ± term;`
≙ `val ±= term;`

`val = val * 2;`
≙ `val *= 2;`

incr/decr (by ± 1): `x = x + 1;`
≙ `x += 1;`
≙ `x++;`

→ **LOOPS:** "Execute set of statements multiple times."

```
for (i = 0; i < N; i++)
{ "DO THIS"   /* DONE N TIMES */
}
```

```
/* ADD 10 INTEGERS */
for (i=0; i < 10; i++)
{
  printf ("Integer to be added to sum: ");
  value = GetInteger();
  sum += value;
}
```

IMPORTANT: INITIALIZE:

`sum = 0;`

W2

ITERATION

⇒ USE FOR OR WHILE LOOP — BETTERLOOP??

```
for (i=0; ... )
     ↑ INDEX VARIABLE        /* COUNTING LOOP */
```

POSSIBILITIES

```
for (i=0; i<N; i++)
or: for (i=1; i<=N; i++)
or: for (i=1; i<(N+1); i++)
```

DECLARATION & INITIALIZATION

```
main ()
{ int i, val, sum;          /* DECLAR. */

  sum = 0;                  /* INIT. */
  for (i=0; i<10; i++)
  { printf (" ? ");
    val = GetInteger ();
    sum += val;
  }
  printf ("The sum is %d.. \n", sum);
}
```

W2

→ **WHILE LOOP:** "Execute set of statements until a condition is violated."

EX: /* READ INTEGERS AND ADD TIL '0' IS INPUT */

```
main()
{
   int     val, sum;        /* DECLAR. */

   sum = 0;                 /* INIT. */

   while (TRUE)             BOOLEAN CONSTANT
   {
      printf ("Input integer: ");
      val = GetIntegr();
      if (val == 0) break;        EXIT LOOP
                                  /* '==' means EQUAL */
      sum += val;
   }
...
}
```

FIG. 3-3

"RELATIONAL OPERATORS"

==	EQUAL
!=	NOT EQUAL
>	GREATER THAN
<	LESS THAN
>=	GREATER THAN OR EQUAL
<=	LESS THAN OR EQUAL

W2

"Conditional Execution":

Execute set of statements (once) if ("a condition) is met."

```
if ('condition')
{ "DO THIS"
}
```

EX:

```
/* DEDUCT $10.00 FOR BOUNCING CHECK */

main ()
{ double entry, bal ;                    /* DECL. */
  printf ("Initial bal?\n" );
  bal = GetReal() ;                      /* INIT. */
  while (TRUE)
  {printf ("Enter check (-) or deposit amount: " );
   entry = GetReal();
   if (entry == 0) break ;               /* TERMINATION */
                                         /* OF LOOP */
   bal += entry ;
   if ( bal < 0.0 && entry < 0.0)
                                         /* NOT CORRECT: */
                                         /* if (bal < 0.0) */
   { printf ("Check bounced - $10.00 deducted.\n");
     bal -= 10.0;
   }
  } /* end while */

} /* end main */
```

FORMATTING

... double balance;

printf ("Balance is %.2f. \n", balance);

"Print as a floating-point number using 2 digits after decimal point."

- **GENERAL:**

printf ("control string", expr1, ..., exprN);

set of CHARACTERS, STRINGS & PLACEHOLDERS (STARTING WITH '%')
→ no. of placeholders = no. of expr. !

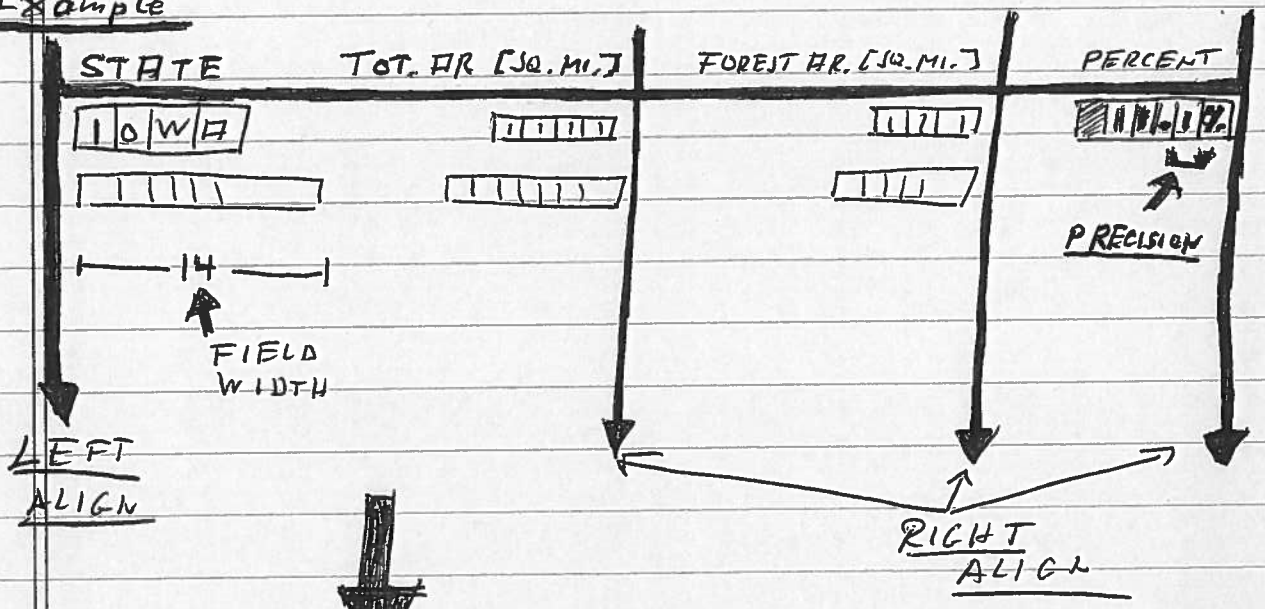
- **TYPES**

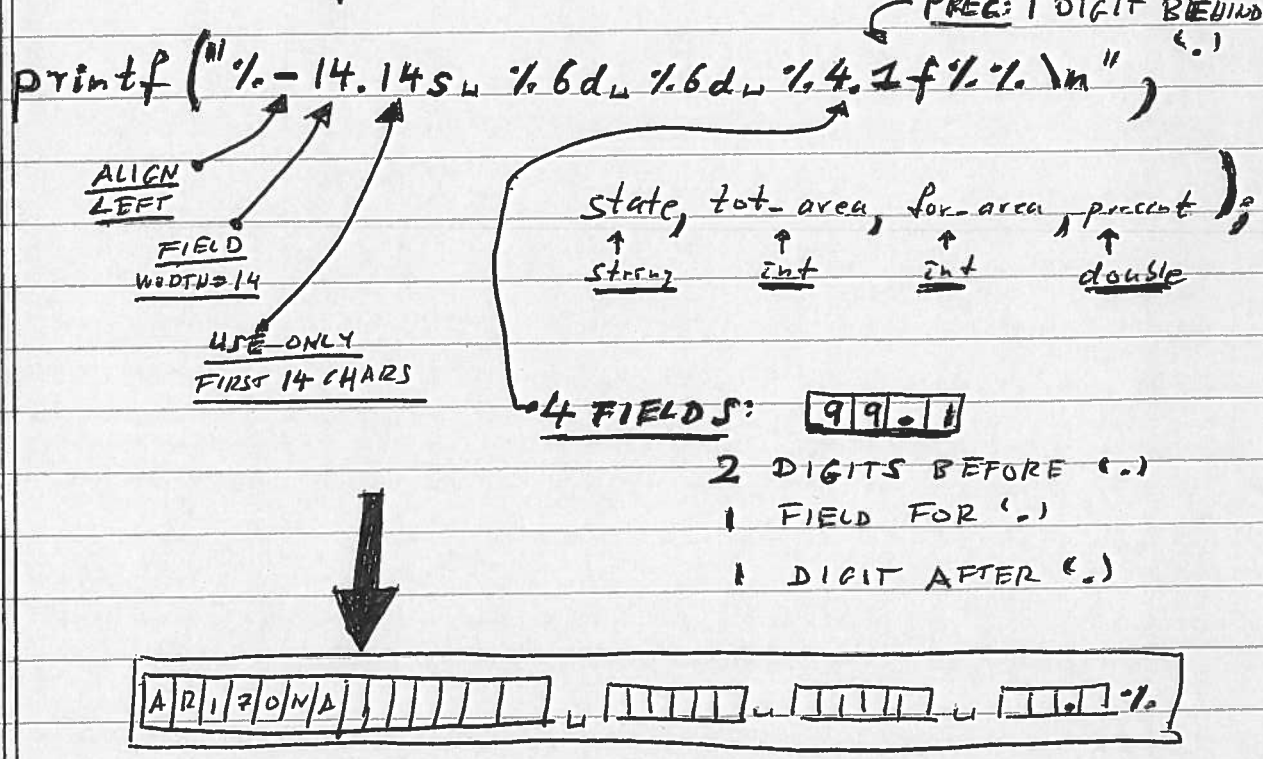
%d	dec. integer	-1000, ..., MAX_INT
%f	floating-point format	1.89, ..., 999.44
%e	scientific format	1.111e-14
%E	scientific format	1.111E-14
		(= 1.111 * 10^{-14})
%g	general format (will %f or %e format)	
%s	string "	
%c	character "	
%%	character '%' itself	

SPACING/ALIGNMENT/PRECISION

for printf

Example





GENERAL INSTRUCTIONS FOR PROGRAMMING ASSIGNMENTS

(1) Your program must solve the problem EXACTLY as stated. Do not solve an alternate problem or change any of the requirements. If you have any questions or the problem is not clear, see a teaching assistant for clarification.

(2) Programs must display good STYLE. Use blank lines to separate blocks of related statements. Indent to enhance readability. Use at least four (4) spaces for each indentation; fewer spaces do not aid readability.

(3) Use in-line COMMENTS to explain the usage of declared variables and the various steps of your algorithm.

(4) Identifiers must be meaningful NAMES. For example, if an input value is a cost, name the variable "cost", not "x". With the possible exception of 0, 1, -1 and numbers in format strings, programs must use constant macros for numbers (with upper-case identifiers such as TAX RATE) rather than constant values, e.g., 0.07.

(5) Prior to reading INPUT VALUES from the keyboard, a brief explanation should be written that explains what should be typed. OUTPUT VALUES should also be accompanied by a brief explanation. Writing out input values (echoing them) assists in checking that they were stored correctly.

(6) Assure that your program works for ALL POSSIBLE input values. While extensive testing cannot prove the absence of bugs, you should strive to show the correctness of your program. Programs should also be robust, anticipating user errors.

(7) The PROGRAM HEADER must contain your name, ECS 30, your section, the instructor's name, the programming assignment number, the completion date, and a brief description of the program.

(8) Each function must be preceded by a FUNCTION HEADER that briefly describes the function and its preconditions. Postconditions may also be helpful.