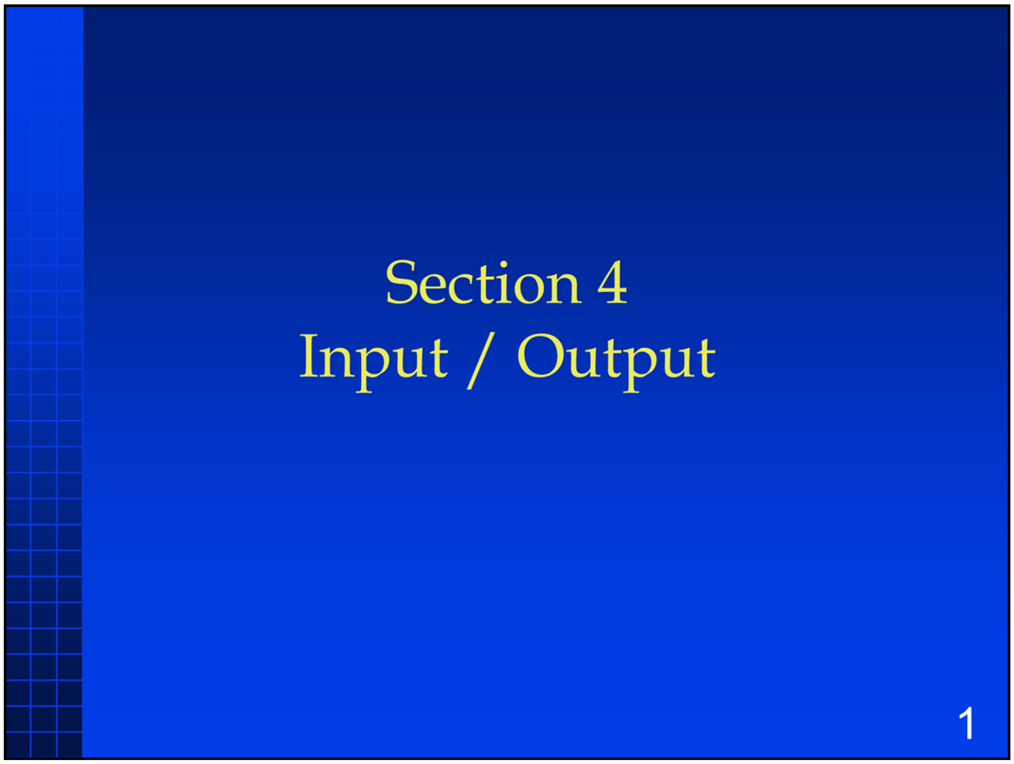
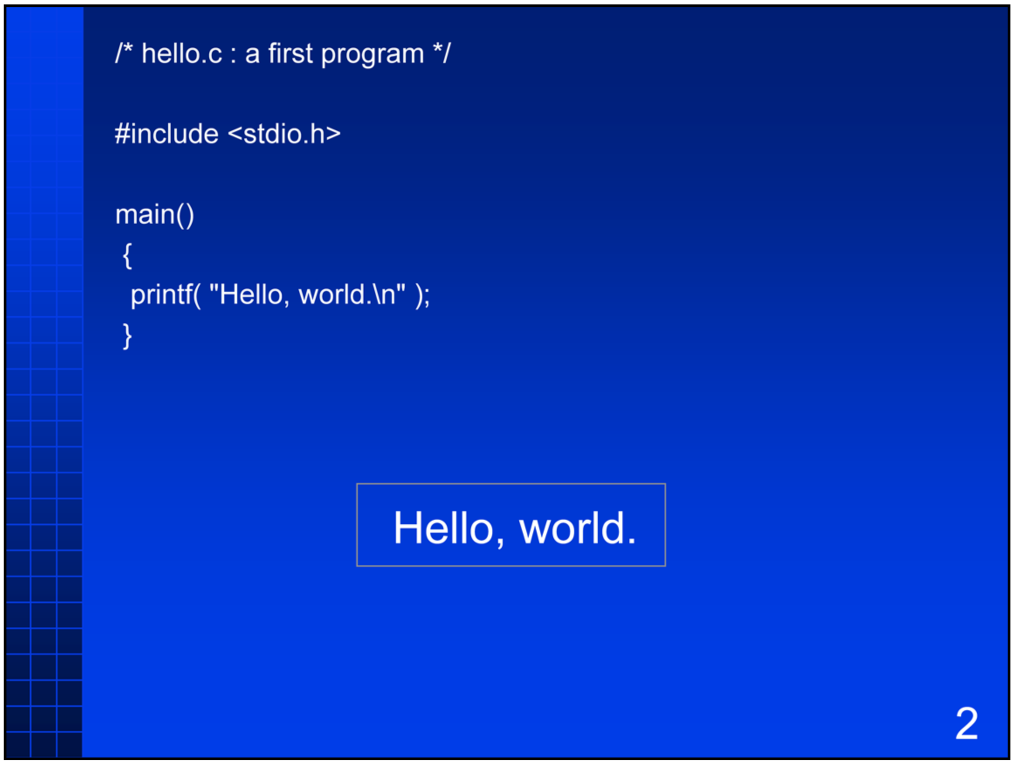


Section 4
Input / Output

More about printf and scanf;

Quick look at textfile I/O (reading and writing files that contain text, suitable for use with a text editor like notepad)



```
/* hello.c : a first program */

#include <stdio.h>

main()
 {
  printf( "Hello, world.\n" );
 }
```

```
Hello, world.
```

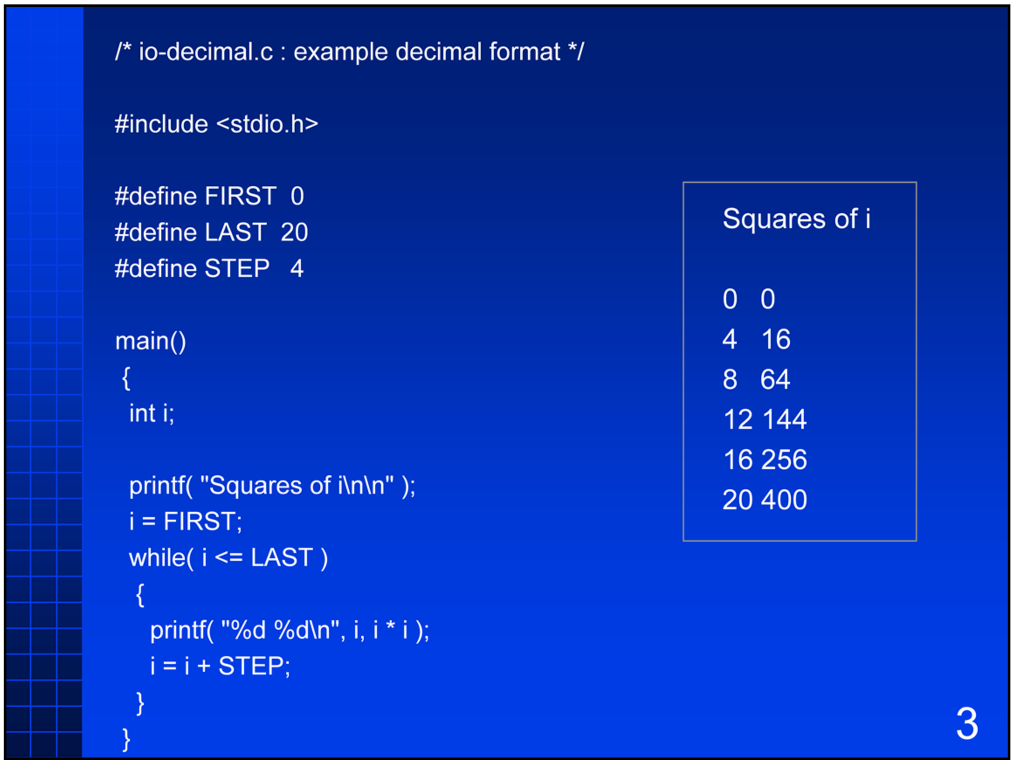
The classic c program, prints/displays string somewhere.

Where? C adopts from UNIX and elsewhere notion of standard files: in C’s case there are three:
stdout, where output goes (eg printf),
stdin, where input comes from (eg scanf),
stdterr, where error-messages go

These entities are defined in <stdio.h> and automatically initialized.

Meaning and behaviour is spstem-dependent. On line-mode systems like old-style UNIX & DOS, stdout = screen, stdin = keyboard, stderr = screen.

In current windowing systems, unclear clear. Concept not really supported in these environments. Lab software, watcom c, creates a window that behaves like line-mode screen.



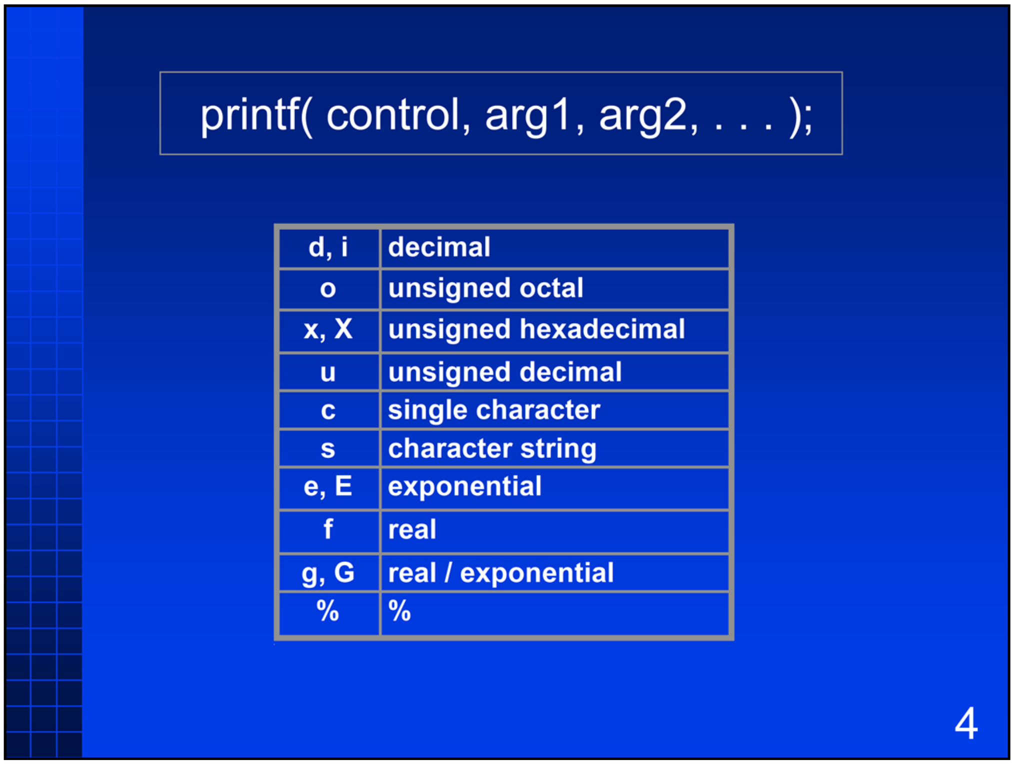
```
/* io-decimal.c : example decimal format */

#include <stdio.h>

#define FIRST  0
#define LAST  20
#define STEP   4

main()
 {
  int i;

  printf( "Squares of i\n\n" );
  i = FIRST;
  while( i <= LAST )
   {
     printf( "%d %d\n", i, i * i );
     i = i + STEP;
   }
 }
```

```
Squares of i

0   0
4   16
8   64
12 144
16 256
20 400
```

Review:

printf, control string, formatting directives, values contained in subsequent parameters, programmer’s responsibility to get these correct.

using %d for integers here. note left aligned



printf(control, arg1, arg2, . . .);

d, i	decimal
o	unsigned octal
x, X	unsigned hexadecimal
u	unsigned decimal
c	single character
s	character string
e, E	exponential
f	real
g, G	real / exponential
%	%

there are lots of directives, this is most (missing %p for pointer, %n for target output length (written to parameter))

d,i: decimal integers [i for compatibility with scanf]

o: converts to octal

x,X: converts to hex, case controls output case

.

.

.

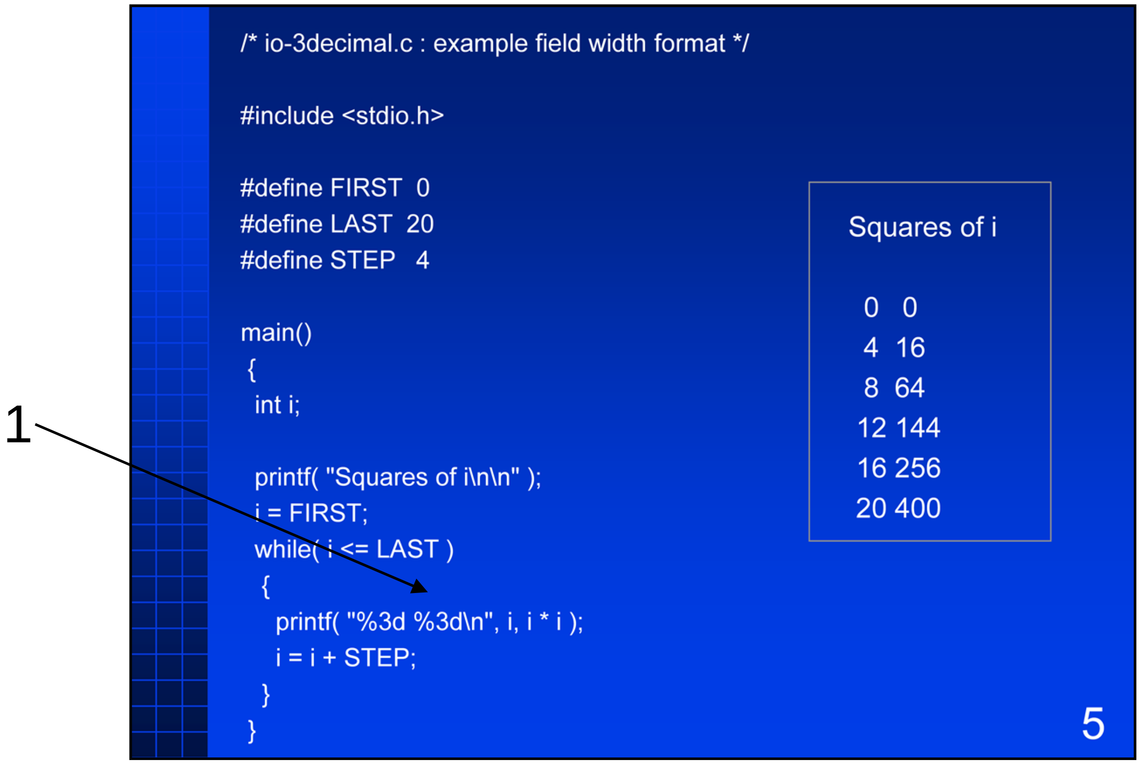
e,E: exponential (scientific: d.ddd eii), case controls output case

f: real (fixed-point)

g,G: decides for itself between e and f

%: prints a %

Again: control string defines types of parameters and how to process, therefore what to pass; user’s responsibility

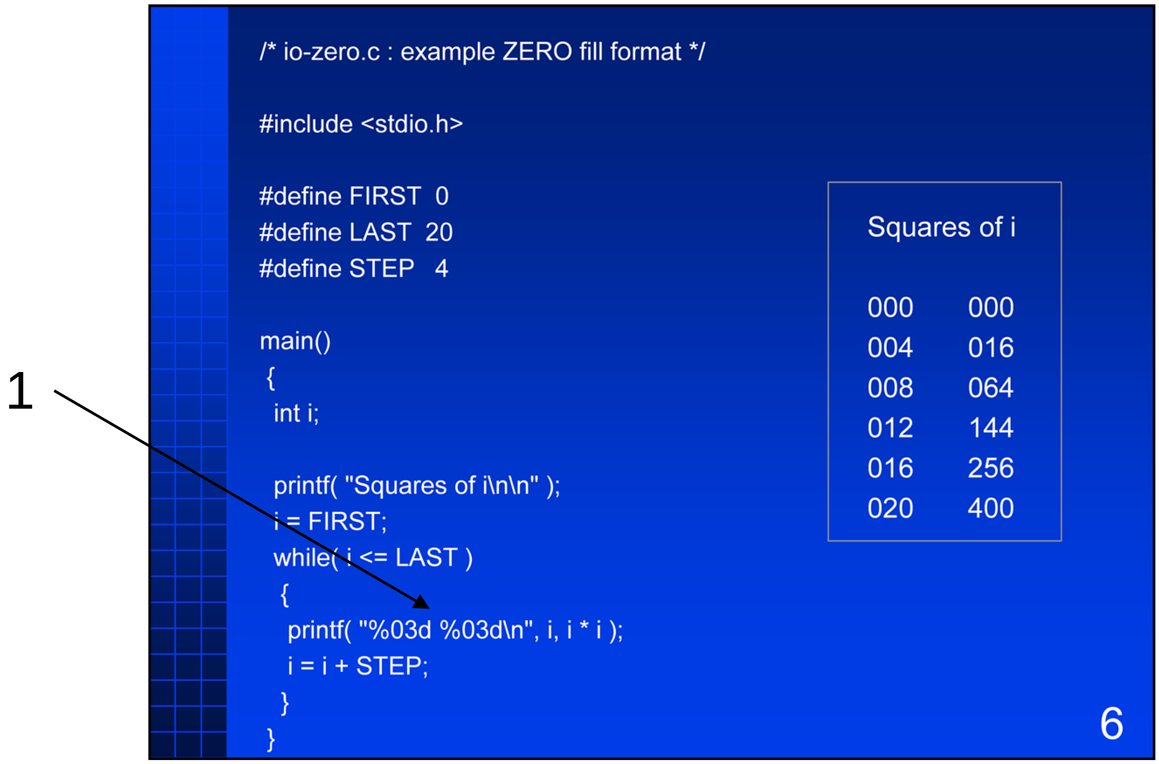


```
/* io-3decimal.c : example field width format */

#include <stdio.h>

#define FIRST  0
#define LAST  20
#define STEP   4

main()
 {
  int i;

  printf( "Squares of i\n\n" );
  i = FIRST;
  while( i <= LAST )
   {
     printf( "%3d %3d\n", i, i * i );
     i = i + STEP;
   }
 }
```

```
Squares of i

  0   0
  4  16
  8  64
 12 144
 16 256
 20 400
```

1) minimum field width, will be enlarged if necessary. note right alignment of output

- minus sign inserted if necessary

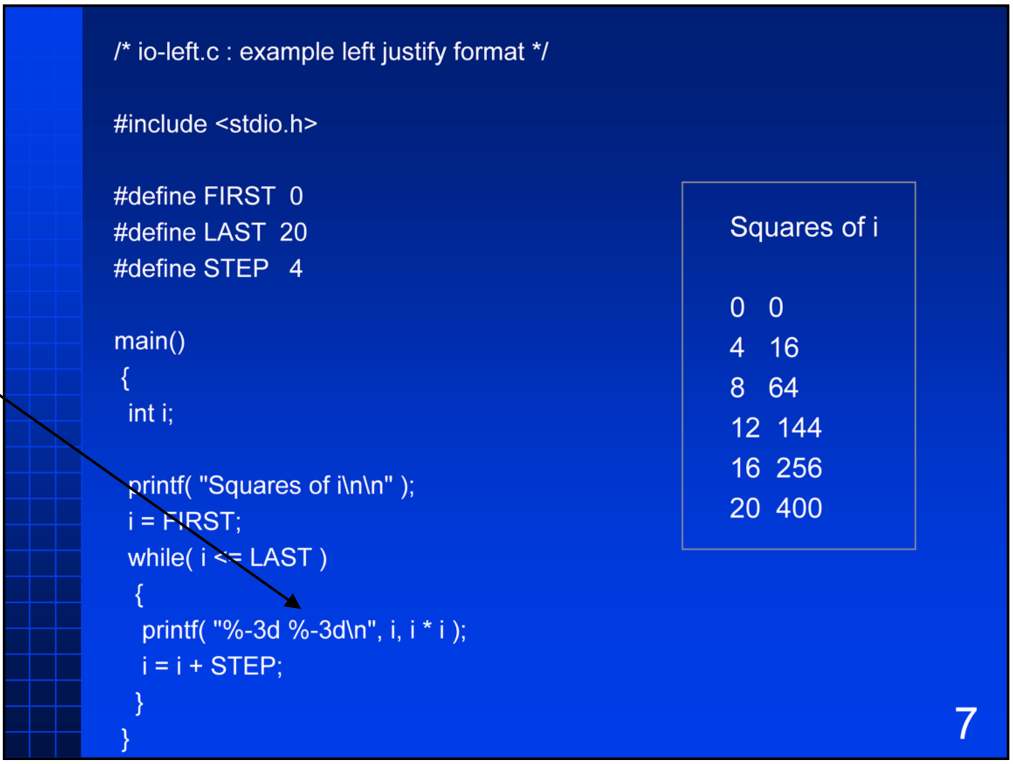


```
/* io-zero.c : example ZERO fill format */

#include <stdio.h>

#define FIRST  0
#define LAST  20
#define STEP   4

main()
 {
  int i;

  printf( "Squares of i\n\n" );
  i = FIRST;
  while( i <= LAST )
   {
    printf( "%03d %03d\n", i, i * i );
    i = i + STEP;
   }
 }
```

```
Squares of i

000     000
004     016
008     064
012     144
016     256
020     400
```

1) same, leading zeroes

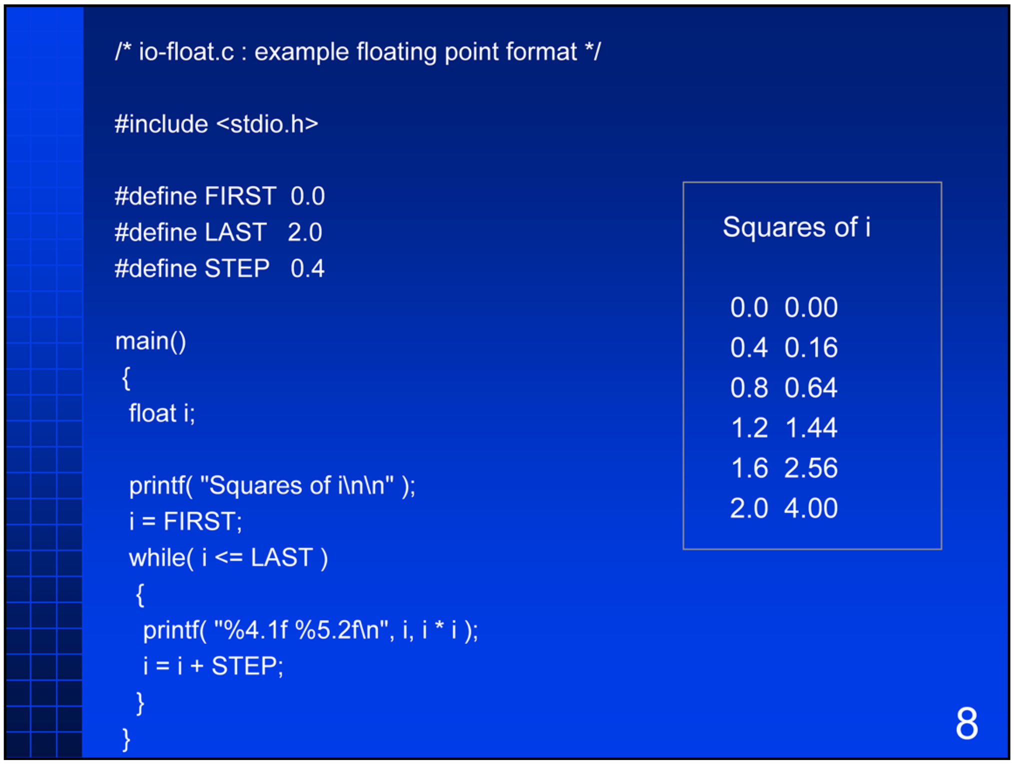
1



```
/* io-left.c : example left justify format */

#include <stdio.h>

#define FIRST  0
#define LAST  20
#define STEP   4

main()
 {
  int i;

  printf( "Squares of i\n\n" );
  i = FIRST;
  while( i <= LAST )
   {
    printf( "%-3d %-3d\n", i, i * i );
    i = i + STEP;
   }
 }
```

```
Squares of i

0   0
4   16
8   64
12  144
16  256
20  400
```

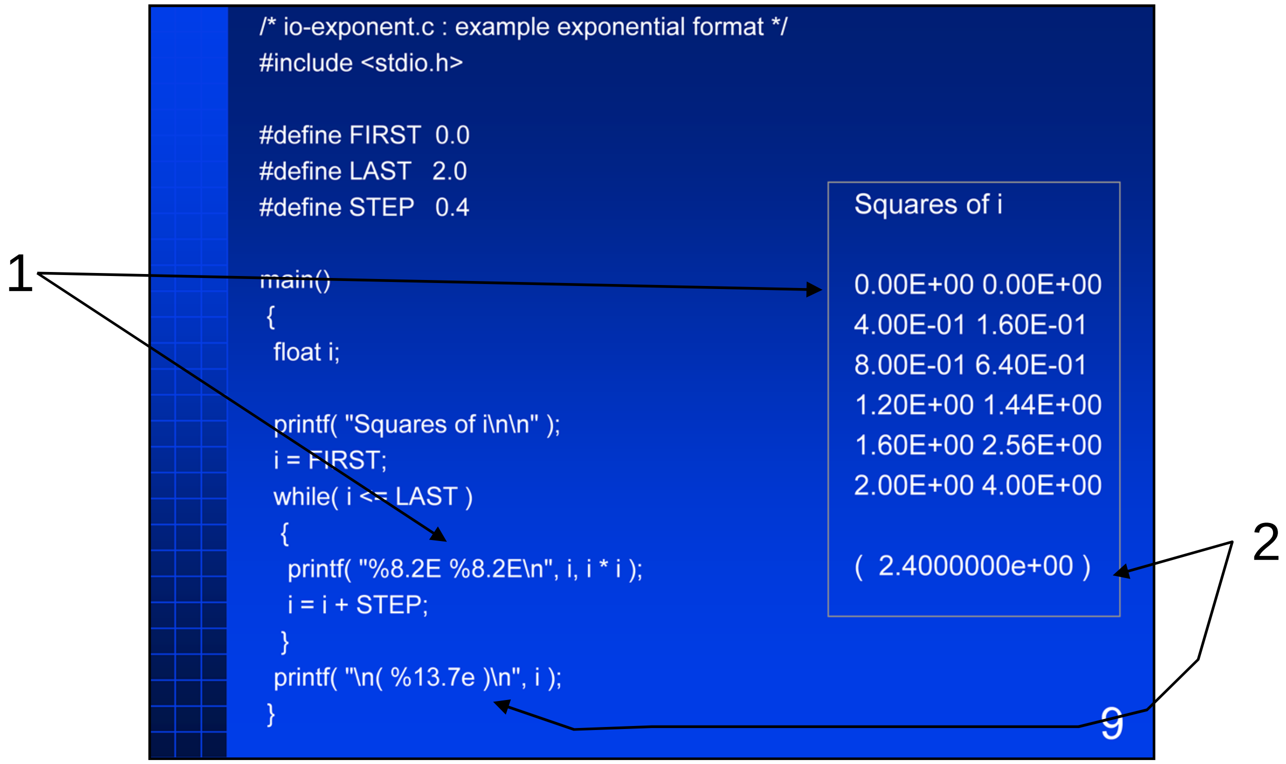
1) negative field-width: get rid of zeroes, left-align in fixed-width columns



```
/* io-float.c : example floating point format */

#include <stdio.h>

#define FIRST  0.0
#define LAST   2.0
#define STEP   0.4

main()
 {
  float i;

  printf( "Squares of i\n\n" );
  i = FIRST;
  while( i <= LAST )
   {
    printf( "%4.1f %5.2f\n", i, i * i );
    i = i + STEP;
   }
 }
```

```
Squares of i

 0.0  0.00
 0.4  0.16
 0.8  0.64
 1.2  1.44
 1.6  2.56
 2.0  4.00
```

floating-point example, nothing new here

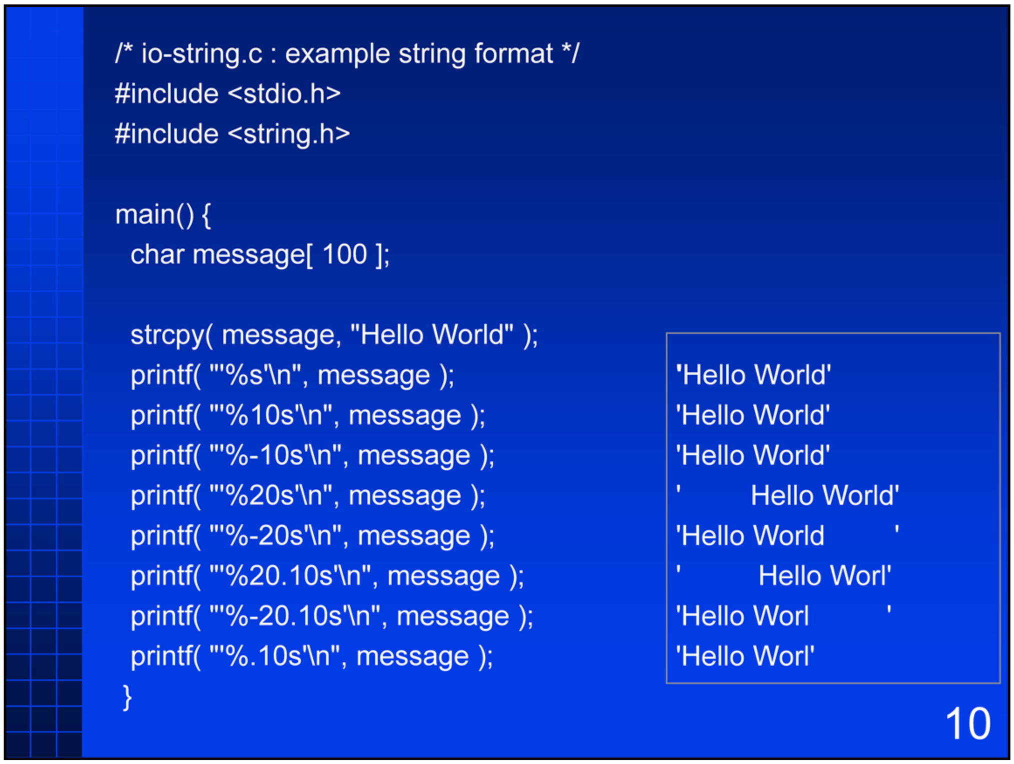


```
/* io-exponent.c : example exponential format */
#include <stdio.h>

#define FIRST  0.0
#define LAST   2.0
#define STEP   0.4

main()
 {
  float i;

  printf( "Squares of i\n\n" );
  i = FIRST;
  while( i <= LAST )
   {
    printf( "%8.2E %8.2E\n", i, i * i );
    i = i + STEP;
   }
  printf( "\n( %13.7e )\n", i );
 }
```

```
Squares of i

0.00E+00 0.00E+00
4.00E-01 1.60E-01
8.00E-01 6.40E-01
1.20E+00 1.44E+00
1.60E+00 2.56E+00
2.00E+00 4.00E+00

(  2.4000000e+00 )
```

1) exponential/scientic format: mantissa and exponent; precise format is system-dependent and usually coordinated with system datatypes.

2) upper and lower case “e’



```
/* io-string.c : example string format */
#include <stdio.h>
#include <string.h>

main() {
  char message[ 100 ];

  strcpy( message, "Hello World" );
  printf( "'%s'\n", message );
  printf( "'%10s'\n", message );
  printf( "'%-10s'\n", message );
  printf( "'%20s'\n", message );
  printf( "'%-20s'\n", message );
  printf( "'%20.10s'\n", message );
  printf( "'%-20.10s'\n", message );
  printf( "'%.10s'\n", message );
 }
```

```
'Hello World'
'Hello World'
'Hello World'
'         Hello World'
'Hello World         '
'          Hello Worl'
'Hello Worl          '
'Hello Worl'
```

string has 11 visible characters, not use of singles to show fields

1) print whole string

2) field length is a minimum, so no-op in this case

3) left-aligned, still a no-op

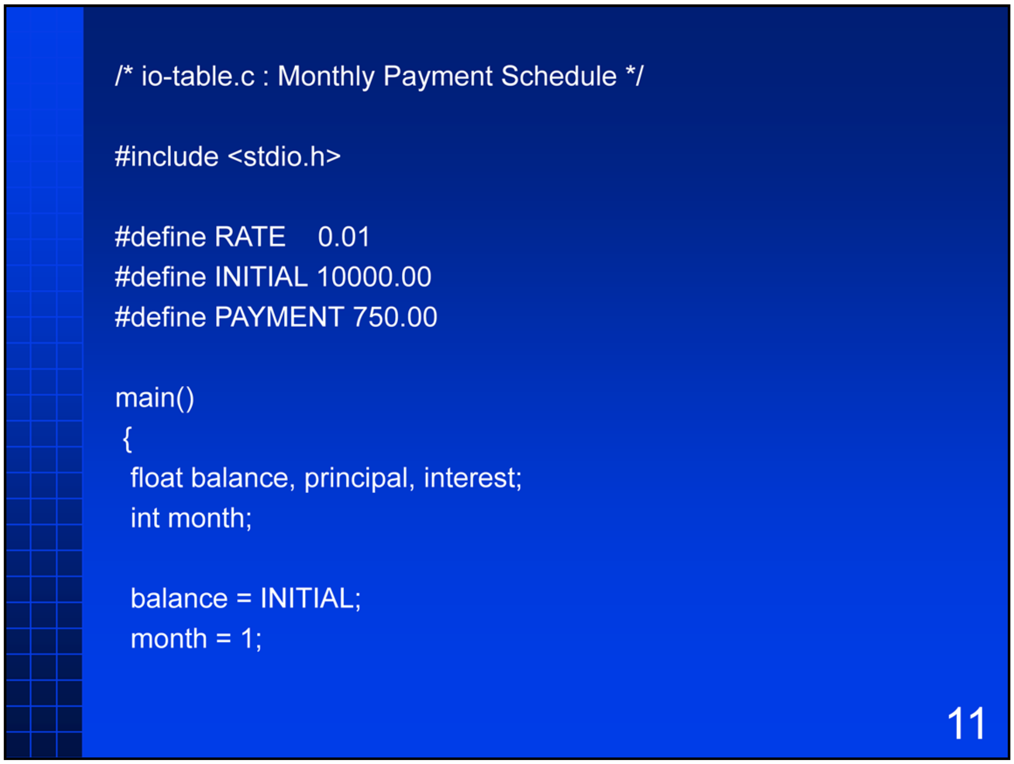
4) pads with blanks on the left (right-aligned by default)

5) left-aligned with minus, pads on right with blanks

6) format: width.maximum, used to truncate part of string that is displayed; right-aligned by default

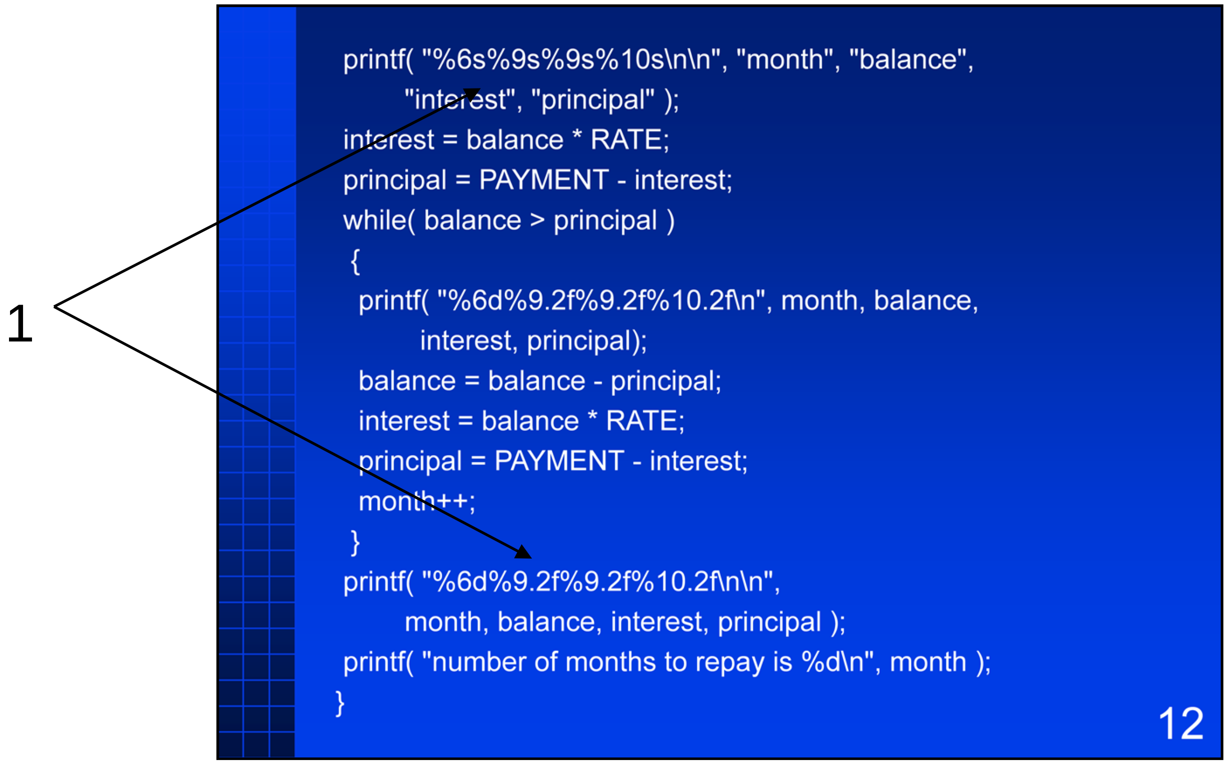
7) same as above, left aligned

8) no width, but truncation: common use to display first “n” characters



```
/* io-table.c : Monthly Payment Schedule */

#include <stdio.h>

#define RATE    0.01
#define INITIAL 10000.00
#define PAYMENT 750.00

main()
 {
  float balance, principal, interest;
  int month;

  balance = INITIAL;
  month = 1;
```

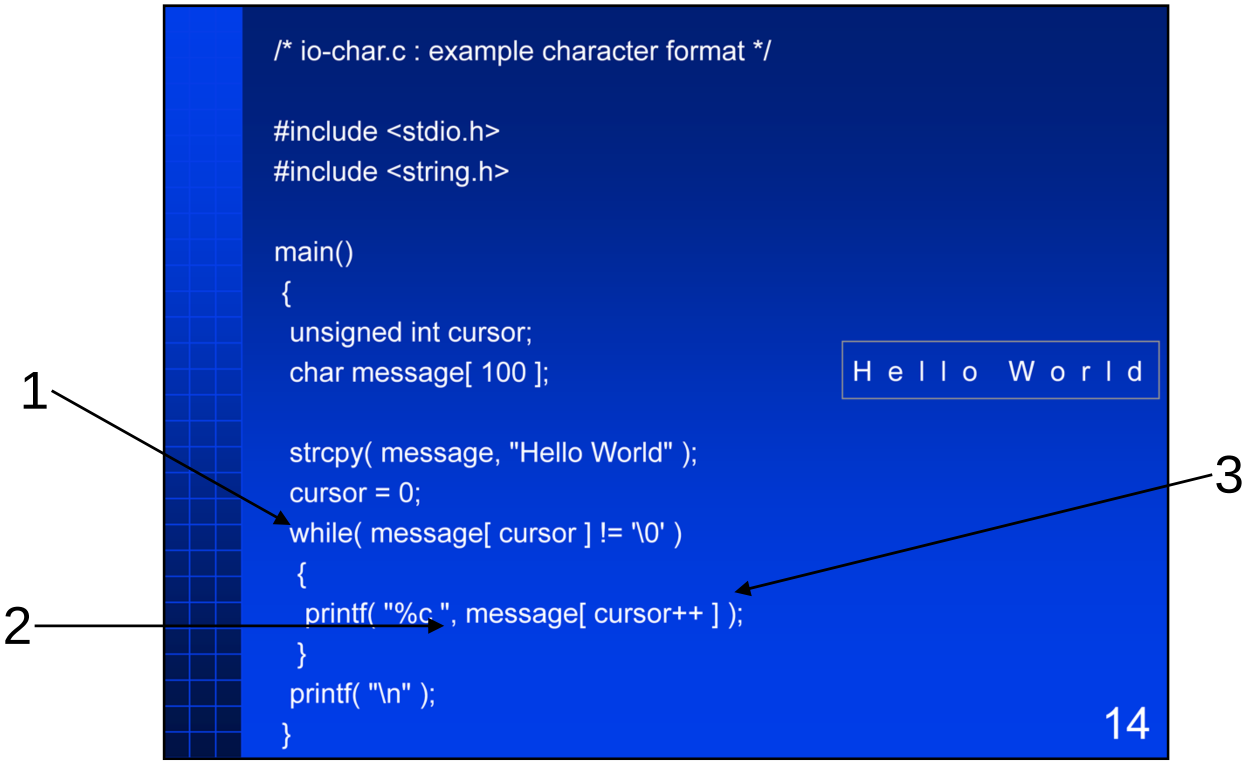
sample program that incorporates many of the ideas. employs a standard technique to guarantee columns with proper spacing.



```
printf( "%6s%9s%9s%10s\n\n", "month", "balance",
        "interest", "principal" );
interest = balance * RATE;
principal = PAYMENT - interest;
while( balance > principal )
 {
  printf( "%6d%9.2f%9.2f%10.2f\n", month, balance,
          interest, principal);
  balance = balance - principal;
  interest = balance * RATE;
  principal = PAYMENT - interest;
  month++;
 }
printf( "%6d%9.2f%9.2f%10.2f\n\n",
        month, balance, interest, principal );
printf( "number of months to repay is %d\n", month );
}
```

1) note use of equal field width in both places to ensure columns, in particular, substituting constants in field widths.

```
month   balance interest principal

    1 10000.00   100.00    650.00
    2  9350.00    93.50   656.50
    3  8693.50    86.93   663.07
    4  8030.43    80.30   669.70
    5  7360.73    73.61   676.39
    6  6684.34    66.84   683.16
    7  6001.18    60.01   689.99
    8  5311.19    53.11   696.89
    9  4614.30    46.14   703.86
   10  3910.44    39.10   710.90
   11  3199.55    32.00   718.00
   12  2481.54    24.82   725.18
   13  1756.36    17.56   732.44
   14  1023.92    10.24   739.76
   15   284.16     2.84   747.16

number of months to repay is 15
```



```
/* io-char.c : example character format */

#include <stdio.h>
#include <string.h>

main()
 {
  unsigned int cursor;
  char message[ 100 ];

  strcpy( message, "Hello World" );
  cursor = 0;
  while( message[ cursor ] != '\0' )
   {
    printf( "%c ", message[ cursor++ ] );
   }
  printf( "\n" );
 }
```

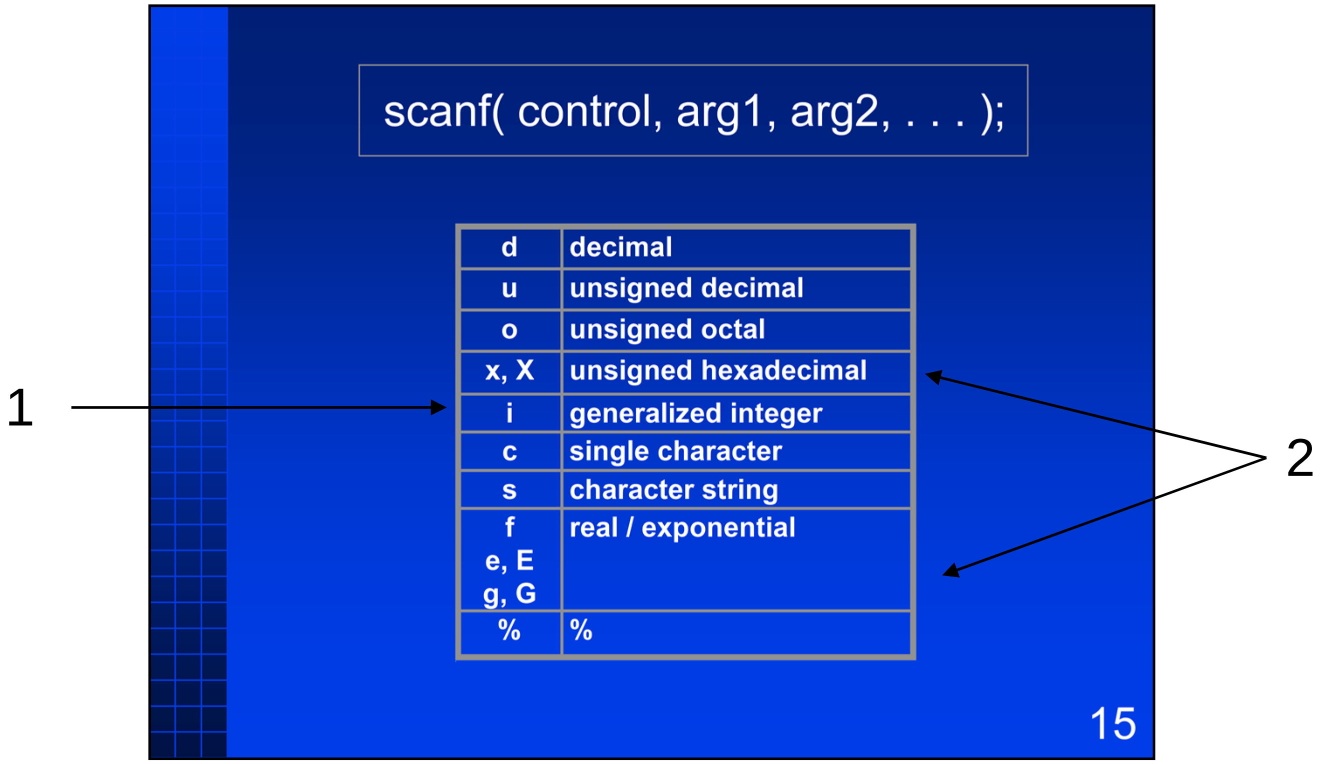
```
H e l l o   W o r l d
```

printing character strings on character at a time: more array preview

1) loop starts at first character and proceeds until nullchar encountered.

2) uses %c formatting directive to display single character

3) note use of post auto increment



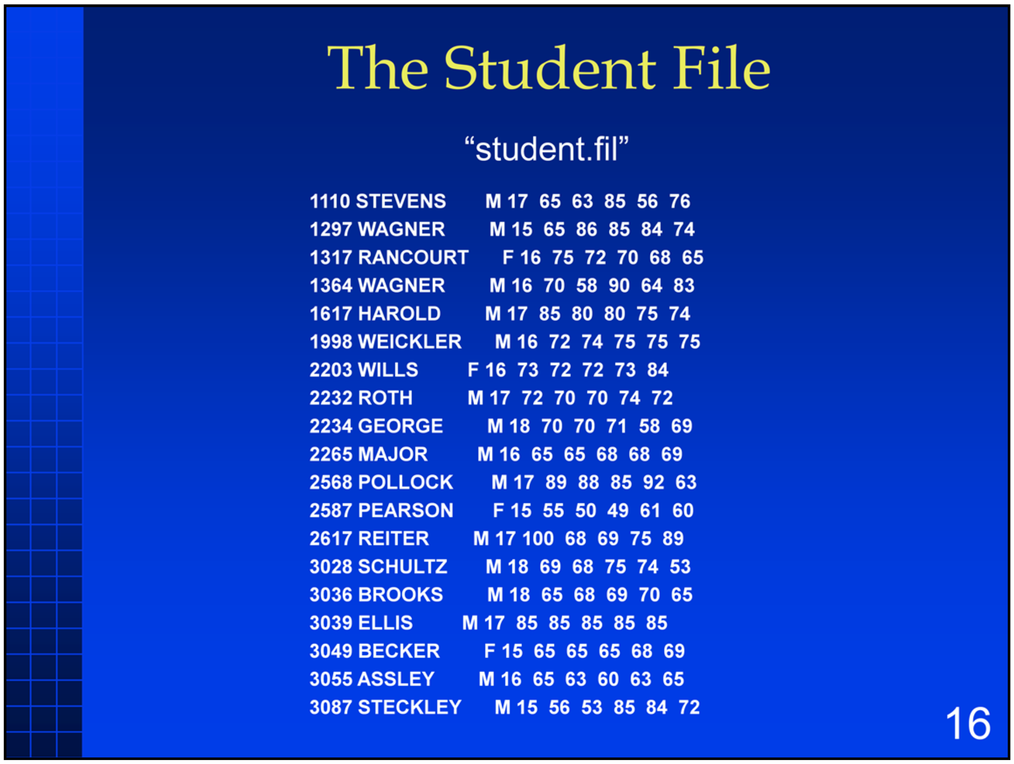
scanf(control, arg1, arg2, . . .);

d	decimal
u	unsigned decimal
o	unsigned octal
x, X	unsigned hexadecimal
i	generalized integer
c	single character
s	character string
f e, E g, G	real / exponential
%	%

scanf directives, much the same idea as printf ones.

1) %d for decimal (base 10) integers; %i for integral values of any base (in which case, must follow rules for numbers)

2) no distinction in case (both allowed for completeness)



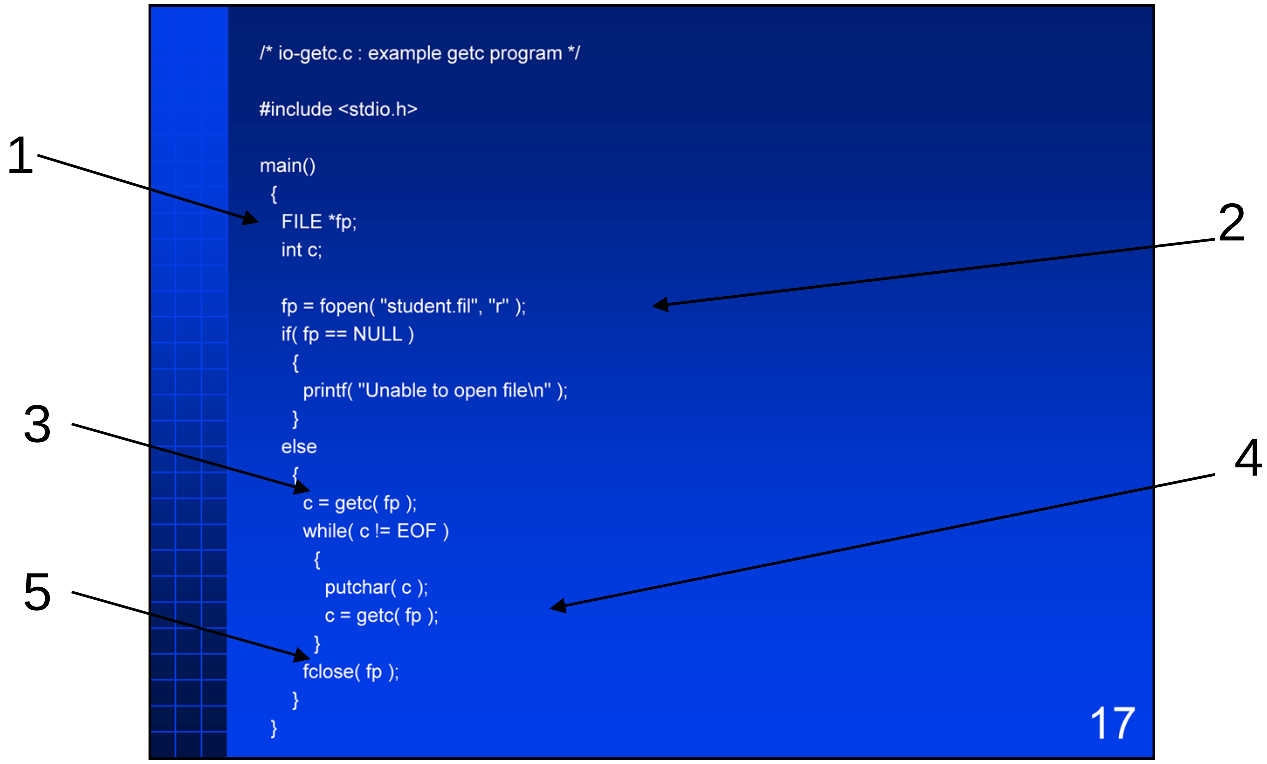
The Student File

“student.fil”

```
1110 STEVENS      M 17  65  63  85  56  76
1297 WAGNER        M 15  65  86  85  84  74
1317 RANCOURT      F 16  75  72  70  68  65
1364 WAGNER        M 16  70  58  90  64  83
1617 HAROLD       M 17  85  80  80  75  74
1998 WEICKLER      M 16  72  74  75  75  75
2203 WILLS        F 16  73  72  72  73  84
2232 ROTH         M 17  72  70  70  74  72
2234 GEORGE        M 18  70  70  71  58  69
2265 MAJOR        M 16  65  65  68  68  69
2568 POLLOCK       M 17  89  88  85  92  63
2587 PEARSON       F 15  55  50  49  61  60
2617 REITER       M 17 100  68  69  75  89
3028 SCHULTZ      M 18  69  68  75  74  53
3036 BROOKS        M 18  65  68  69  70  65
3039 ELLIS        M 17  85  85  85  85  85
3049 BECKER       F 15  65  65  65  68  69
3055 ASSLEY       M 16  65  63  60  63  65
3087 STECKLEY      M 15  56  53  85  84  72
```

Standard files are OK, move on now to look at permanent-file (disk-file) processing. Will be looking at text-files eg the student file. lines of text arranged into columns or fields. Processing will be sequential (start at beginning of file, move forward until end-of-file).

other file-access methods possible, use a different library (eg binary, random-access)



```
/* io-getc.c : example getc program */

#include <stdio.h>

main()
 {
  FILE *fp;
  int c;

  fp = fopen( "student.fil", "r" );
  if( fp == NULL )
   {
    printf( "Unable to open file\n" );
   }
  else
   {
    c = getc( fp );
    while( c != EOF )
     {
      putchar( c );
      c = getc( fp );
     }
    fclose( fp );
   }
 }
```

Read student-file and copy verbatim onto standard output (display file)

1) magic variable declaration (pointer)

2) stdio function fopen: open the named file (name is sysdep) for read “r”. returns constant NULL (defined in stdio) if failure, some magical value otherwise (don’t care what)

3) get a character from the file and put into variable c.if there is no character available, put character constant EOF (defined in stdio) into var c.

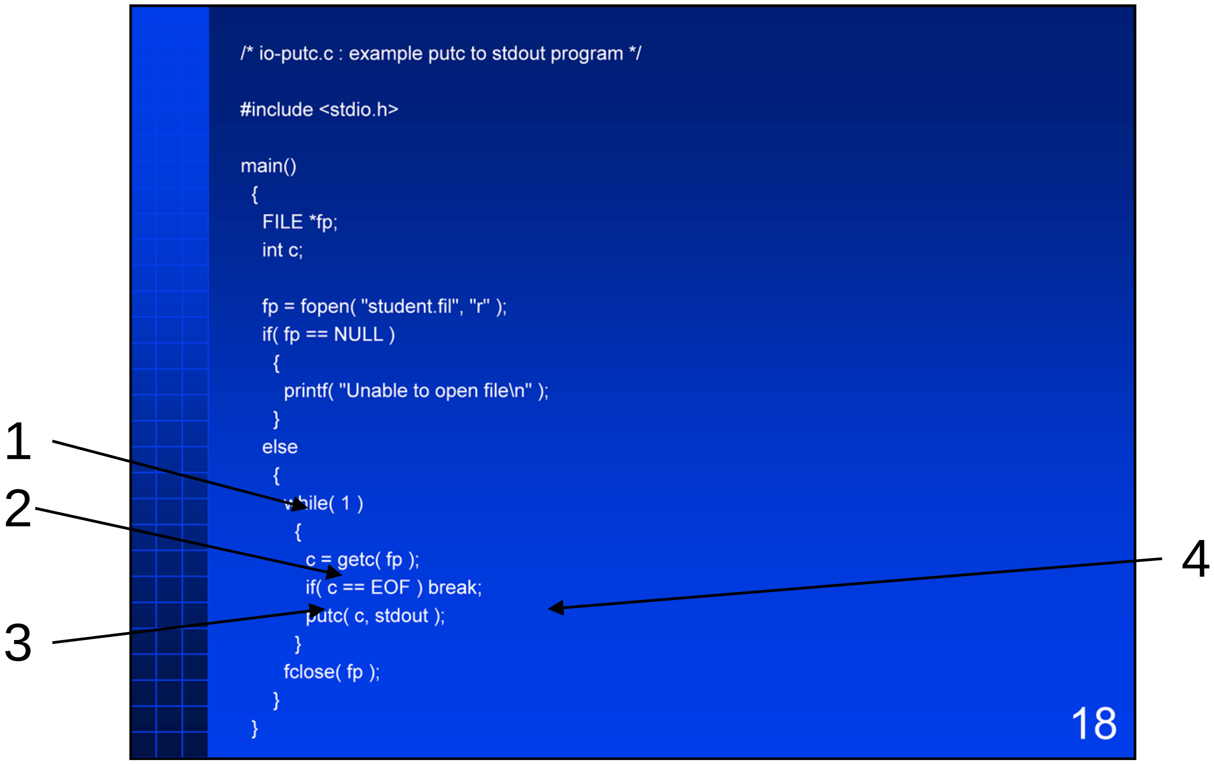
- note that c is declared as a int. allows arbitrary character codes. In particular, allows EOF to be de fined as a value that is not any character

4) putchar: put a character on stdout

5) close the file

Note we pay no at tention for line structure, no \n processing, just copy char-for-char. if \n encountered, treated and normal char and written. equivalent to printf(\n”);

This is a UNIX-ism, view file as sequence of characters with no particular structure, newline may have an interpretation on some devices, others not.



```
/* io-putc.c : example putc to stdout program */

#include <stdio.h>

main()
 {
   FILE *fp;
   int c;

   fp = fopen( "student.fil", "r" );
   if( fp == NULL )
     {
       printf( "Unable to open file\n" );
     }
   else
     {
       while( 1 )
         {
           c = getc( fp );
           if( c == EOF ) break;
           putc( c, stdout );
         }
       fclose( fp );
     }
 }
```

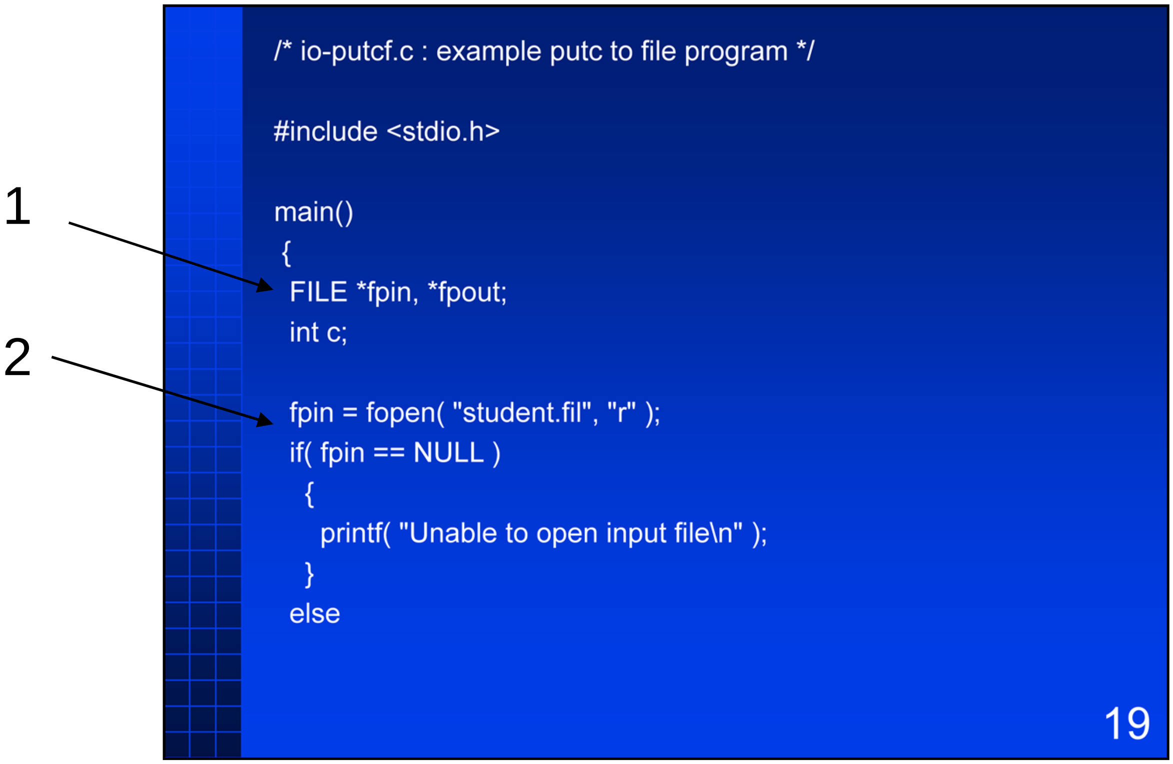
same function, some variations:

1) infinite loop: 1 is non-zero whbich is true

2) get out of loop: break. exist from closest-enclosing loop construct. note style of typing

3) put a character on specified file. in this case, file is “stdout”, so this is equivalent to putchar(c)

4) stdout is declared in stdio.h, declared as FILE *stdout



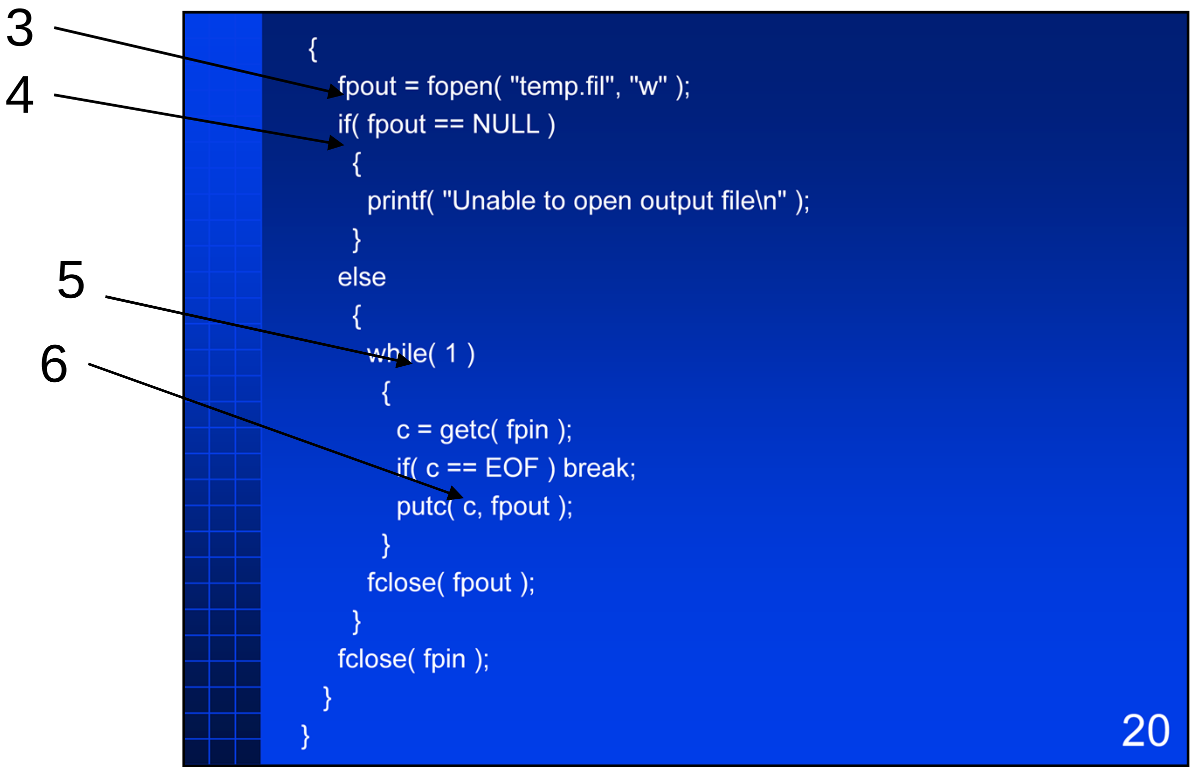
```
/* io-putcf.c : example putc to file program */

#include <stdio.h>

main()
 {
  FILE *fpin, *fpout;
  int c;

  fpin = fopen( "student.fil", "r" );
  if( fpin == NULL )
    {
      printf( "Unable to open input file\n" );
    }
  else
```

19

now, write to a different place than stdout: make a copy of the disk file

1) need two file variables: input file and output file

2) open input file and make sure it opened OK



```
 {
   fpout = fopen( "temp.fil", "w" );
   if( fpout == NULL )
     {
       printf( "Unable to open output file\n" );
     }
   else
     {
       while( 1 )
         {
           c = getc( fpin );
           if( c == EOF ) break;
           putc( c, fpout );
         }
       fclose( fpout );
     }
   fclose( fpin );
  }
}
```

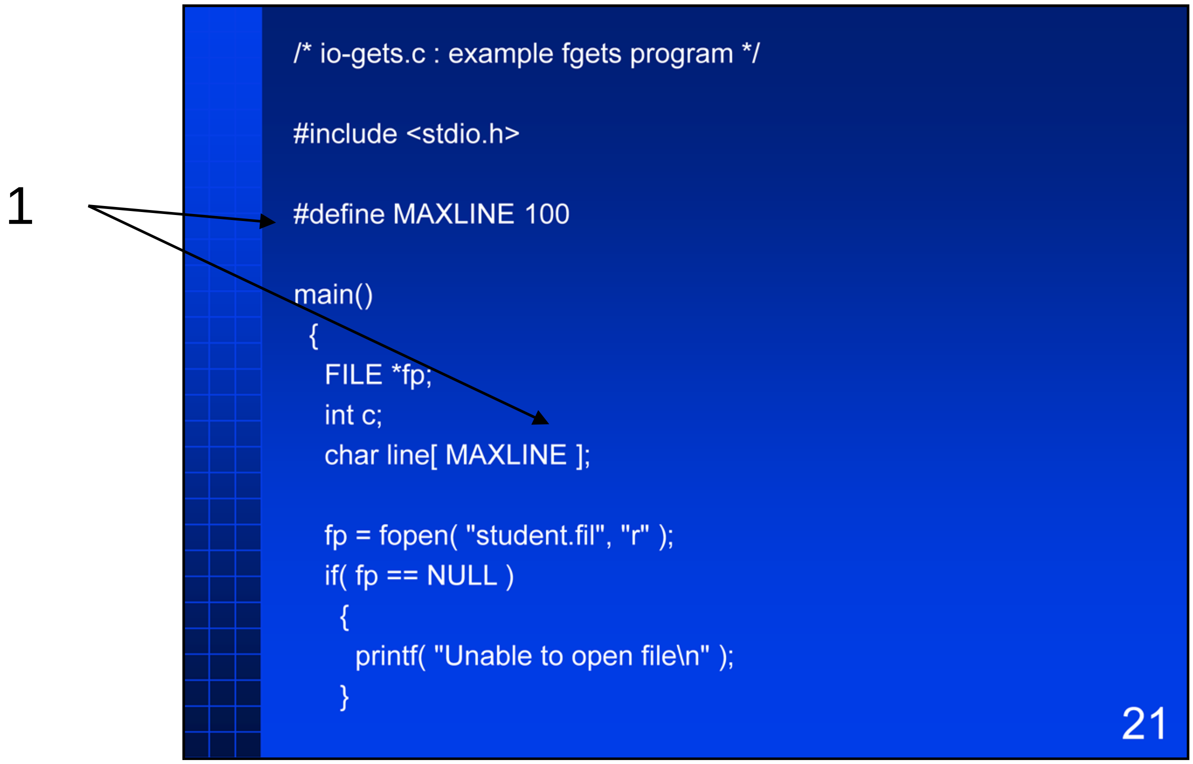
3) open output file: name is “temp.fil”, mode is “w” for write

4) as opening for read, make sure file opens OK

5) loop has same structure: copy characters, ignore line structure

6) reference our fpout instead of stdout

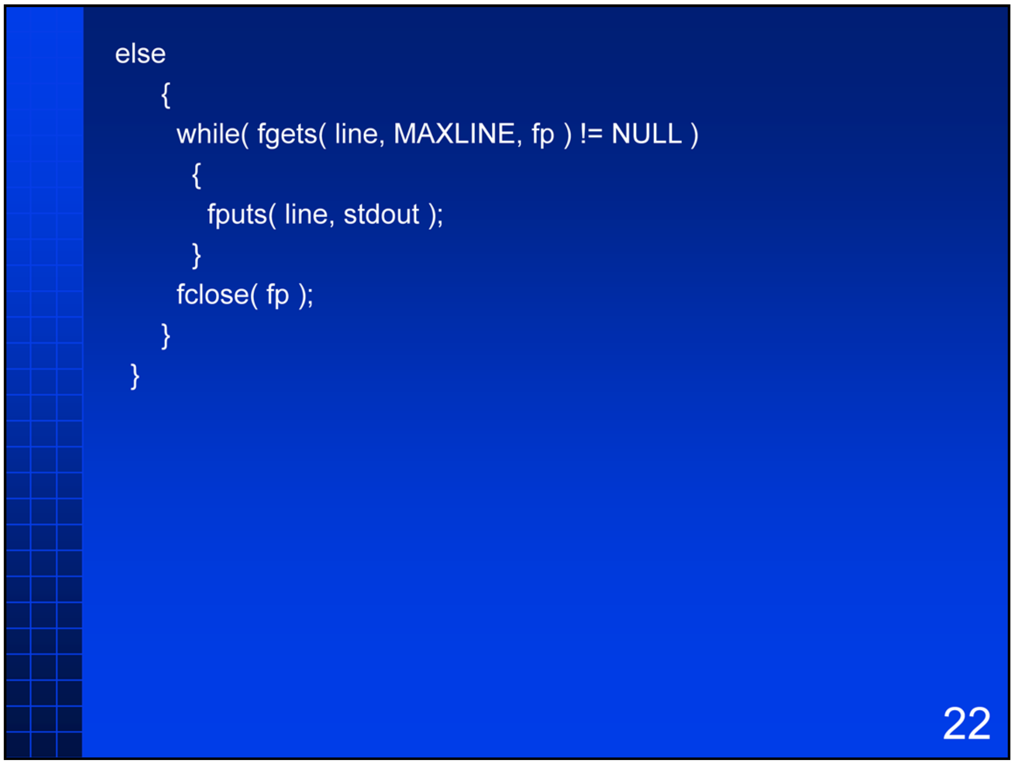
also: make sure not to close a file that didn’t open, so have to pay attention to nesting etc.



```
/* io-gets.c : example fgets program */

#include <stdio.h>

#define MAXLINE 100

main()
 {
  FILE *fp;
  int c;
  char line[ MAXLINE ];

  fp = fopen( "student.fil", "r" );
  if( fp == NULL )
   {
    printf( "Unable to open file\n" );
   }
```

sometimes want to maintain record structure. program to read file one line at a time and dispay on stdout

1) declare a string variable to be used as a line/record buffer. use a symbolic constant for the length, since we’ll need this later

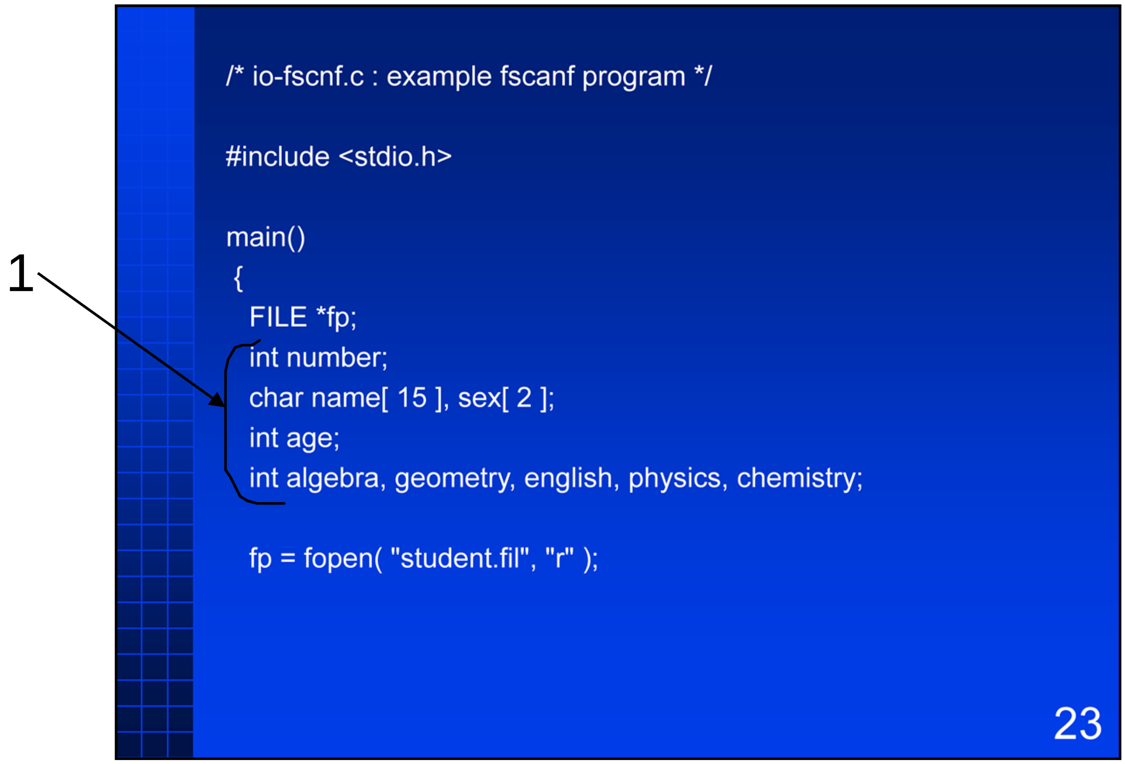


```
else
    {
      while( fgets( line, MAXLINE, fp ) != NULL )
        {
          fputs( line, stdout );
        }
      fclose( fp );
    }
  }
```

2) fgets: gets a string up to and including \n (if any) from a file. destination is "line", source is fp. read no more that MAXLINE characters (use of symbolic constant in both places guarantees no buffer overrun)

3) put a string on given file (stdout here). note that string already contains a \n, so no additional stuff

4) loop termination: fgets returns null if eof or error, so keep going as long as fgets doesn't return null

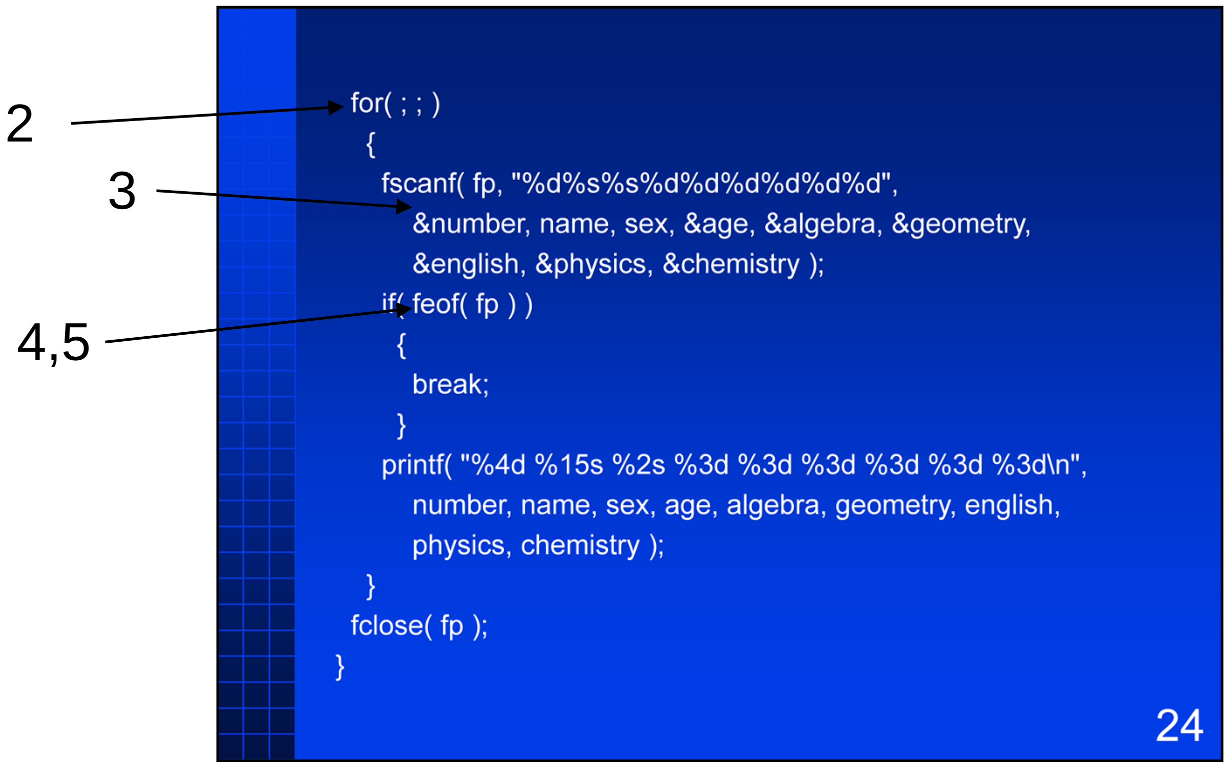


```
/* io-fscnf.c : example fscanf program */

#include <stdio.h>

main()
 {
   FILE *fp;
   int number;
   char name[ 15 ], sex[ 2 ];
   int age;
   int algebra, geometry, english, physics, chemistry;

   fp = fopen( "student.fil", "r" );
```

can use scanf to get at individual fields

construct control string to match record layout

1) declare bunch of variables to receive fields



```
  for( ; ; )
    {
      fscanf( fp, "%d%s%s%d%d%d%d%d%d",
          &number, name, sex, &age, &algebra, &geometry,
          &english, &physics, &chemistry );
      if( feof( fp ) )
        {
          break;
        }
      printf( "%4d %15s %2s %3d %3d %3d %3d %3d %3d\n",
          number, name, sex, age, algebra, geometry, english,
          physics, chemistry );
    }
  fclose( fp );
}
```

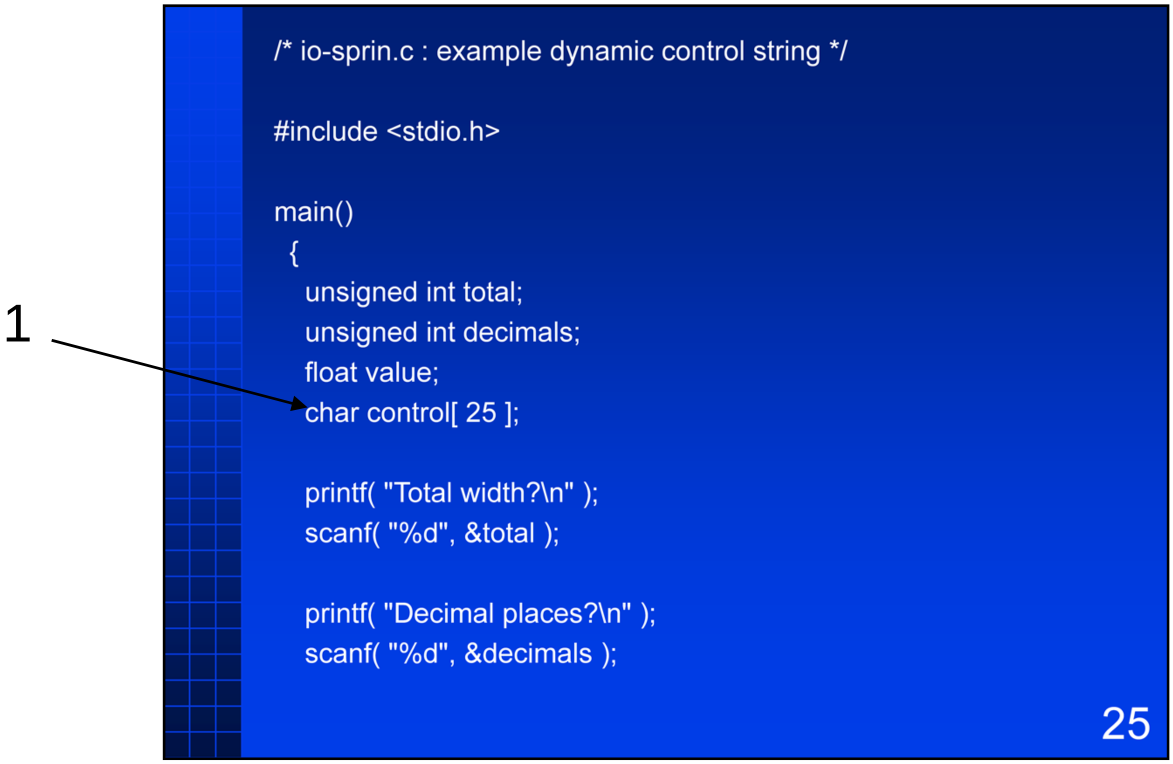
2) infinite loop with for statement instead of while

3) fscanf instead of scanf: first parameter is file variable, rest are the same. note mixture of & and not &

- note also that data file is carefully constructed to ensure that each field is blank-delimited.

4) unlike getchar, gets, have no indication from scanf about EOF. have to test explicitly: feof returns true if no more characters in the file.

5) alternate style for if -- break

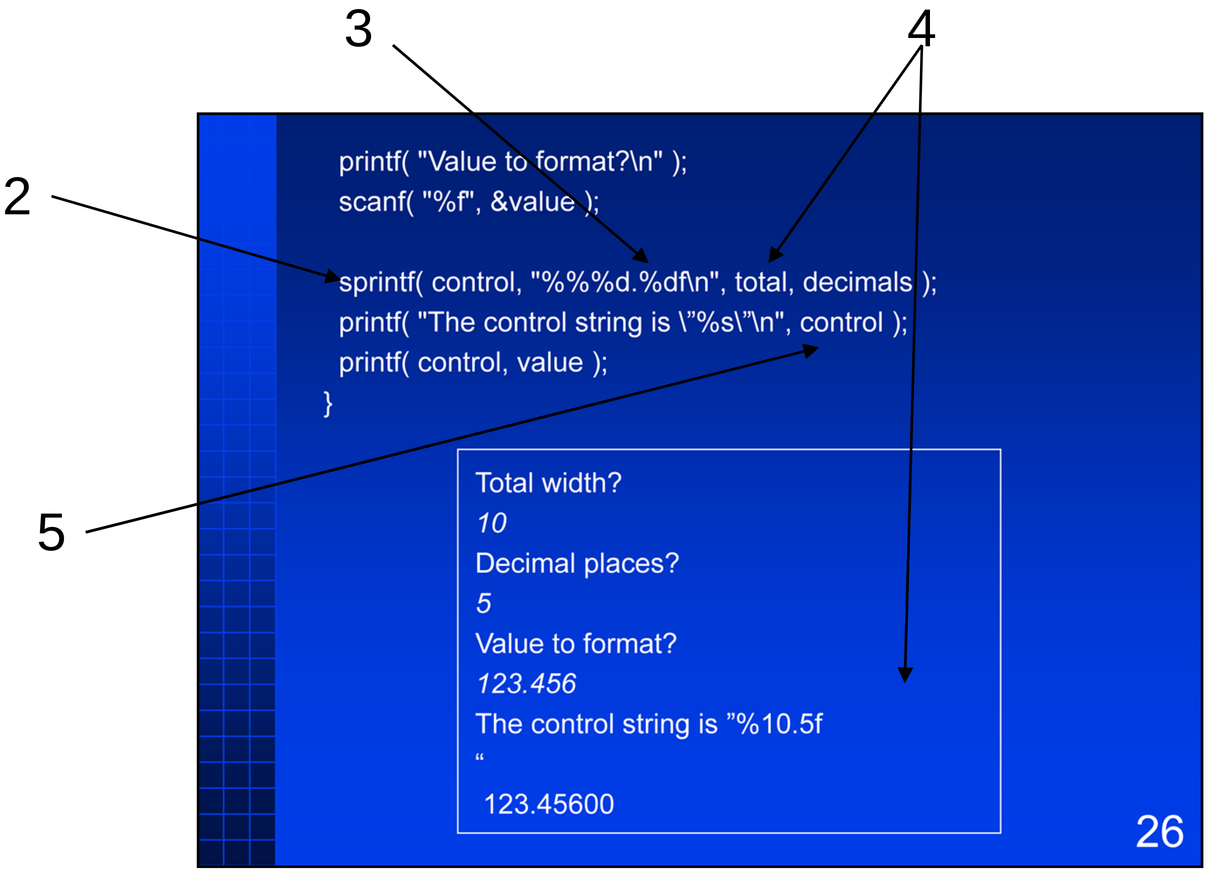


```
/* io-sprin.c : example dynamic control string */

#include <stdio.h>

main()
 {
   unsigned int total;
   unsigned int decimals;
   float value;
   char control[ 25 ];

   printf( "Total width?\n" );
   scanf( "%d", &total );

   printf( "Decimal places?\n" );
   scanf( "%d", &decimals );
```

Remember that control strings are interpreted at run-time. They can be created and modified at run-time.

Example creates a control string from input provided by the user, then displays a number according to that control string.

1) string variable that will contain the control string, 25 is arbitrary.



```
   printf( "Value to format?\n" );
   scanf( "%f", &value );

   sprintf( control, "%%%d.%df\n", total, decimals );
   printf( "The control string is \"%s\"\n", control );
   printf( control, value );
}
```

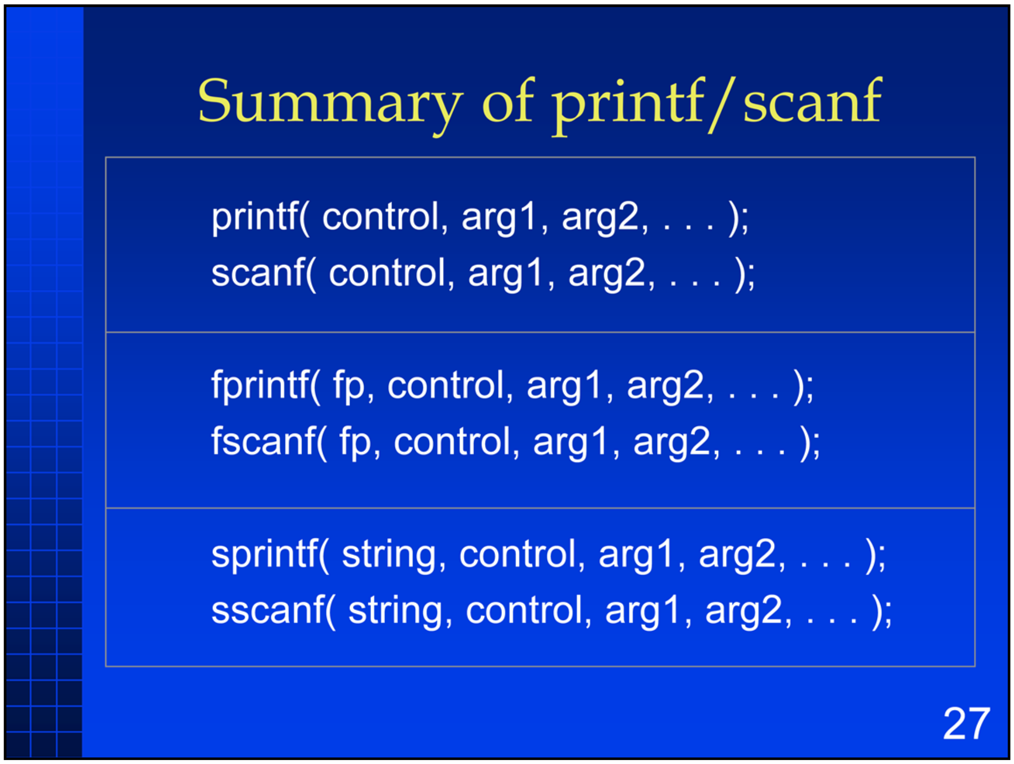
```
Total width?
10
Decimal places?
5
Value to format?
123.456
The control string is ”%10.5f
“
 123.45600
```

2) sprintf: like printf, except that target is not a file, target is a string variable

3) want to produce a control string with a %, so need %% [go over control string char by char]

4) note \n in created control string; note how displayed

5) use \” to display a “



Summary of printf/scanf

```
printf( control, arg1, arg2, . . . );
scanf( control, arg1, arg2, . . . );
```

```
fprintf( fp, control, arg1, arg2, . . . );
fscanf( fp, control, arg1, arg2, . . . );
```

```
sprintf( string, control, arg1, arg2, . . . );
sscanf( string, control, arg1, arg2, . . . );
```

printf/scanf to standard files, arbitrary files, string variables.