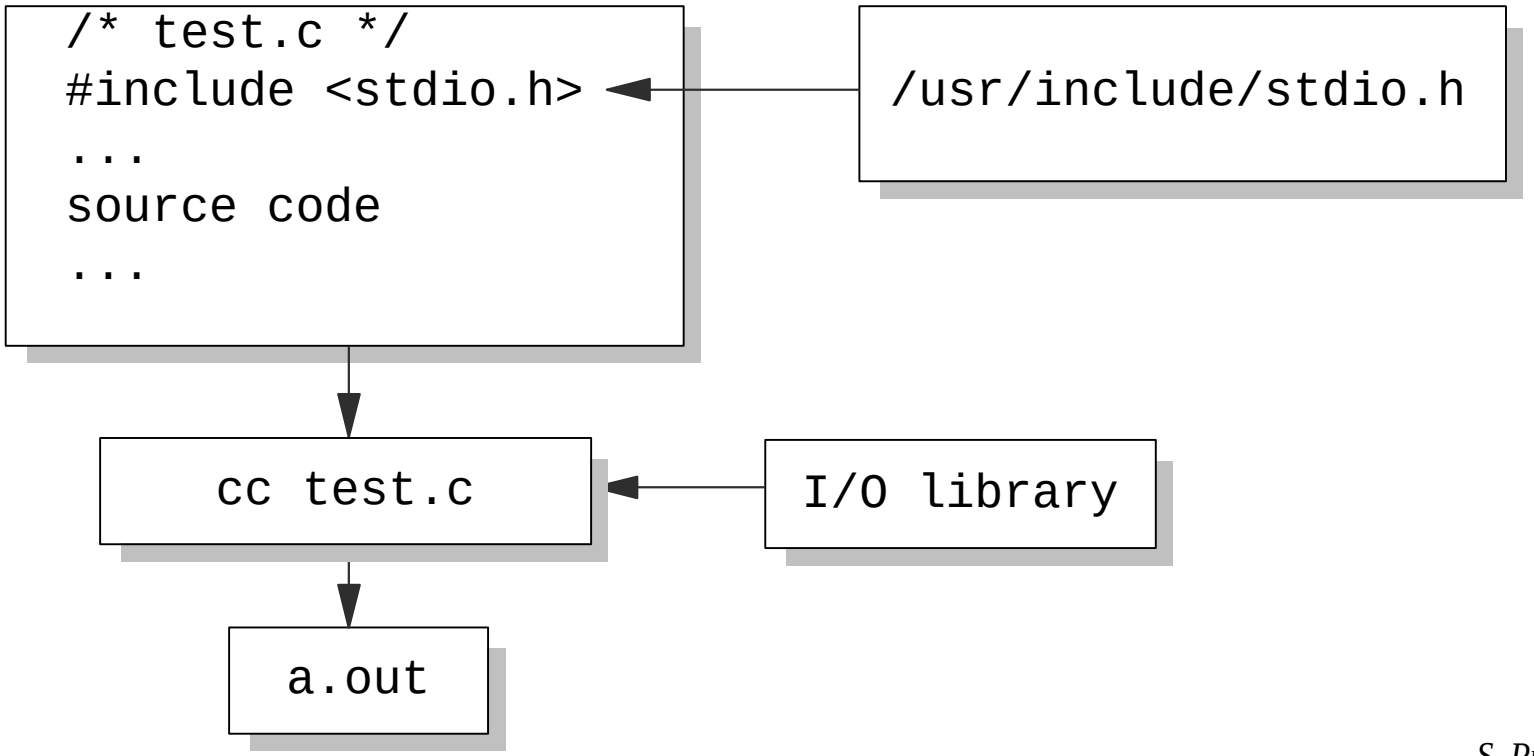
Outline

- Standard input and output
- Formatted output
- Formatted input
- File access

Standard Input/Output Library

- `stdio.h` defines macros and variables used by the I/O library.
- `stdio.h` is typically kept in */usr/include* (UNIX).



`getchar` and `putchar`

- `getchar()` returns the next input character from the standard input.
- `putchar(c)` puts the character c on the standard output.

```
#include <stdio.h>

void main()
{
  int    c;

  while( (c=getchar()) != EOF )
    putchar( isupper(c) ? tolower(c) : c );
}
```

puts

- `puts(s)` puts the string s on the standard output.
- automatically adding a newline at the end.
- car

```
#include <stdio.h>

void main()
{
  int    c;

  puts("--- enter text now ---");
  while( (c=getchar()) != EOF )
    putchar( islower(c) ? toupper(c) : c );
  puts("--- bye bye ---");
}
```

gets

- get a line of text from the standard input.
- return a pointer to the string.

```
#include <stdio.h>

void main()
{
  int    buffer[80];

  puts( gets( buffer ) );
}
```

Formatted Output : `printf`

- prints the formatted information to the standard output.
- format string
  - ordinary characters
  - conversion specifiers

```
printf( format-string );
printf( format-string, arguments );
```

```
printf("This is a printf statement");
printf("\nThis\tis\ta\tprintf\tstatement");
printf("This is a %s statement\n", sText);
```

Conversion Specifiers : `printf`

Specifier	Meaning
%c	Single character
%d	Signed decimal integer
%f	Decimal floating point number
%s	Character string
%u	Unsigned decimal integer
%%	% symbol

```
printf("Product: %s, price = %f, amount = %d\n",
        sProduct, fPrice, iAmount );

printf("%u\t%c\t%f%%\n, a, func1(), c+d );
```

Conversion Specifiers : `printf`

```
printf("Product: %s, price = %f, amount = %d\n",
        sProduct, fPrice, iAmount );
```

```
Product: Ma-ma, price = 5.250000, amount = 50
```

```
Product: Wai-wai, price = 4.500000, amount = 500
```

Conversion Specifiers : `printf`

Specifier	Meaning
%o	Octal integer
%x	Hexadecimal integer
%e	Decimal floating point number of the form m.nnnnnnExx (e.g. 2.35E-6)
%g	use %e or %f, whichever is shorter

Formatted Input : `scanf`

- read data from the standard input.
- also use a format string for direct interpretation of input sequences (conversion).

```
scanf( format-string, arguments );
```

```
int    x, y;
float  p, q;

scanf( "%d", &x );
scanf( "%d %d %f", &x, &y, &p );

scanf( "%d", &q );  /* error */
scanf( "%f", &y );  /* error */
```

Formatted Input : scanf

- `scanf()` uses whitespace to separate input to fields.
- Whitespace can be spaces, tabs, or new lines.

```
char     cCode;
float    fPrice

scanf( "%d %f", &cCode, &fPrice );
```

```
A  10.55
```

```
A                    10.55
```

```
A
 10.55
```

```
A10.55
```

Conversion Specifiers : `scanf`

Specifier	Meaning
%c	Single character
%d	Signed decimal integer
%f	Decimal floating point number
%e	Floating point number of the form such as -267.344e6
%s	Character string
%o	Octal integer (with or without a leading zero)
%x	Hexadecimal integer (with or without a leading 0x)
%h	Short integer

```
%ld, %lo, %lx, %lf, %le
```

Conversion Specifiers : scanf

- the character %
- an optional assignment suppression character *
- an optional number specifying a maximum field width.
- a conversion character

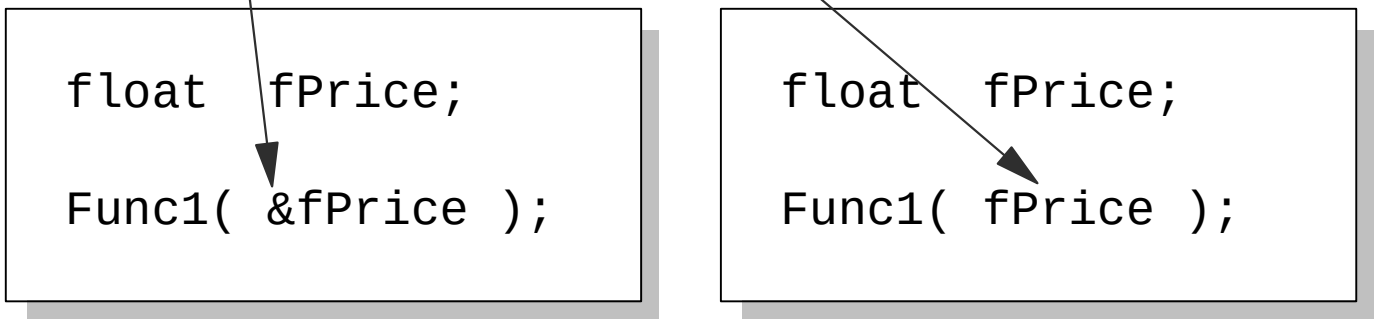
```
int      i;
float    x;
char     name[50];
scanf( "%2d %f %*d %2s", &i, &x, name );
```

```
56789 0123 45a72
```

```
%2d  - read in two characters as an integer.
%*d  - skip over the next integer.
%10s - read in the next ten characters stored as a string
```

Address of Operator : &

- Values of the arguments to scanf are changed according to the input.
- C function can only changes values of the variables in a calling function via addresses of the variables
- Pass address, not value



Address of Operator : &

- Name of a variable defined as an *array* is the address of the zero-th element in the array.

```
int     iData[100];

scanf( "%d", iData );

scanf( "%d", &(iData[0]) );

scanf( "%d", &(iData[78]) );

scanf( "%d", &iData ); /* error */
```

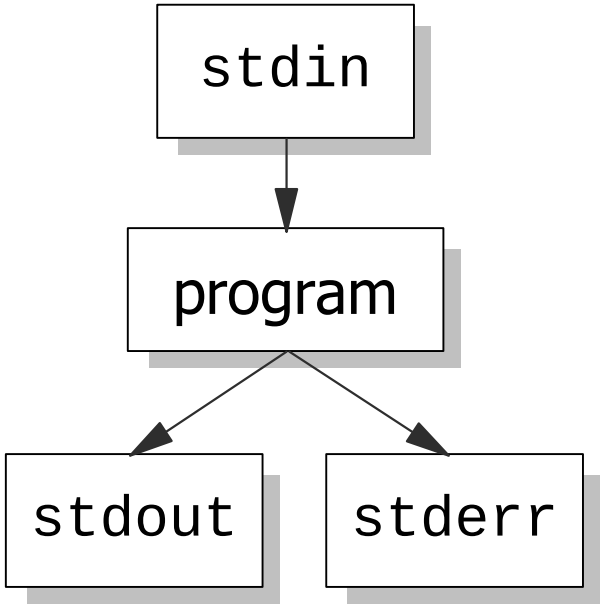
Q & A

- Why should I use `puts()` if `printf()` does everthing `puts()` does and more ?
- Why do I need to input `STDIO.H` when I use `printf()`, `puts()`, or `scanf()` ?
- What happens if I leave the address of operator (`&`) off a `scanf()` variable ?

File Access

- Standard input and output
  - I/O redirection
  - pipe mechanism
- Named files

```
csh> cat
csh> cat < infile > outfile
csh> cat infile | wc
csh> cat infile1 infile2
csh> cp infile outfile
```



Standard Input and Output

- Read only one input stream
- Write only one output stream
- Input / Output redirections
- Pipe mechanism
- Programs only aware of standard input and output.

```
csh> prog1 < infile
csh> prog2 > outfile
csh> prog2 | prog1
```

File Access

- `fopen` to get file pointer
- file input / output functions
- `fclose` to free the file pointer and flush buffer

```
FILE   *fp;

fp = fopen( "infile.dat", "r" );

   ...

fclose(fp);
```

Opening a File : `fopen`

- File name
- Operation mode
  - `"r"` : read
  - `"w"` : write
  - `"a"` : append

```
fp = fopen( filename, mode );
```

- return a file pointer or `NULL` if error
  - invalid filename
  - open a file on a disk that is not ready
  - open a nonexistent file in mode `"r"`

getc, putc

- `getc()` : place in c the next character from the referred to by fp, and EOF when it reaches end of file.

```
c = getc( fp );
```

- `putc()` : put the character c on the file fp and return c.

```
putc( c, fp );
```

```
#define getchar()    getc( stdin )
#define putchar(c)   putc( c, stdout )
```

type.c

```
#include   <stdio.h>

main( int argc, char *argv[] )
{
  int      c;
  FILE     *fp;

  if ( argc == 1 )
    printf("Usage : type <filename>\n");
  else
  if ( (fp = fopen(argv[1],"r")) == NULL )
    printf("type: can open %s\n", argv[1]);
  else {
    while ( (c = getc(fp)) != EOF )
      putc( c, stdout );
    fclose( fp );
  }
}
```

```
csh> type type.c

argc = 2
argv[0] = "type"
argv[1] = "type.c"
```

fprintf & fscanf

- `printf` : output to stdout
- `fprintf` : output to the specified file pointer
- `scanf` : input from stdin
- `fscanf` : input from the specified file pointer

```
fpIn  = fopen( "input.txt", "r" );
fpOut = fopen( "output.txt", "w" );
fscanf( fpIn, "%d %f", &i, &p );
fprintf( fpOut, "No: %d\tPoint: %f\n, i, p );

fprintf( stdout, "Enter your choice : " );
fscanf( stdin, "%d", &choice );
printf( "Enter your choice : " );
scanf( "%d", &choice );
```

type.c

```
#include   <stdio.h>

main( int argc, char *argv[] )
{
  int      c;
  FILE     *fp;

  if ( argc == 1 )
    fprintf(stderr, "Usage : type <filename>\n");
  else
  if ( (fp = fopen(argv[1],"r")) == NULL )
    fprintf(stderr, "type: can open %s\n", argv[1]);
  else {
    while ( (c = getc(fp)) != EOF )
      putc( c, stdout );
    fclose( fp );
  }
}
```