CSC180: Lecture 30

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

File I/O

File i/o function calls

```
fgets(buffer, n, file_handle)
```

- Reading lines (CStrings)
- buffer is where the line is stored
- n is the max number of characters to be stored in buffer
- *file_handle:* is address returned by `openf()`
- Reads characters from file and stores them in buffer
- Stops when ‘\n’ is reached or when n-1 characters have been read
- Returns NULL on failure and buffer on success

File i/o function calls

```
fputs(buffer, file_handle)
```

- Writing CStrings to file
- Writes the contents of buffer to file_handle
- *file_handle:* is address returned by `openf()`
- Writes each character until the ‘\0’ is reached
    - Does not write ‘\0’ to the file

File i/o function calls

- `fgetc`
    - Reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc(file_handle )` equivalent to `getchar()`

- `fputc`
    - Writes one character to a file
    - Takes a `FILE` pointer and a single character to write as an argument
    - `fputc( 'a', file_handle )`

      equivalent to `putchar( 'a' )`

Access File by File Pointer – feof

- check the end of file: feof

```
feof(file_handle)
```

- returns true if read has already failed due to EOF

Example:

```
while (!feof (infile))
  putc(getc(infile), outfile);
```

Example: File Copy: part 1

```
int copy( const char *destFile, const char *sourceFile ) {
int charsCounted = 0, ch;
FILE *sfp, *dfp;

if( strcmp( sourceFile, destFile ) == 0 ) {
  printf( "Cannot copy to self\n" );
  return -1;
}

if( ( sfp = fopen( sourceFile, "r" ) ) == NULL ) {
  printf( "Cannot open input file %s\n", sourceFile );
  return -1;
}

if( ( dfp = fopen( destFile, "w" ) ) == NULL ) {
  printf( "Cannot open output file %s\n", destFile );
  fclose( sfp ); return -1;
}
```

Part 2: Character at a Time

```
while( ( ch = getc( sfp ) ) != EOF )   {
    if( putc( ch, dfp ) == EOF )
    {
       printf( "Unexpected error during write.\n" );
       break;
    }
    else
       charsCounted++;
 }

 fclose( sfp );
 fclose( dfp );
 return charsCounted;
```

File Copy: Line at a Time

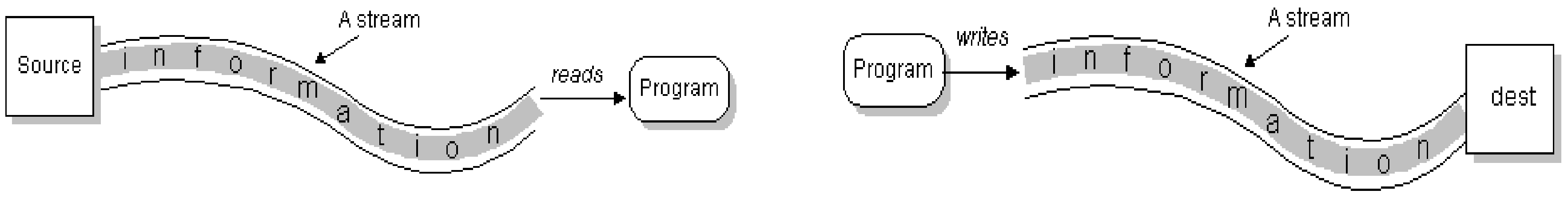
```
#define MAX_LINE_LEN 256
int copy( const char *destFile, const char *sourceFile )
{
  int charsCounted = 0;
  char oneLine[ MAX_LINE_LEN + 2 ];
  FILE *sfp, *dfp;
  // ... same start

  while( ( fgets( oneLine, MAX_LINE_LEN, sfp ) ) != NULL )
    if( fputs( oneLine, dfp ) < 0 ) {
      printf( "Unexpected error during write.\n" );
      break;
    }
    else
      charsCounted += strlen( oneLine );

  // ... same finish
}
```

Streams
&& Files

Stream I/O



- Stream – abstract concept of input and output
  - Sequence of data
  - Has a source or a destination

Reading and Writing

READ

```
open(stream);
while (more info)
  read(stream);
close(stream);
```

WRITE

```
open(stream);
while (more info)
  write(stream);
close(stream);
```

Types of Streams

- Byte Streams (Binary)
  - Operate on *bytes* (8-bit)
  - No further discussion

- Character Streams
  - Operate on 16-bit *characters*

Remember printf?

```
int main()
{
      printf(“Hello World \n”);
}
```

- Sends text to standard output - stdout
  (i.e. monitor)

What about Input?

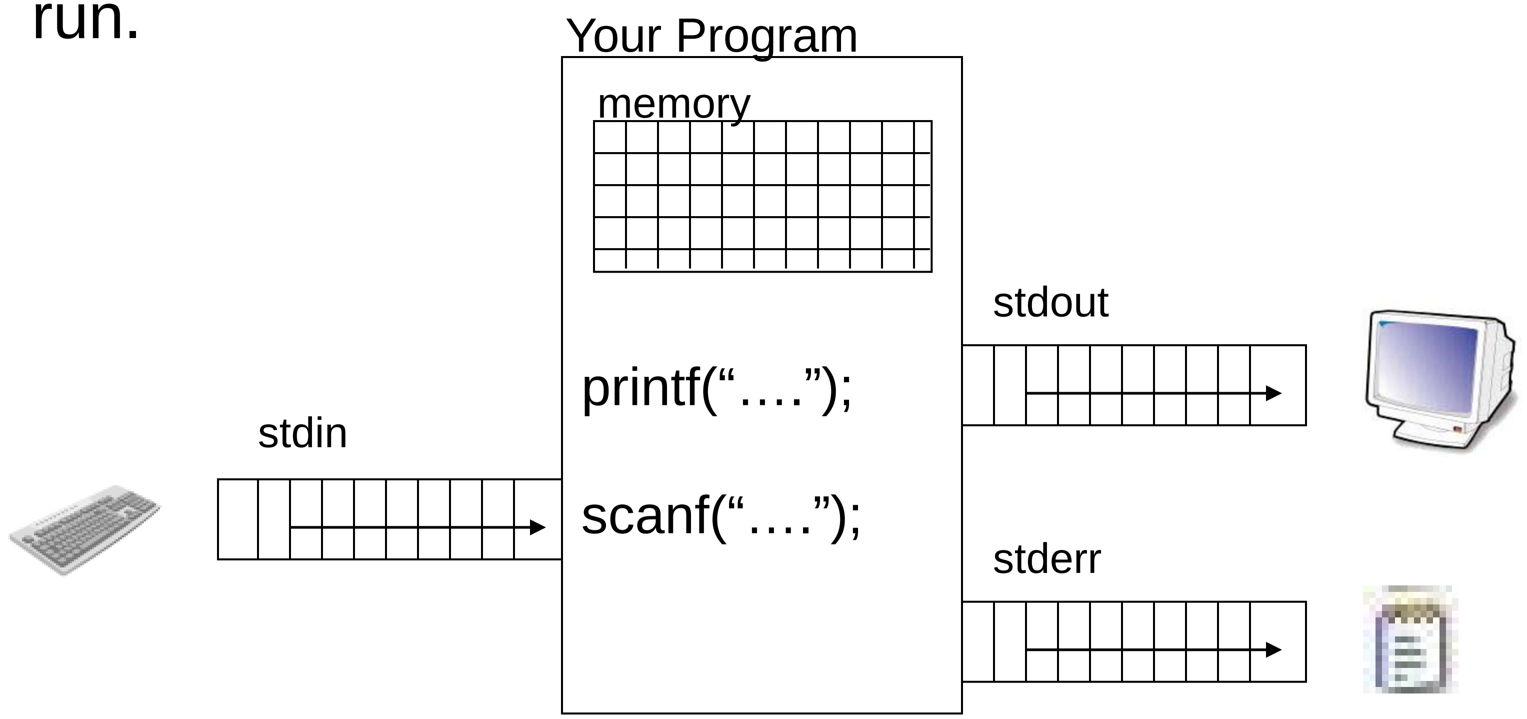
```
void main()
{
    char buffer[32];
    int i;

    scanf(“%s %d”, buffer, &i);
}
```

Scanf -- reads text from standard input - stdin
(i.e., keyboard)

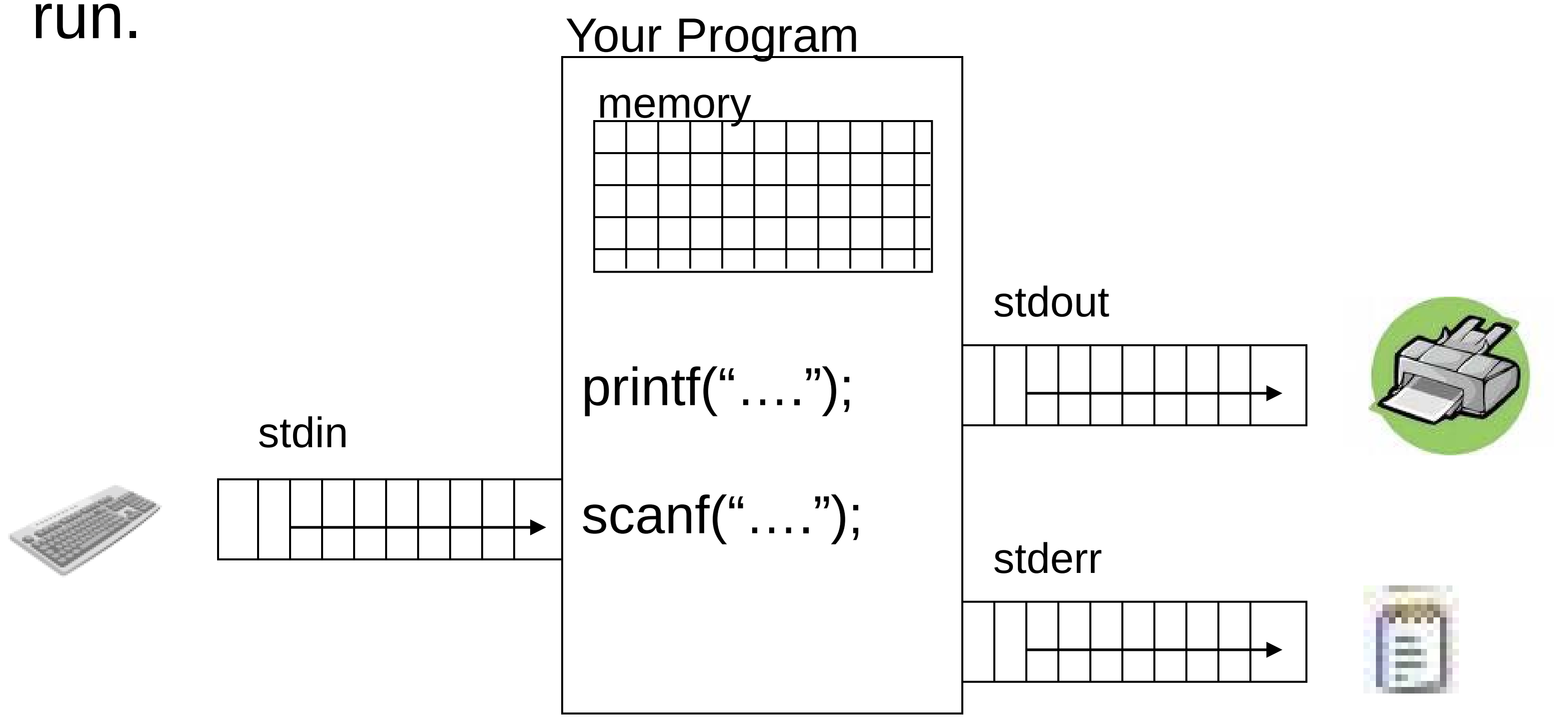
What are stdin, stdout, stderr?

- File descriptors...or more precisely a pointer to type FILE.
- These FILE descriptors are setup when your program is run.



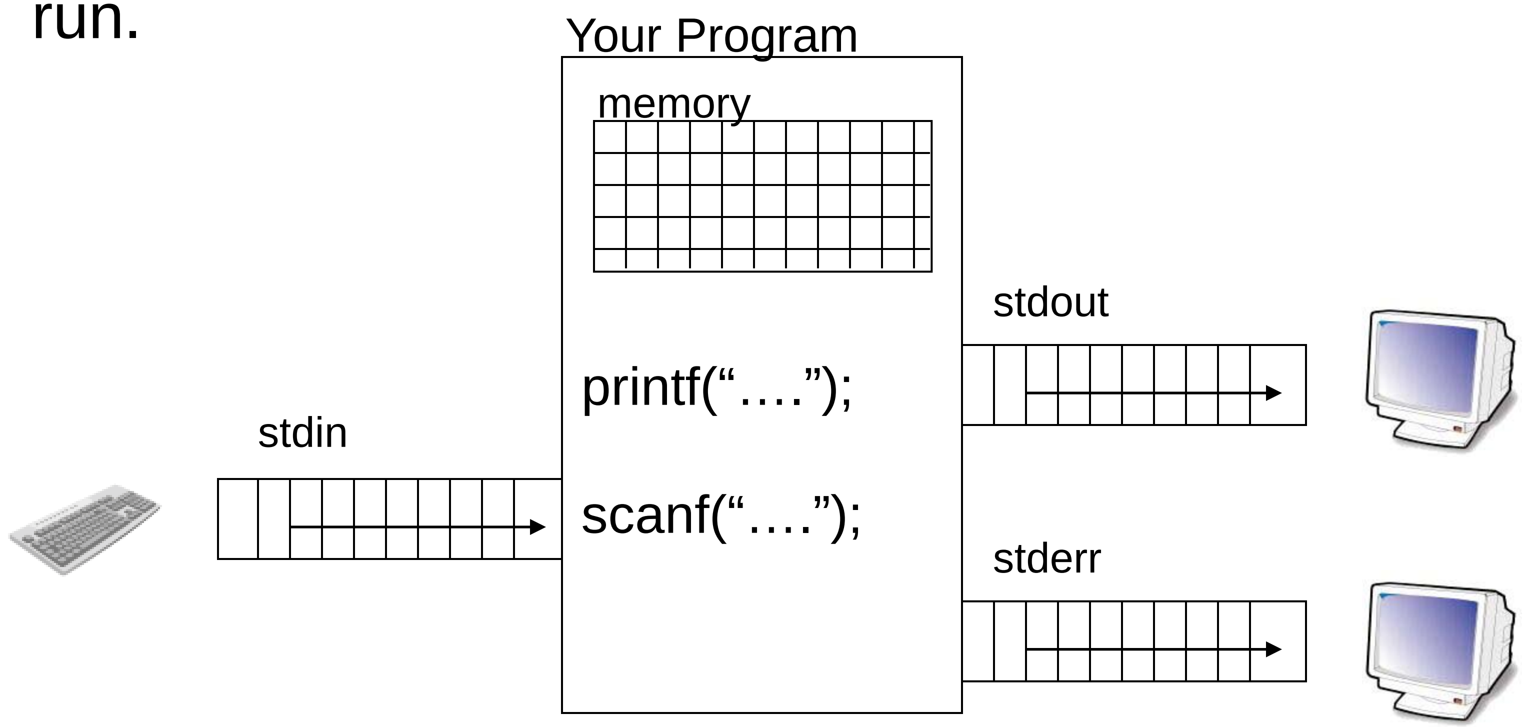
What are stdin, stdout, stderr?

- File descriptors...or more precisely a pointer to type FILE.
- These FILE descriptors are setup when your program is run.



What are stdin, stdout, stderr?

- File descriptors...or more precisely a pointer to type FILE.
- These FILE descriptors are setup when your program is run.



What are stdin, stdout, stderr?

- File descriptors...or more precisely a pointer to type FILE.
- These FILE descriptors are setup when your program is run.

```
int main (int argc, char* argv[])
{
   fprintf( stderr, "This is an error.\n" );
   fprintf( stdout, "This is correct.\n" );
   printf( "This is correct.\n" );
   fscanf(stdin, "%d", &SomeNum );
   fprintf(stdout, "%d\n", SomeNum );

     return 0;
}
```

FD 0: stdin
FD 1: stdout
FD 2: sterr

Passing Arguments to A Program

Passing data from the keyboard

- main() can also be written as

```
main(int argc, char *argv[])
```

- Information can be passed directly from the keyboard to the program at the time of execution

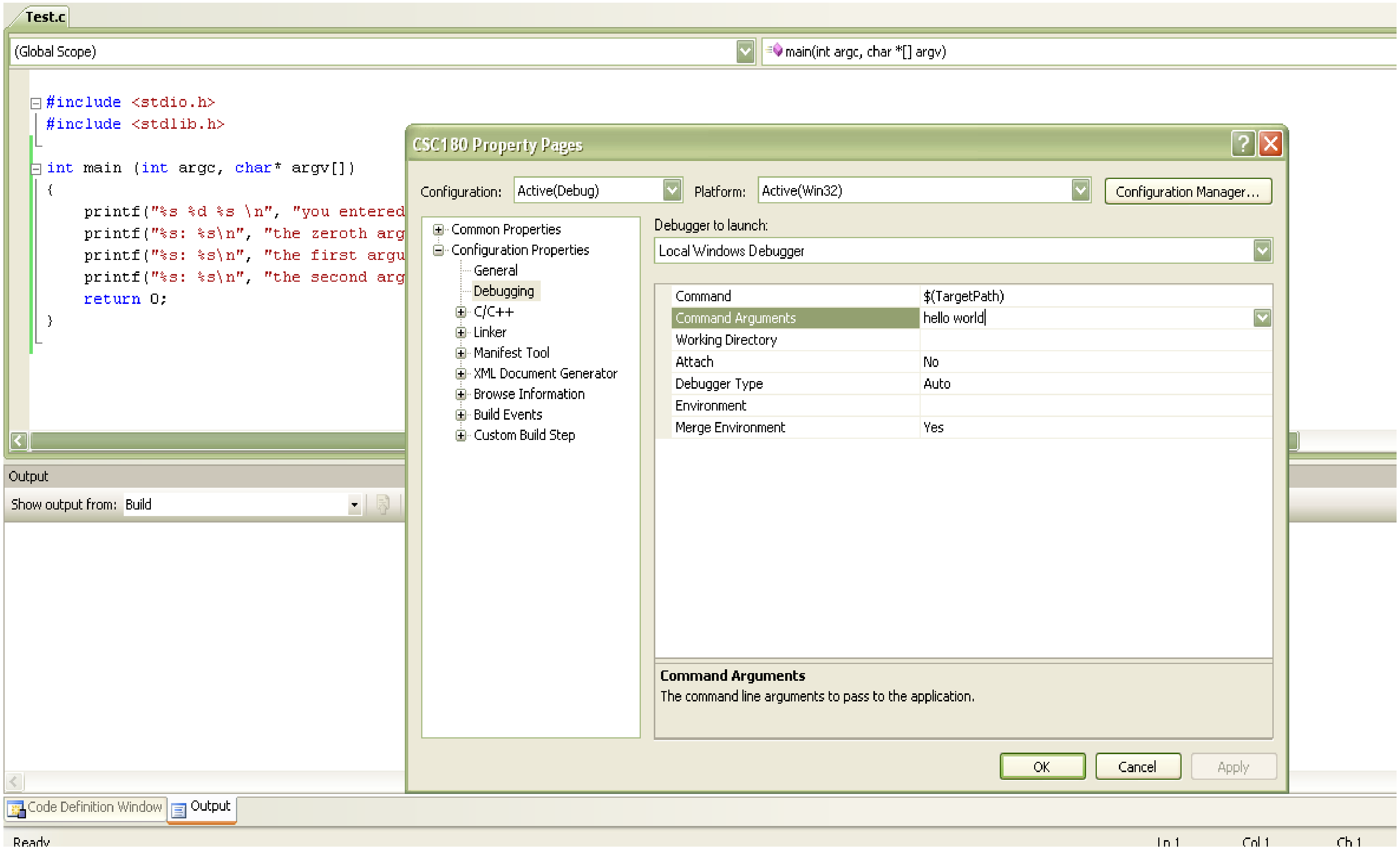
```
> a.out hello world
```

argc/argv example

```
int main (int argc, char* argv[])
{
        printf(“%s %d %s \n”, “you entered”, argc, “arguments”);
        printf(“%s: %s\n”, “the zeroth arg is the program name”, argv[0]);
        printf(“%s: %s\n”, “the first argument is”, argv[1]);
        printf(“%s: %s\n”, “the second argument is, argv[2]);
}
```

```
> gcc argv_example.c –o argv_example
> argv_example hello world
  you entered 3 arguments
  the zeroth argument is the program name: argv_example
  the first argument is hello
  the second argument is world
```

argc/argv example



Test.c

(Global Scope) | main(int argc, char *[] argv)

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv[])
{
    printf("%s %d %s \n", "you entered
    printf("%s: %s\n", "the zeroth arg
    printf("%s: %s\n", "the first argu
    printf("%s: %s\n", "the second arg
    return 0;
}
```

CSC180 Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step

Debugger to launch:
Local Windows Debugger

Command	$(TargetPath)
Command Arguments	hello world
Working Directory	
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes

Command Arguments
The command line arguments to pass to the application.