

CSE / ENGR 142 Programming I

Display Input and Output (I/O)

© 2000 UW CSE

1/10/00

D-1

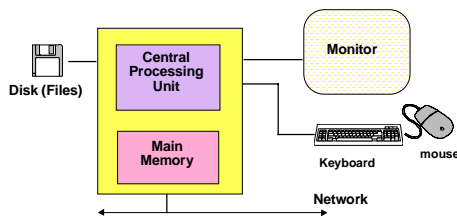
Writing Useful Programs

- It's hard to write useful programs using only variables and assignment statements
- Even our Fahrenheit to Celsius program needed more:
 - Needed a way to get data into and out of the program
- We'll learn more about doing this today
 - Lots of terminology and messy details, but worthwhile.

1/10/00

D-2

What's a Computer?



1/10/00

D-3

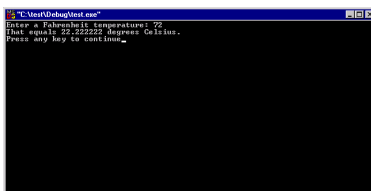
Basic definitions

- **Input:** movement of data **into memory** from outside world (e.g., from keyboard).
 - Changes the value of a variable
 - “*read*” operation
- **Output:** movement of data **from memory** to outside world (e.g., to monitor).
 - “*write*” operation
 - Does not change value of memory

1/10/00

D-4

ASCII Output



1/10/00

D-5

Examples of I/O Statements

```
printf("Enter a Fahrenheit temperature: ");  
scanf("%lf", &fahrenheit);  
celsius = (fahrenheit - 32.0) * 5.0 / 9.0;  
printf("That equals %f degrees Celsius.",  
       celsius);
```

1/10/00

D-6

Display Input and Output

The functions `printf` and `scanf` provide basic display I/O services.

```
printf("control string", list of expressions);  
scanf("control string", list of &variables);
```

Control string gives the format of output or input.

Expressions are what to output.

Variables are where to store the input.

'&' is magic (that is REQUIRED for scanf!)

1/10/00

D-7

`printf()`: display output

```
int numPushups;
```

```
numPushups = 5;  
printf("Hello. Do %d pushups.\n", numPushups);
```

output: Hello. Do 5 pushups.

`%d` is a **placeholder** ("conversion character") for an *int* value.

`\n` is an **escape sequence** for "newline" character.

1/10/00

D-8

What Does the 'n' Do?

```
int numPushups;
```

```
numPushups = 5;  
printf("Hello.");  
printf(" Do %d pushups.\n", number);  
printf("Do them now.\n");
```

output: Hello. Do 5 pushups.
Do them now.

1/10/00

D-9

Multiple Output Expressions

Basic rule:

% placeholders in format string match expressions in output list in *number*, *order*, and *type*.

```
int multiplier;
```

```
double pi;
```

```
pi = 3.14;
```

```
multiplier = 2;
```

```
printf("%d times %f is %f.\n",  
multiplier, pi, (double) multiplier * pi);
```

Output: 2 times 3.14000 is 6.28000.

1/10/00

D-10

Formatting Output

- A few of many things you can do:
 - Control number of decimals
 - 3.1 vs 3.100000
 - Exponential (scientific) or decimal notation
 - 3.1 vs 3.1E0
 - Control total width (including spaces)
 - ____3.1 vs _3.1

How? Look in textbook or a reference manual, or online help!

1/10/00

D-11

Output Format Examples

```
%10.2f ____ 1 2 3 . 5 5 double
```

```
%10.4f __ 1 2 3 . 5 5 0 0
```

```
%.2f 1 2 3 . 5 5
```

```
%10d ____ 4 7 5 int
```

```
%-10d 4 7 5 _____
```

```
%10c _____ a char
```

1/10/00

D-12

scanf(): read input

```
scanf ( "control string", &input list );
```

```
int numPushups ;
```

```
printf ( "Hello. Do how many pushups? " );  
scanf ( "%d", &numPushups );  
printf ( "Do %d pushups.\n", numPushups );
```

output: Hello. Do how many pushups? 5
Do 5 pushups.

input list variables **MUST** be preceded by an **&**.
input list variables **MUST** be preceded by an **&**.

D-13

If You Forget the '&'

The program will compile, but when you execute...



1/10/00

D-14

Whitespace

- space (' '), tab ('\t'), newline ('\n') are "whitespace"
- Skipped by scanf for int ("%d"), and double ("%lf")
 - user can type spaces before a number and they are ignored
- Not skipped for char input "%c"
 - each character typed, including spaces, is used

1/10/00

D-15

Multiple Inputs

•Basic rule:

- % placeholders in the format must match variables in the input list
- **MUST!** match one-for-one in **number**, **order**, and **type**.

```
int studentID ;  
double grade ;  
scanf ("%d %lf", &studentID, &grade );
```

1/10/00

D-16

Format Items Summary

Type	scanf()	printf()	
char	%c	%c	
int	%d	%d	%i also works
double	%lf	%f	← (long) float

What happens if types don't match?

printf -- garbled output
scanf -- unpredictable errors
and don't forget the **&!**

1/10/00

D-17

I/O Programming Considerations

Suppose your program has

```
scanf("%lf", &fahrenheit);
```

and the user types "comfortable" as the input?

--> Nothing is read (so fahrenheit is uninitialized)

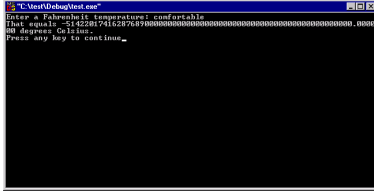
--> Ka-boom

This is YOUR problem (not the user's)

1/10/00

D-18

Here's What Happens



```
TC:\test\Debug\test.exe
Enter a Fahrenheit temperature: comfortable
That equals 21.42278741627070 degrees Celsius
Enter any key to continue: _
```

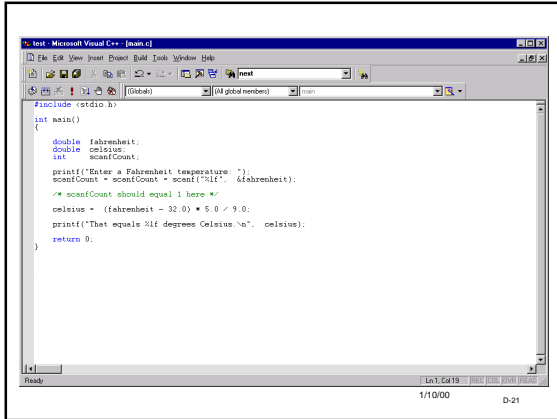
1/10/00 D-19

What You Can Do

scanf() "returns" the number of items read successfully

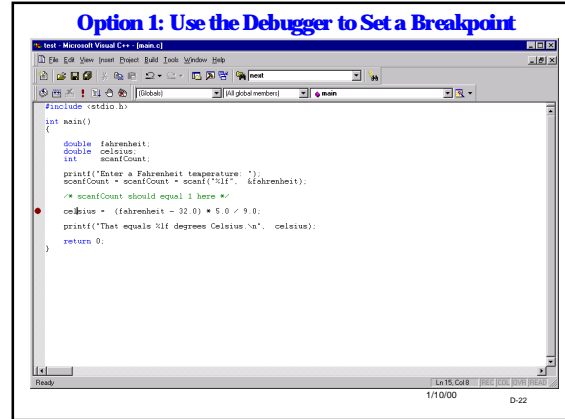
```
int scanfCount;
scanfCount = scanf("%d", &studentID);
/* if scanfCount is not equal to 1 at this point,
the user has made some kind of mistake.
Handle it. */
```

1/10/00 D-20



```
#include <stdio.h>
int main()
{
    double fahrenheit;
    double celsius;
    int scanfCount;
    printf("Enter a Fahrenheit temperature: ");
    scanfCount = scanfCount + scanf("%lf", &fahrenheit);
    /* scanfCount should equal 1 here */
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %lf degrees Celsius\n", celsius);
    return 0;
}
```

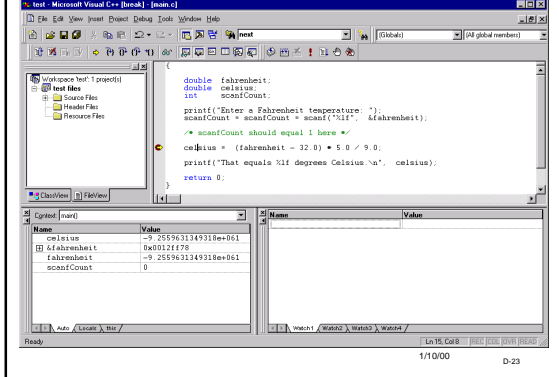
1/10/00 D-21



```
#include <stdio.h>
int main()
{
    double fahrenheit;
    double celsius;
    int scanfCount;
    printf("Enter a Fahrenheit temperature: ");
    scanfCount = scanfCount + scanf("%lf", &fahrenheit);
    /* scanfCount should equal 1 here */
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %lf degrees Celsius\n", celsius);
    return 0;
}
```

1/10/00 D-22

Execution Pauses at the Breakpoint

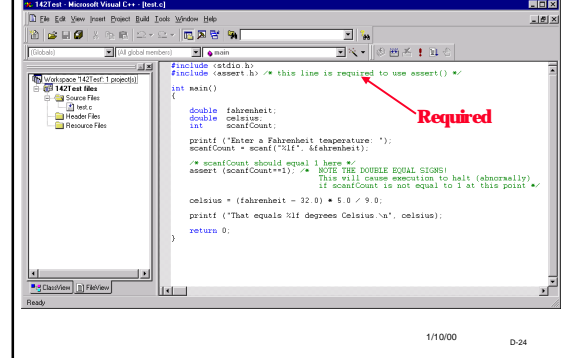


```
#include <stdio.h>
int main()
{
    double fahrenheit;
    double celsius;
    int scanfCount;
    printf("Enter a Fahrenheit temperature: ");
    scanfCount = scanfCount + scanf("%lf", &fahrenheit);
    /* scanfCount should equal 1 here */
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %lf degrees Celsius\n", celsius);
    return 0;
}
```

Name	Value
celsius	2559631349318e+061
fahrenheit	0x00124f78
scanfCount	0

1/10/00 D-23

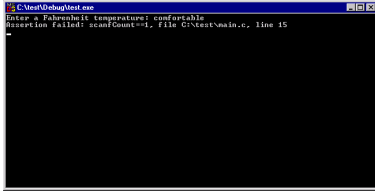
Option 2: Use assert()



```
#include <stdio.h>
#include <assert.h> /* this line is required to use assert() */
int main()
{
    double fahrenheit;
    double celsius;
    int scanfCount;
    printf("Enter a Fahrenheit temperature: ");
    scanfCount = scanfCount + scanf("%lf", &fahrenheit);
    /* scanfCount should equal 1 here */
    assert(scanfCount==1); /* NOTE THE DOUBLE EQUAL SIGNS!
This will cause execution to halt (abnormally)
if scanfCount is not equal to 1 at this point */
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %lf degrees Celsius\n", celsius);
    return 0;
}
```

1/10/00 D-24

What Happens



1/10/00

D-25

More Terminology: Syntax vs Semantics/Logic

- **Syntax**: the required form of the program
 - punctuation, keywords, word order, etc.
 - The C compiler always catches these “syntax errors” or “compiler errors”
- **Semantics and logic**: what the program means
 - what you want it to do
 - **The C compiler cannot catch these kinds of errors!**
 - They can be extremely difficult to find
 - Logic errors may not show up right away

1/10/00

D-26

Syntax or logic errors?

```
#include <stdio.h>
int main (void) {
double far, cel;

far = 56.0;
cel = (far-32.0)*5.0/9.0

printf ('Celsius is %f ', cell);

retrun (0);
}
```

```
#include <stdio.h>
int main (void) {
double far, cel;

far = 56.0;
cel = far-32.0*5.0/9.0;

printf ("Celcius is %d", far);

return (0);
}
```

1/10/00

D-27

Syntax or logic errors?

```
#include <stdio.h>
int main (void) {
double far, cel;

far = 56.0;
cel = (far-32.0)*5.0/9.0;

printf ('Celsius is %f ', cell);

retrun (0);
}
```

```
#include <stdio.h>
int main (void) {
double far, cel;

far = 56.0;
cel = far-32.0*5.0/9.0;

printf ("Celcius is %d", far);

return (0);
}
```

1/10/00

D-28

I/O Summary

- Output: `printf("control string", output list);`
 - output list – expressions; values to be printed
 - control string – types and desired format
 - for now, **NO "&", ever!**
- Input: `scanf("control string", &input list);`
 - input list – variables; values to be read
 - control string – types and expected format
 - can be a way of initializing variables
 - for now, **YES "&", always!**
 - **Check that you actually read some input!**
- Both: `%x`'s, I/O list match in number, order, type

1/10/00

D-29

More on Initializing variables

- Review: **Initialization** means giving something a value for the **first** time.
- Potential ways to initialize:
 - Assignment statement
 - `scanf`
 - Yet another way: initializer with declaration

1/10/00

D-30

Initializing when Declaring

```
int product, i;  
/*declarations without  
initializers */
```

```
product = 40;  
i = 5;  
/*initialization via  
assignment statements  
*/
```

Initializers are part of the declaration;
they are not assignment statements (despite the
= sign).

```
int product = 40, i =5;  
/*declaration with  
initializers, */
```

```
i = 6;  
/*not an  
initialization! */
```

1/10/00

D-31

Initialization Quiz

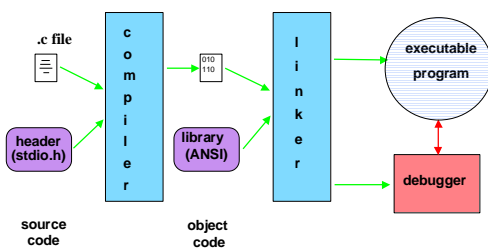
```
int main (void){ /*line 1*/  
int a, b, c, d=10; /*line 2*/  
b=5; /*line 3*/  
d=6; /*line 4*/  
scanf("%d %d", &b, &c); /*line 5*/  
}
```

Q: Where is each of a, b, c, and d
initialized?

1/10/00

D-32

Compilers, Linkers, etc.



1/10/00

D-33