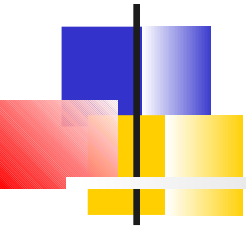
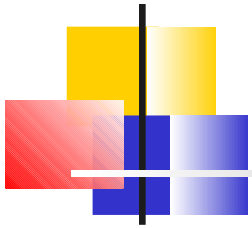


Basic Types

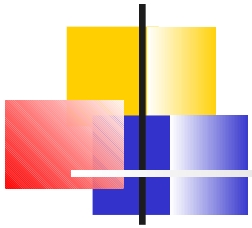
Chapter 7



Objectives

You will be able to:

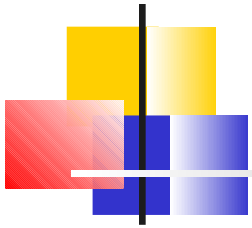
- Use integer and floating point types appropriately for variables and literals.
- Write correct statements for input and output of the different numeric types.
- Avoid common problems resulting from automatic type conversions.



About Chapter 7

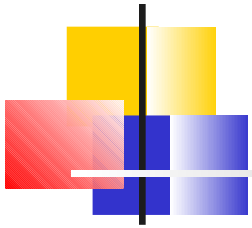
- A lot of complexity
  - Read the chapter, but don’t try to memorize
  - Should be *aware* of types available in C
  - Usually don’t need to use most of them.

- This presentation will cover what you really need to know for most real world programming.



Kinds of Numbers

- Computers provide mathematical operations for two kinds of numbers:
  - Integer
  - Floating Point
- You can’t tell one kind of number from the other by looking at it in memory.
- If you were writing machine language code, you would have to remember which kind of number you stored in a given memory location and use the right kind of operation.

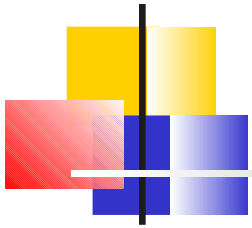


Kinds of Numbers

- In the C Language you *declare* variables as one form or the other.

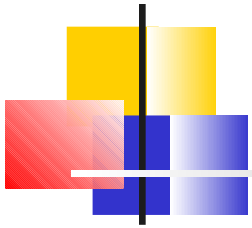
```
int i;
double radius;
```

- The compiler generates machine code for the right kind of operation based on your declarations.



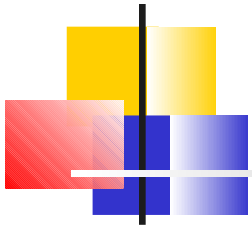
Which Kind of Number to Use?

- Determined by the use.
- Counting
  - Use integer
- Measurement
  - Use floating point
- Scientific Calculations
  - Use floating point
- What about accounting?
  - Dollars and cents



Numbers in C

- Integer
  - Exact representation
  - Signed or unsigned
  - Various sizes (implementation dependent)
    - Typically 32, 16, 8 bits
    - 64 bits on very large systems



Numbers in C

- Floating-point
  - Represent mathematical “real” numbers
    - or “rational” numbers
  - Always signed
  - Two sizes, or maybe three
    - 32 bits, 64 bits, maybe more
  - Approximation in most cases

- ANSI/IEEE Standard 754-1985
- For a concise summary see:
  http://www.psc.edu/general/software/packages/ieee/ieee.html

Names for Integer Types in C

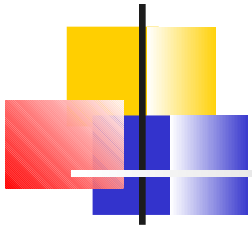
Common Name	Full Name	Other Acceptable Names	
int	signed int	signed	
long	signed long int	long int	signed long
short	signed short int	short int	signed short
unsigned	unsigned int		
unsigned long	unsigned long int		
unsigned short	unsigned short int		

C99 adds even more.

Remember int.

Look up the others if you need them.

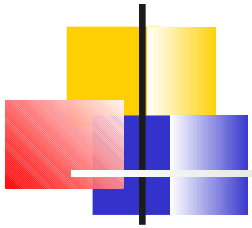
Rarely need any integer type other than int.



Unsigned Integer Types

- There are a few appropriate uses
  - Don't use unsigned integers unless you have a good reason to do so.
  - Will discuss as we reach them.

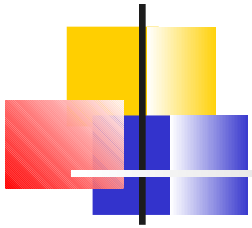
- *Never use unsigned integers in calculations.*



Size and Range of Integer Types

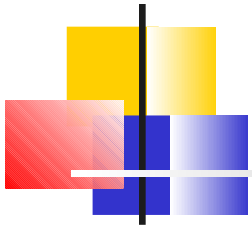
- Size of the various types is implementation dependent.
  - Complier writers make the choice, considering hardware characteristics of the target system.

- Symbolic constants defined in limits.h
  - INT_MIN INT_MAX
    - Values are specific to the system on which the file resides.



Size and Range of Integer Types

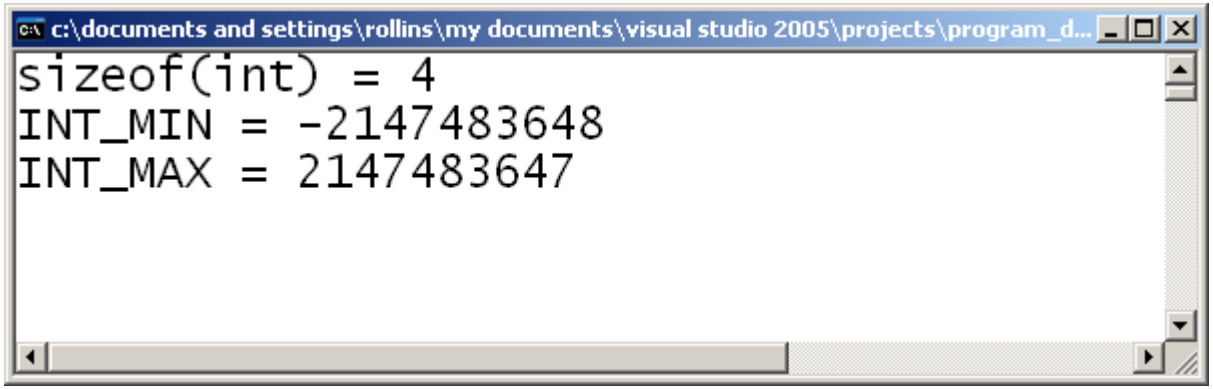
- sizeof() tells you the size of variables and types, in bytes.
  - sizeof() returns an unsigned integer value.
- Book says to typecast as unsigned long and print using **`%lu`**
  - On systems that I use printing with %d works OK.



Program size.c

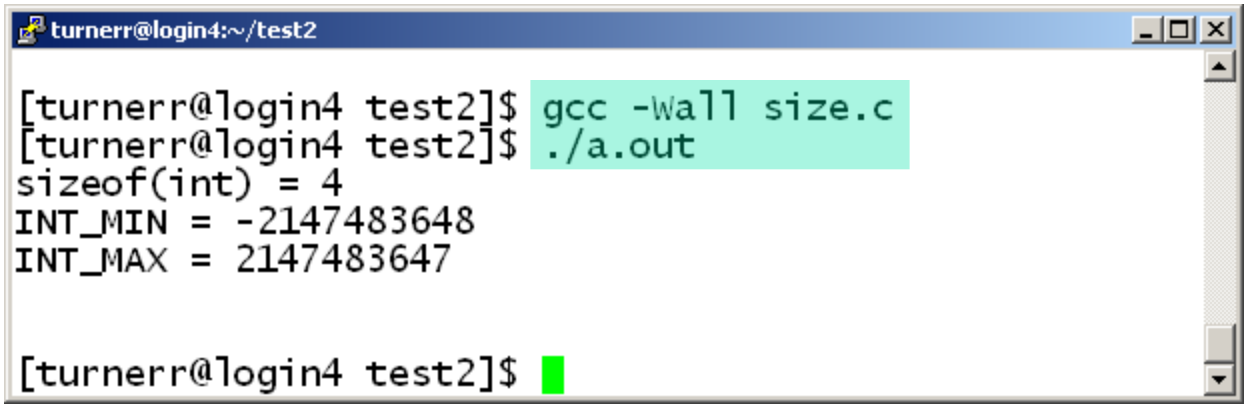
```
#include <stdio.h>
#include <limits.h>

int main()
{
    printf ("sizeof(int) = %lu\n", sizeof(int));
    printf ("INT_MIN = %d\n", INT_MIN);
    printf ("INT_MAX = %d\n\n", INT_MAX);
    getchar();
    return 0;
}
```

Size of integers on Windows XP



An Intel Pentium 4 desktop computer.

Size of Integers on Circe



Same results as for Windows PC

Range for int

INT_MAX = 2,147,483,647 = $2^{31} - 1$

or, in binary

0111 1111 1111 1111 1111 1111 1111 1111

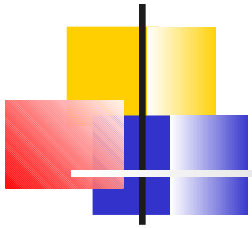
31 1 bits

INT_MIN = -2,147,483,648 = -2^{31}

or, in binary

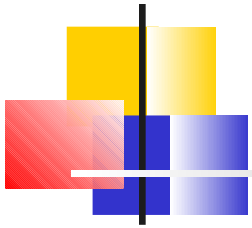
1000 0000 0000 0000 0000 0000 0000 0000

31 0 bits



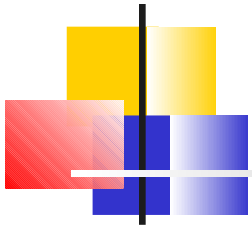
Range for int

- You don’t need to memorize these numbers.
- Should be able to compute them if necessary.
- Should know the bit patterns.
- Should know order of magnitude.
  - INT_MAX is about 2 billion



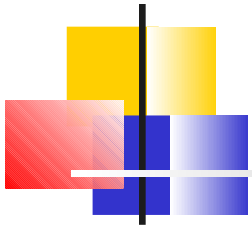
How big is INT_MAX?

- About 2 billion
  - 2 giga-things
- Big enough to hold
  - Population of US (~300 million)
  - Number bytes of RAM on my PC (~1 billion)
  - Loop counter for “while” or “for” loop



How big is INT_MAX?

- About 2 billion
  - 2 giga-things
- Not big enough to hold
  - Population of world (~6.4 billion)
  - Number bytes of disk on my PC (80 billion)
  - US Budget in dollars (2.3 trillion)
  - Avogadro's number ($6.02214199 \times 10^{23}$)

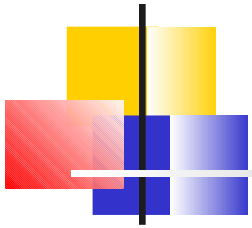


How big is INT_MAX?

- Big enough for most things that you will ever need to keep track of individually

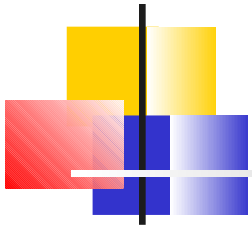
  But not necessarily everything.

- You have to think about how large numbers can get in your programs.



What if 32 bits are not enough?

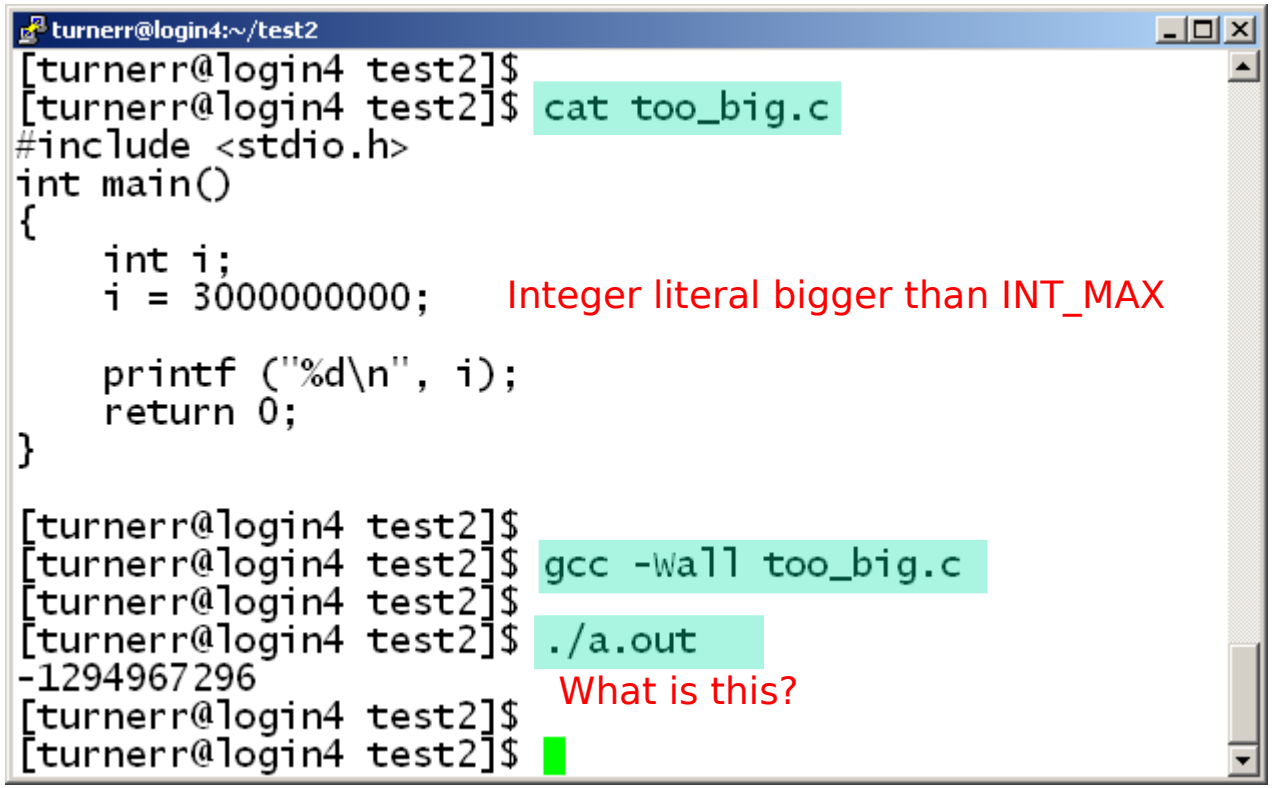
- Floating point
  - double supports integer values up to 15 digits
- Scaled arithmetic
  - Keep track of scaling factor separately
- Extended precision software packages are available.
- Some languages have extended types implemented in software.
  - “Decimal” in .NET Framework (VB, C#)



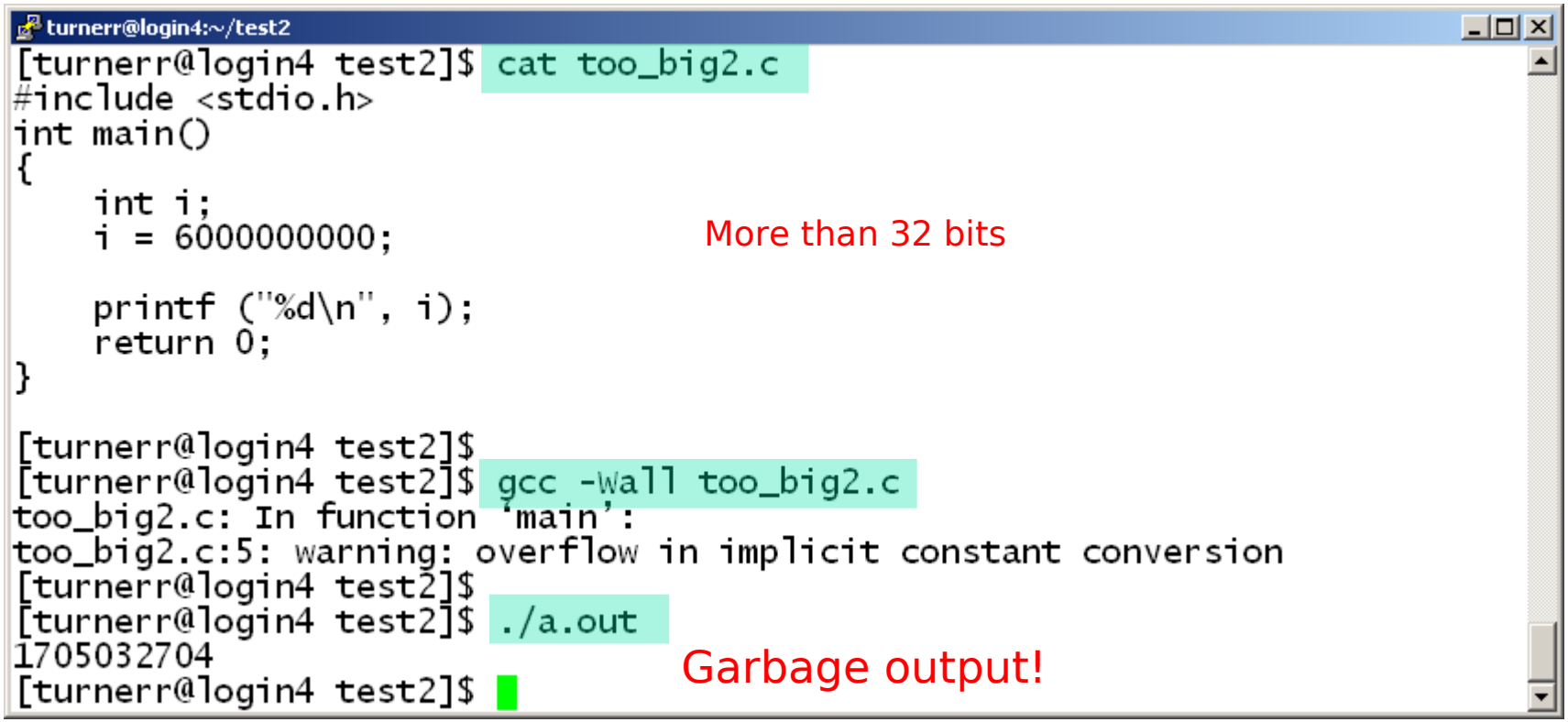
Integer Literals

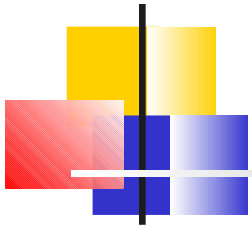
- A number without sign or decimal point written out “literally” in a program.
  - Sign is consider a unary operator
    - Not part of the number
  - Decimal point makes it a floating point literal
    - A double by default.
  - Commas are not permitted

A Literal Too Big



An Invalid Literal





Integer Literals

- Be aware of size.
  - 32 bits is OK for up to 2 billion.

- Don’t ignore compiler warnings.
- Proceed if you completely understand
  the warning and it fits what you intended

Floating-Point Types in C

What to remember:

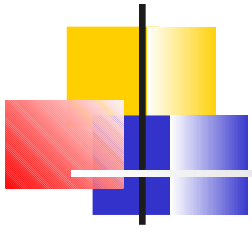
These are the minimum value ranges for the IEEE floating-point types. The names given in this table are the ones defined by the C standard.

Type Name	Digits of Precision	Name of C Constant	Minimum Value Range Required by IEEE Standard
`float`	6	`±FLT_MIN...±FLT_MAX`	±1.175E−38 . . . ±3.402E+38
`double`	15	`±DBL_MIN...±DBL_MAX`	±2.225E−308 . . . ±1.797E+308

Figure 7.4. IEEE **floating-point types.**

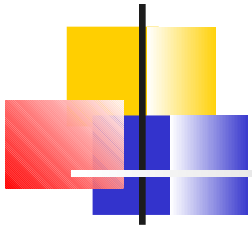
Fischer, page 232

For most practical purposes, double is both necessary and sufficient.



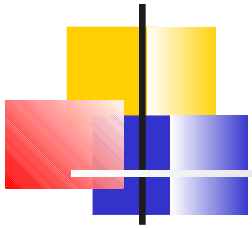
Floating-Point Types in C

- No physical measurements are precise to more than 15 digits.
- No numbers used in normal mathematical or scientific calculations are larger than 10^{308}
  - or smaller than 10^{-308}



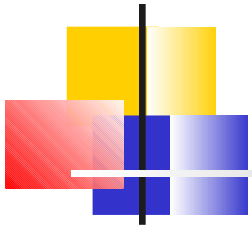
Floating-Point Types in C

- Number of atoms in the visible universe:
  - $4 * 10^{79}$
  - http://en.wikipedia.org/wiki/Observable_universe
- Age of the universe
  - $13 * 10^{9}$ years = $4 * 10^{23}$ microseconds
  - http://www.astro.ucla.edu/~wright/age.html
- Mass of an electron in kilograms
  - 9.10939×10^{-31}
  - http://hypertextbook.com/facts/2000/DannyDonohue.shtml



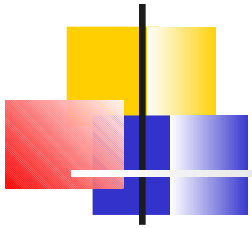
Floating-Point Types in C

- Numbers representing physical quantities in the real world do not tax the capacity of a double.
- Intermediate results in some calculations might.
  - Separate subject: numerical analysis



Floating Point Literals

- Two ways to write a floating point literal
  - 3.14159
    - Decimal point tells compiler to use floating point
    - Default type is double
  - 1.05792e+05
    - Like scientific notation
    - Means $1.05792 * 10^5$
    - Can use “e” or “E”
    - “+” may be omitted
    - Leading “0” may be omitted



Reading and Writing Numbers

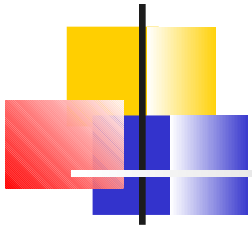
Format specifiers for scanf and printf

- For int
  - Use **`%d`** for scanf and printf

Memorize this

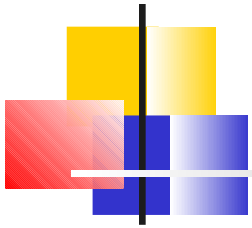
- For long int
  - Use **`%ld`**
- For short int
  - Use **`%hd`**

Look these up if you ever need them.



Format Specifiers for int

- %d and %i are interchangeable for printf
- Subtle differences for scanf
  - %i accepts hexadecimal input as well as decimal.
    - Example: 0x789a
  - %i interprets any integer with a leading zero as octal!
  - %d accepts only decimal input
- Forget %i



Floating Point Input

- You have to tell scanf the size of input variable:

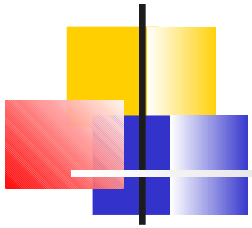
For double **"%lf"**

Memorize this

For float "**%g**" or "**%f**" or "**%e**"

For long double "**%Lf**"

Look these up if you ever need them.



Floating Point Output

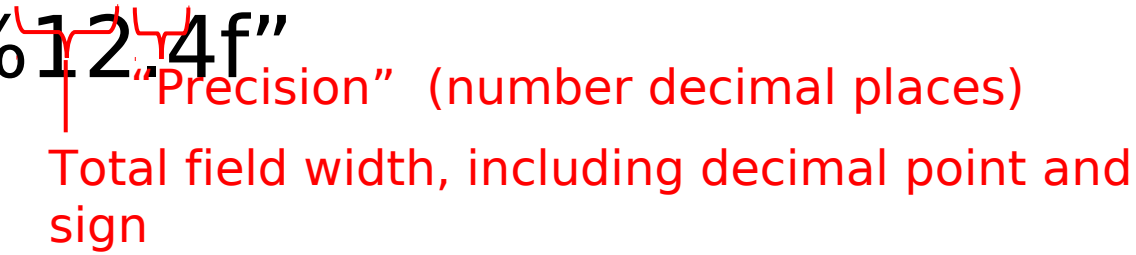
- Same format specifier works for both float and double printf
  - Unlike scanf!
  - All float values are converted to double when they are passed to printf.

- `%f` Use normal decimal notation 123.456
- `%e` Use exponential format 1.23456e+02
- `%g` System chooses format

Floating Point Output

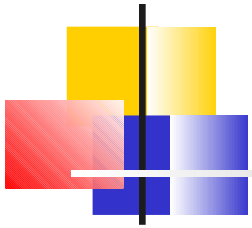
- Floating point formats can specify both field width and number of decimal places.

- “%12.4f”



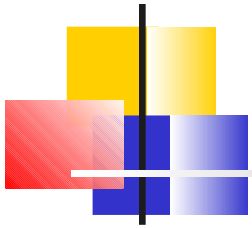
More space will be used if necessary to show integer part

Floating point numbers are rounded to last position printed.



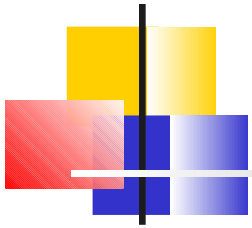
Mixing Types in Computations

- C allows you to mix types in a computation.
  - Sometimes safe
  - Generally asking for trouble.
  - Try to avoid!
- Basic Type Conversions
  - Length
    - Convert between integers of different length
    - Convert between floating point types of different length
  - Representation
    - Convert between integer and floating point



Length Conversions

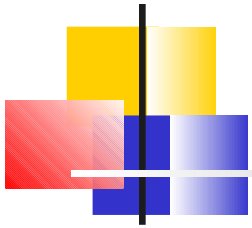
- Converting to a longer version is safe.
  - Cannot lose information
  - short or char to int
    - Always OK
  - float to double
    - Can be misleading
    - Number *appears* to have more precision than it actually has.



Unsafe Conversions

- Value converted to a *shorter* type
  - May lose significant bits.
  - Result can be garbage.

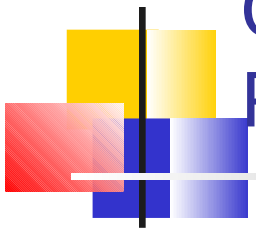
- Integer
  - Only the *least significant* bits are transferred
  - Lose high order bits from large values
  - Lose sign bit from negative values
  - No warning at compile time or run time.



Unsafe Floating Point Conversions

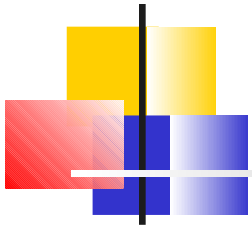
- double to float
  - Lose low order bits from mantissa
    - Results in loss of precision
  - May lose bits from exponent
    - Results in complete garbage

- Don’t do these conversions
- Won’t happen if you only use int and double.



Converting Between Integer and Floating Point

- Integer Value Stored as Floating Point
  - Generally Safe
  - Always safe to store int as double
- Floating Point Value Stored as Integer
  - Fractional part is *dropped*, not rounded
  - 1.9999999 becomes 1 as integer
  - May lose significant bits of the integer part



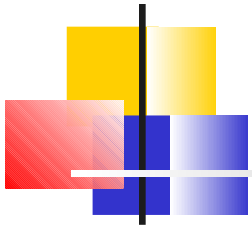
Type Casts

- You can tell the compiler to do a type conversion.

```
double x, y;
int k, m;
...
k = (int) y;
x = (double) k;
```

Typecasts

Typecast has no effect here, because the conversion would be done automaticlly.



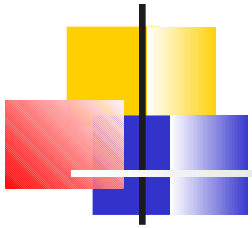
Automatic Type Coercion

- The C compiler does type conversion automatically when necessary.

```
double x = 0.0;
...
x = x + 1;
```

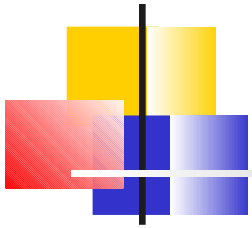
- Can’t add an int to a double.
- Compiler treats this as

```
x = x + (double) 1;
```



Necessary Typecasts

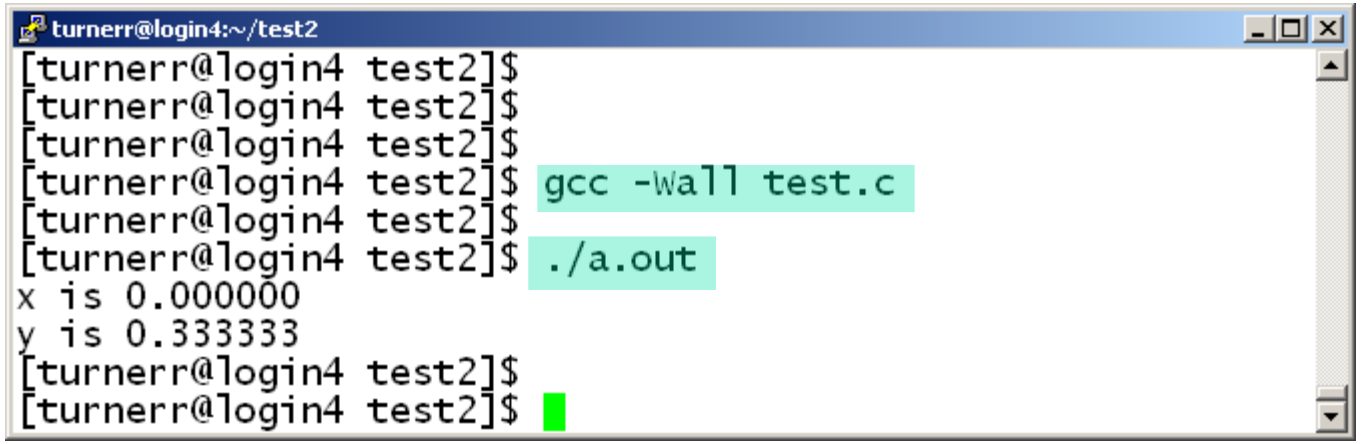
- Typecasts are necessary in order to tell the compiler to do a type conversion for a variable in an expression.



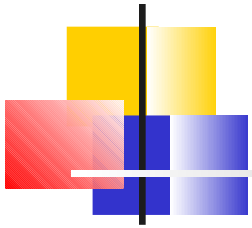
Necessary Typecasts

```
#include <stdio.h>

int main()
{
    int i = 1;
    int j = 3;

    double x = 0.0;
    double y = 0.0;

    x = i/j;      // Result is converted to double automatically
    printf ("x is %f\n", x);

    y = (double)i / j;
    printf ("y is %f\n", y);

    return 0;
}
```

Program Running

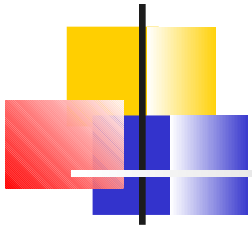


End of Section



Summary

- There are multiple ways to represent numbers in a C program
  - Different amounts of memory
  - Different representation
    - Integer vs. Floating Point
- C converts between representations when necessary
  - May not be what you meant!
- You can specify conversion explicitly with typecasts.



Summary

- Type conversions are a snake pit!
- Stick to int and double.
- Convert between integer and floating point only when necessary.
  - Use explicit typecasts for intermediate results.
  - Be very careful.

End of Presentation