GNG 1106

Fundamentals of Engineering Computation

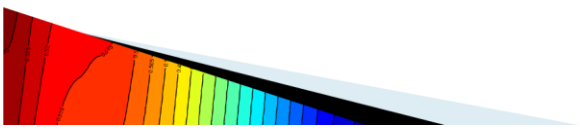
Module 2 - More on Fundamental Concepts

Fall 2016

Topic 1: Data Types and Simple Variables

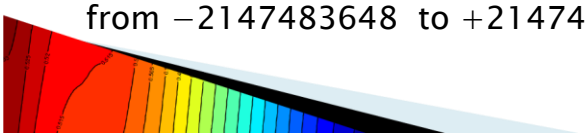
Data Types in C

- The basic "data types" in C are literals, variables and symbolic constants.
- Literals express values and are used in expressions.
- The content of a variable can be changed by a program but a symbolic constant cannot be changed.
  - Symbolic constants also have names, as variables, but do not occupy memory, they are treated more like literals.
- All variables and symbolic constants must be declared and defined before usage in a C program.



Literal values

- In all programming languages, numbers and values are represented as literal values.
- Literal values have types as do variables. The following shows how literal values can be represented in C.
  - Integer values: `1, 2, -301`, etc.
  - Real values: `1.0, 2.0, -310.345, 3.124e-21`, etc.
  - Characters: `'a', 'b', 'Z', '0', '$'`, etc.
  - Character strings: `"This is a string of characters"`
- The range of values that can be represented vary according to compilers used.
  - For example, CodeBlocks uses 4 bytes to represent an integer, which means that integer values can range from −2147483648 to +2147483647



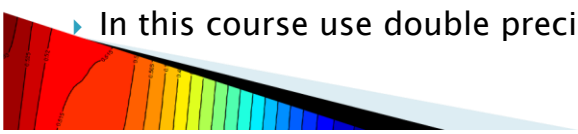
Integer Literal Values

- By default, C treats integer literal values as signed values (with negative values)
  - Occupies 2 or 4 bytes according to the compiler used (CodeBlocks uses 4 bytes).
  - With 4 bytes an integer values can range from −2147483648 to +2147483647 or as unsigned values from 0 to 4294967295
- C programs cannot treat integer values outside its valid range.
- In this course we shall work with signed integer values (default).

5

Real Literal Values

- The real literal value in C contains three parts: xx.yyezz
  - The decimal part (or integer), xx, to the left of the decimal point (mandatory including the decimal point).
  - The fraction part, yy, to the right of the decimal point,
  - The exponential part, **e**zz, that follows the fractional part. The exponential part multiplies xx.yy by 10^{zz}
- Real values in C are represented in either single precision or double precision (IEEE 764)
  - Single precision (type `float`) occupies 4 bytes
  - Double precision (type `double`) occupies 8 bytes
  - Literal values in CodeBlocks are represented using double precision.
- In this course use double precision.

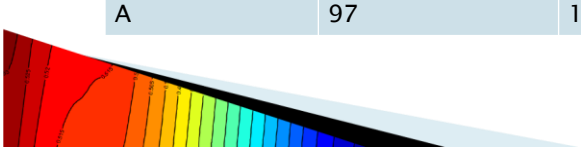


6

Character Literal Values

- As with all data, a character is represented using a binary code
  - The ASCII code (American Standard Code for Information Exchange).
  - Some examples of ASCII code:

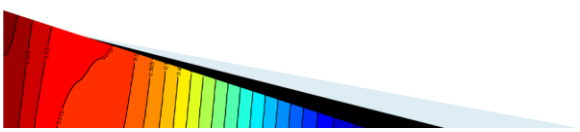
Character	Integer Equivalent	Binary Code
LF (new line)	10	0001010
SP (space)	32	0100000
0	48	0110000
A	65	1000001
A	97	1100001



7

Character Literal Values

- In C a character literal value normally consists of a letter surrounded by single quotes
  - Lower case characters: 'a', 'z'
  - Upper case characters: 'A', 'M'
  - Control characters: '\n' (new line), '\t' (tabulation), '\b' (carriage return)
  - Space: ' '
  - Punctuation: '?', '\"', '!', '?', '\\'
    - Note the use of the \ in the case of the characters ' and \.

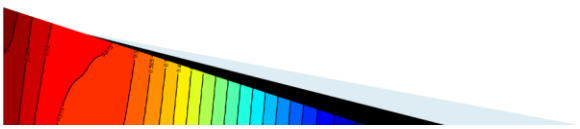


8

Character Strings

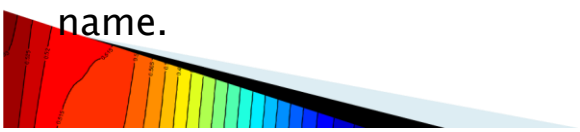
- The character string is not a simple type
  - It is composed of a sequence of characters.
  - We shall study more closely the characters string in Module 4 after introducing arrays.
- Strings play an important roles in programs
  - Recall `printf("Character string\n");`
- String literals consist of characters enclosed within a pair of double quotes.

```
"Simple character string"
"Line 1\n Line 2\n Line 3\n"
"A string with\"double quotes\"\n"
```



The Computer Variable

- The *variable* plays an important role in software, since it is the basic unit that contains data on which programs operate.
- Recall the three characteristics of a variable:
  - Has **a location** in memory that contains its binary value.
  - Has **an address** used to locate the variable in memory.
  - Has a **type** which defines how to interpret the binary value (also defines the number of bytes occupied by the variable).
- Recall the rules and conventions for variable name.

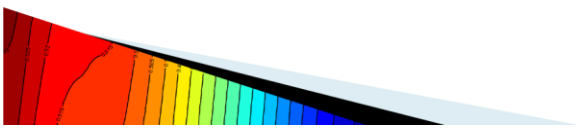


C Variables

- Variables must be declared via declarations.
  - Declarations are usually placed after the first `{` of `main` (or of a function definition) and before the first command.
  - A declaration contains at variable name and its type.

- There are 4 basic data types in C:

`char`	stores one character: `a` to `z`, `A` to `Z`, `!`, `$`,…
`int`	stores an integer: `1`, `101`, `-1462`,…
`float`	stores a real number: `0.5`, `-122.34`,…
`double`	stores a real number in double precision.



Symbolic Constants

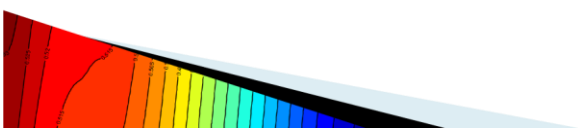
- "Magic numbers" sprinkled throughout a program is generally considered to be an example of bad programming style.

- If a "magic number" must be used in the program, then it is best to associate the number to a symbolic constant that has a descriptive name and then to use the constant in the program.

- Symbolic constants are defined using the pre-processor directive `define` E.g.:

```
#define NBR_OF_CUBES 455
#define PI 3.141593
```

  - Pre-processor directives do not require a `;` at the end.
  - The same rules for variables names are applied to symbolic constants, BUT convention dictates that symbolic constants names are in UPPER CASE as shown above.
  - Use the _ character to make names readable.

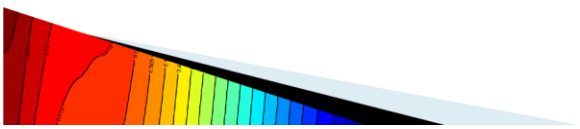


Symbolic Constants

- The pre-processor performs a textual substitution of the symbolic constant name with the associated quantity (directly to the right) everywhere in the code before compilation.
  - ➔ No memory space is used for a symbolic constant. They are not variables.
  - ➔ A program cannot change the value associated with a symbolic constant.
  - ➔ Be careful with the `;`

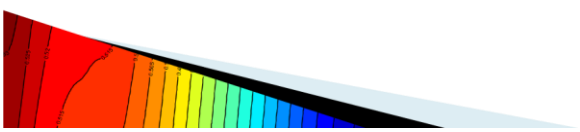
    E.g.: `#define MAX 100;`

    will result in a substitution of `MAX` by `100;` everywhere in the program!



13

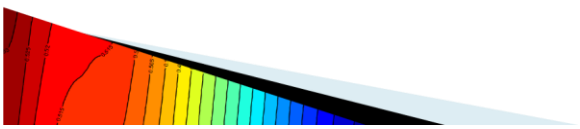
Topic 2: Arithmetic and Logic Expressions



14

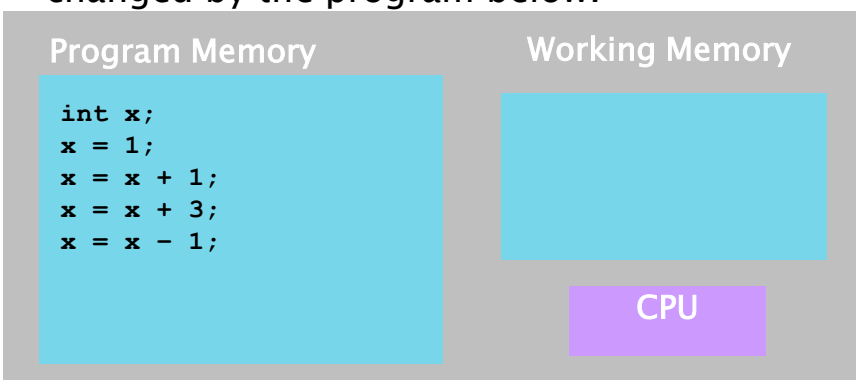
Evaluating arithmetic expressions

- Recall basic arithmetic expressions from last week
  - Binary operators: +, –, /, *
  - CPU applies operators one at a time
  - Assignment operator, =, assigns a value to a variable

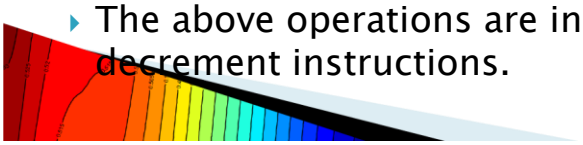


Exercise: Increment/decrement instructions

- Show how the value of the variable x is changed by the program below.

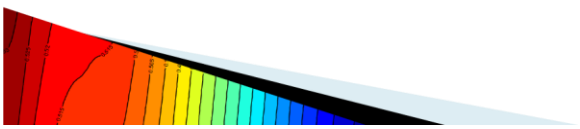


- The above operations are increment and decrement instructions.



Increment/Decrement Operators

- There exists in C, special operators for adding or subtracting 1 from a variable
  - These operators are
    - The increment operator: ++
    - The decrement operator: --.
  - These operators present subtleties that can easily lead to bugs.
  - You may use them, but at your own risk.



Rules of Operator Precedence

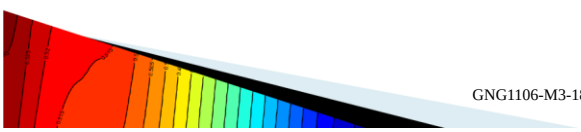
- C evaluates arithmetic expressions according to the following rules of operator precedence.

1 Contents of parentheses are evaluated first.
  - If there are many "levels" of parentheses, then the innermost pair is evaluated first; the next innermost pair is evaluated second…
  - If there are many pairs of parentheses on the same level then they are evaluated left to right.

2 Negation (unary) is evaluated next.

3 Multiplications, divisions and moduli are evaluated next.
  - If there are many then they are evaluated from left to right.

4 Additions and subtractions are evaluated last.
  - If there are many, they are evaluated left to right.



Examples of precedence

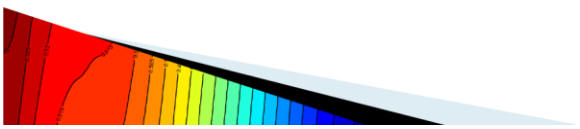
```
e = a * (b * (c + d));
      3     2     1

e = (a + b) * (c + d);
      1     3     2

e = p * r % q + w / x - y;
      1   2   4   3   5
```


Operations with integers

- **Operations with integer values gives integer values.**
  - "3 + 4" gives the integer value "7", and thus the instruction "**c = 3 + 4;**" assigns the value 7 to the variable **c**.
  - "3 * 4" gives the integer value "12", and thus the instruction "**c = 3 * 4;**" assigns the value 12 to the variable **c**.
- **The division of integer values gives only the quotient.**
  - The instruction "**c = 3/4;**" gives the integer value 0 (note that the remainder 3 is lost), i.e. the integer value 0 is assigned to the variable **c**.
  - L The instruction "**c = 4/3;**" gives the integer value 1 (note the remainder 1 is lost), i.e. the integer value 1 is assigned to the variable **c**.
- **The operation that gives the remainder of an integer division is the modulo operation; in C the modulo operator is %.**
  - The instruction "c = 3%4;" gives the integer value 3, i.e. the integer value 3 is assigned to the variable c.
  - The instruction "c = 4%3;" gives the integer value 1, i.e. the integer value 1 is assigned to the variable c.

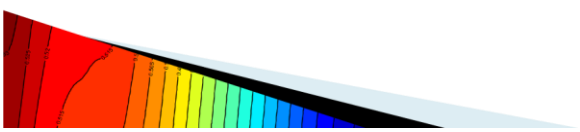
Operations with real values

- **Operations with real values gives real values.**
  - "3.1 + 4.2" gives the real value "7.3", thus the instruction "c = 3.1 + 4.2;" assigns the value 7.3 to the variable c.
  - "3.1 * 4.2" gives the real value "13.02", thus the instruction "c = 3.1 * 4.2;" assigns the value 13.02 to the variable c.
  - "3.1 / 4.2" gives the real value "0.738095238095238140000", thus the instruction "c = 3.1 * 4.2;" assigns the value "0.738095238095238140000" to the variable c.
    - Note that using the Windows calculator, the result of the division is "0.73809523809523809523809523809524".
    - This means that the precision of the C program is less than that of the calculator. The value shown above was displayed using the instruction "`printf("3.2 / 4.2 = %0.15f\n",3.1/4.2);`".
    - Representation of real values are not exact in software. This can lead to errors in calculations and even logical errors.



Operations with values of mixed types.

- In C, arithmetic operators applied to mixed types (i.e. with integers and real values) use an implicit rule of promotion. The rule of promotion is applied with binary operators that operate on 2 values of different type.
- When an operator is applied to one real value and one integer value, the integer value is converted (i.e. promoted) to a real value before applying the operator.
- In the case of the operation "3 + 4.1", the integer value is first converted to "3.0" and then added to "4.2" to give the value "7.1".
- In the case of the operation "3/4.1", the integer value is first converted to "3.0" and then divided by "4.1" to give the value "0.73170731707317083000"



Division of integers

- Suppose that `a`, `b` and `c` are of type `int`.

E.g.:
```
a = 3;
b = 4;
c = a / b;   → c  0
c = b / a;   → c  1
```

  - The division of integers returns the quotient.

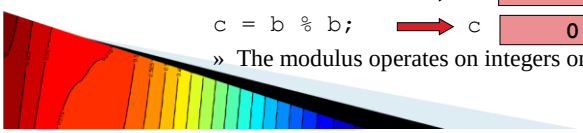
The Modulus

- Operation that computes the remainder of a division of integers. Suppose that `a`, `b` and `c` are of type `int`.

E.g.:
```
a = 3;
b = 4;
c = a % b;     → c  3
c = -a % -b;   → c  -3
c = b % a;     → c  1
c = b % b;     → c  0
```

Sign of remainder is sign of numerator.

  - The modulus operates on integers only.



Mixed Type Arithmetic

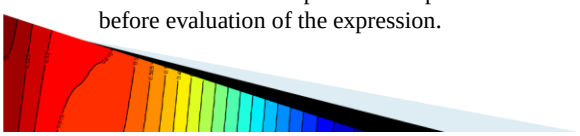
- An expression that contains variables of many types will be converted in memory by the C compiler according to the implicit rule of promotion.

- The promotion hierarchy of our 4 basic data types is as follows:

```
double   ← highest type
float
int
char     ← lowest type
```

  - The hierarchy is organized according to the amount of information that can be stored in the data type.
    - A double variable can contain the same amount of information as all of the "lower" data types.

- The implicit rule of promotion followed by the C compiler when evaluating mixed type expressions is:
  - All variables in an expression are promoted to the highest type (in temporary memory) before evaluation of the expression.



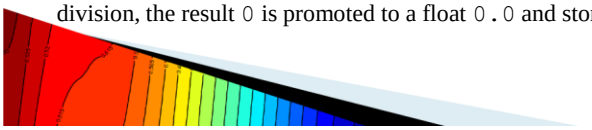
- Consider the following examples of mixed type expressions.

E.g.:

```
int a, b, c;
float x, y, z;
a = 3;
b = 4;
x = 1.0;
y = 2.5;
```

No conversions	`c = a + b;`	→ c	7	
No conversions	`z = x + y;`	→ z	3.5	
1 conv. from `int` to `float`	`z = a + b;`	→ z	7.0	
1 conv. from `int` to `float`	`z = x + a;`	→ z	4.0	
1 conv. from `float` to `int`	`c = x + y;`	→ c	3	Loss of information!
2 conversions	`c = x + a;`	→ c	4	
1 conv. from `int` to `float`	`z = a / b;`	→ z	0.0	

- Care must be taken when storing a real number into an `int`. The real number will be truncated and only the integer portion will be stored.
- Careful with the division of integers; in the last example, `a / b` is evaluated as an integer division, the result `0` is promoted to a float `0.0` and stored in `z`.



The Cast Operator

- The unary `cast` operators can be used to force the type conversion of a variable.

  E.g.: (same variable definitions as in the previous example)

  `z = (float) a / b;` → z 0.75

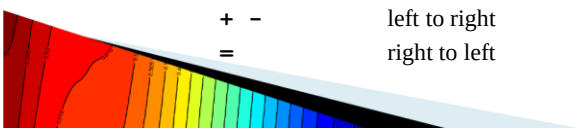
  - `(float) a` forces the conversion of `a` to type `float`,
  - according to the implicit rule of promotion, `b` is converted to type `float`,
  - the division is then evaluated as a division of real numbers and the result is assigned to `z` which is of type `float`.

- A unary operator operates on one argument only: the argument directly to its right.

- A cast operator exists for each data type in C: `(double)`, `(float)`, `(int)`, `(char)`.

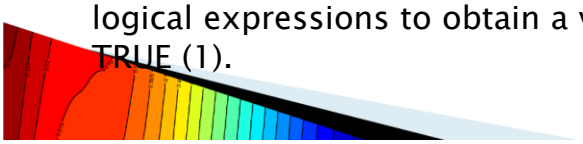
Summary of Hierarchy and Associativity of Arithmetic Operators

`()`	left to right
`- (type)`	right to left
`* / %`	left to right
`+ -`	left to right
`=`	right to left



Logical Expressions

- Logical expressions give a TRUE / FALSE result
  - In C FALSE is the value 0 and TRUE any other value
- We shall see soon that logical expressions plays a role in programming “decisions” and “repetitions”.
- Logical expressions can compare values
- Logical expressions can also include a number of comparisons
- Expressions involving operators comparison operators such as “equal”, “greater than”, and logical operators such as AND, OR, and NOT
- Like arithmetic expressions the computer evaluates logical expressions to obtain a value: FALSE (0) or TRUE (1).



Logical Expressions in C

Relational, Equality and Logical Operators in C

- The following relational operators exist in C:

Operation	C Operator	C Example	Description
x>y	>	`x > y`	***is greater than***
x<y	<	`x < y`	***is less than***
x≥y	>=	`x >= y`	***is greater or equal to***
x≤y	<=	`x <= y`	***is less than or equal to***

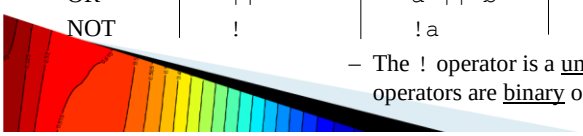
- The following equality operators exist in C:

Operation	C Operator	C Example	Description
x=y	==	`x == y`	***is equal to***
x≠y	!=	`x != y`	***is not equal to***

- The following logical operators exist in C:

Operation	C Operator	C Example	Description
AND	`&&`	`a && b`	***and***
OR	`||`	`a || b`	***or***
NOT	!	`!a`	***not***

  - The `!` operator is a unary operator while the `&&` and `||` operators are binary operators.

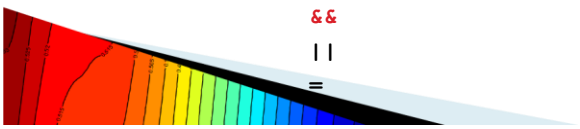


Rules of Logical Operator Precedence in C

- In a logical expression,
  - the logical not `!` operator has highest precedence,
  - the relational operators have the next highest precedence,
  - then come the equality operators,
  - then the logical `&&` operator,
  - and finally the logical `||` operator which has the lowest precedence.

- The following table summarizes in order of precedence the associativity of all the C operators that we have seen to date.

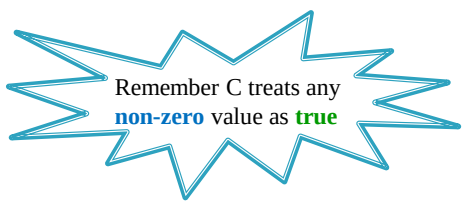
`()`	left to right	
`.`	left to right	
`- ! (type)`	right to left	**unary operators**
`* / %`	left to right	
`+ -`	left to right	
`< <= > >=`	left to right	
`== !=`	left to right	
`&&`	left to right	
`||`	left to right	
`=`	right to left	

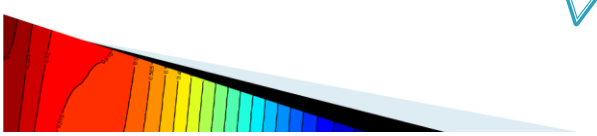


29

- Parentheses can be used in a logical expression. They have the highest precedence and thus can be used to change the precedence of logical operations just like in an arithmetic expression.
- The above rules of precedence and associativity are followed by the C compiler when evaluating expressions that contain an arithmetic part and a logical part.

  ➔ You are strongly encouraged to use parentheses in order to clarify such mixed expressions.
- The result of a logical expression can only be true or false.
- In C, a logical expression that evaluates to **false** has the integer result `0` and a logical expression that evaluates to **true** has the integer value `1`.





30

- In C, a non-zero integer value (`1`, `2`, `-1`...) is interpreted as being true in a logical expression. Suppose that the variables `x` and `y` are of type `int`; shown below are the truth tables for our logical operators in C, operating on the variables `x` and `y`.
  - Truth table for the logical and:

X	Y	X and Y
T	T	T
T	F	F
F	T	F
F	F	F

In a C expression

x	y	x && y
NZ	NZ	1
NZ	0	0
0	NZ	0
0	0	0

  - Truth table for the logical or:

X	Y	X or Y
T	T	T
T	F	T
F	T	T
F	F	F

In a C expression

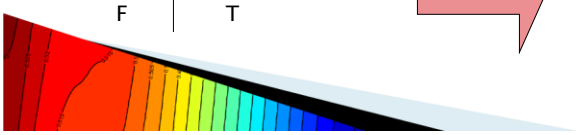
x	y	x || y
NZ	NZ	1
NZ	0	1
0	NZ	1
0	0	0

  - Truth table for the logical not:

X	not X
T	F
F	T

In a C expression

x	!x
NZ	0
0	1



- Here are a few examples of valid logical expressions in C:

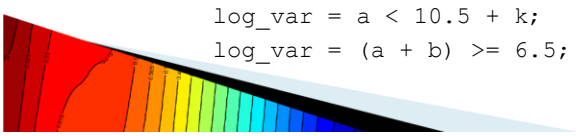
```
int k = -3, m = -4;
float a = 5.5, b = 1.5;
```

	Result
`a < 10.5 + k;`	1
`a + b >= 6.5;`	1
`k != a - b;`	1
`b - k > a;`	0
`!( a == 3 * b);`	1
`-k <= k + 6;`	1
`a < 10 && a > 5;`	1
`k / m >= 0.75;`	0

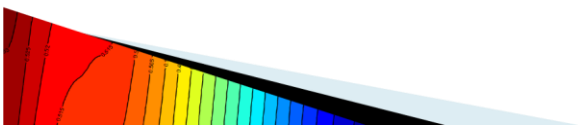
- In C, logical expressions always evaluate to the integers `1` or `0` (for true or false).
  - ➔ We can use a variable of type `int` to store the result of a logical expression and treat this variable as a logical variable.

E.g.:

```
int log_var, k = -3, l = -4;
float a = 5.5, b = 1.5;
log_var = a < 10.5 + k;        → log_var  1
log_var = (a + b) >= 6.5;      → log_var  1
```



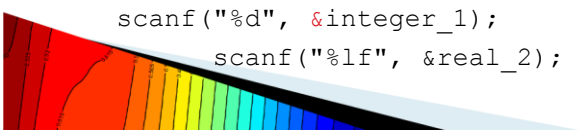
Topic 3: More on Input and Output



Review of Basic Input and Output

- C standard functions are used for input/output: `printf` to display text on the console, and `scanf` to read and convert typed text from the keyboard.
  - The header file `stdio.h` contains the required prototypes and definitions
- Conversion specifiers are used by both functions for converting text to and from data values
  - `%d` converts integer values (type int) used by both `printf` and `scanf`
  - `%f` converts real values (type double) for `printf`
  - `%lf` converts real values (type double) for `scanf`
- Recall that in the case of `scanf` the & operator is used to pass the address of the variable for storing values read from the keyboard. In the case of the `printf`, the value of the variable is passed to the function.
- Examples:

```
printf("The integer value of var is %d\n",var)
printf("The sum is %f\n", sum);
scanf("%d", &integer_1);
scanf("%lf", &real_2);  // real_2 is a double
```

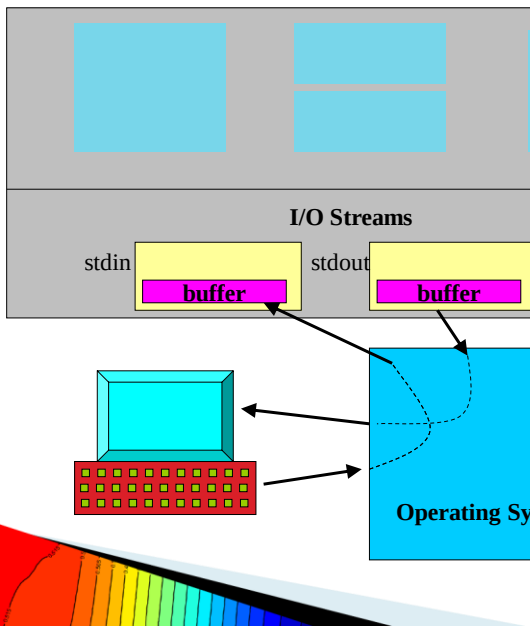


Programming Model and I/O

- For input, the operating system fills a « buffer » with a string
  - See the next slide
  - The characters typed up to the carriage return are stored in the buffer (`stdin` structure this is the standard input stream)
  - The string becomes available to the program only after the carriage return
  - The conversion specifiers are used by `scanf` to know how to convert the characters in the string to the desired values
- For output, `printf` will convert values passed and insert into a string (buffer, `stdout` structure is the standard output stream) used by the OS for display on the console.

35

Programming Model and I/O



stdin and stdout are structure variables that contain variables and buffers (strings) used for supporting I/O

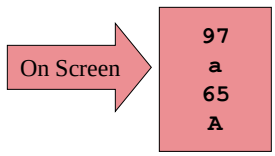
36

Character Output

- Recall that
  - A content of a variable of type `char` can be interpreted as a character or an integer
  - A variable of type `int` containing an integer in the range `0` through `127` inclusively, can be interpreted as being a character or an integer.
  - The content of a variable of type `int` can be interpreted as an integer.
- The conversion specifier `%d` is to be used when we wish to print out the content of a variable as an integer and the conversion specifier `%c` is used when we wish to interpret the variable as a character.

E.g.:

```
char character_1 = 'a';
int integer_1 = 65;
printf("%d\n", character_1);
printf("%c\n", character_1);
printf("%d\n", integer_1);
printf("%c\n", integer_1);
```



37

Character input: the input stream

- Recall that the input stream provides the buffer containing the input data
  - `scanf()` is used to read a character from this buffer using the %c code conversion specifier

E.g.:

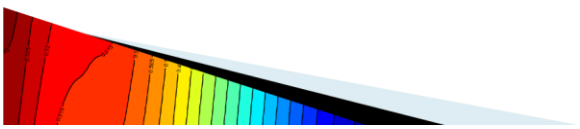
```
scanf("%c", &character_1);
scanf("%c", &integer_1);
```

- Consider the entry of a single character:
  - We type the character on the keyboard and then we hit the enter or carriage return key to signify that we have completed typing in the entry.
    - ✓ This signals the OS that the data can be passed to the program in the buffer of the input stream.
  - The carriage return inserts the character new line (ASCII equivalent 10) into the input stream after our typed character.
  - The input stream thus contains two item:
    - The character typed in and the new line symbol (ASCII equivalent 10).

→ When we read in a character using `scanf`, we must usually clear the input stream to get rid of the new line symbol; this can be achieved by invoking the function `fflush`.

38

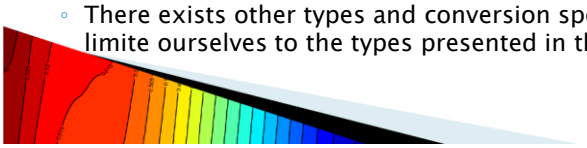
- The input stream is referenced with `stdin`, which is used as the argument in `fflush`.
  - E.g.: `fflush(stdin);`
  - The header file `stdio.h` contains the prototype of `fflush` and the definition of `stdin`
  - `stdin` stands for standard input stream; it is a pointer that points to the standard input that contains the memory buffer where the keyboard entries are stored by the OS.
- It is good practice to clear the input stream using `fflush` before/after reading in characters.



Summary of Conversion Specifiers to be Used in printf and scanf

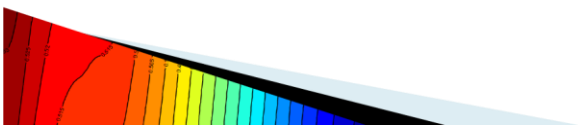
Type	printf	scanf
char	%c, %d,%i	%c
int	%d, %i	%d, %i
float	%f, %e, %E, %g, %G	%f, %e, %E, %g, %G
double	%f, %e, %E, %g, %G	%lf, %le, %lE, %lg, %lG

- `%i` is equivalent to `%d`
- `%d` and `%i` converts the char to an integer value for `char`
- Note the differences for conversion specifiers for a `double` and float. The same specifiers are used for `printf`, but for `scanf`, an "`l`" is added for the `double`.
- The conversion specifier `%e` prints out the number in scientific notation: `1.0e0`
- The conversion specifier `%E` prints out the number in scientific notation: `1.0E0`
- The conversion specifier `%g` prints out the number in `%f` or in `%e`
- The conversion specifier `%G` prints out the number in `%f` or in `%E`
- There exists other types and conversion specifiers in C, but we shall limite ourselves to the types presented in this course.



Example: characters.c

- Program shows how `char` and `int` variables can be treated as both integers and characters.
- Program demonstrates reading characters with `scanf` and how `fflush` is used to clear the input buffer.

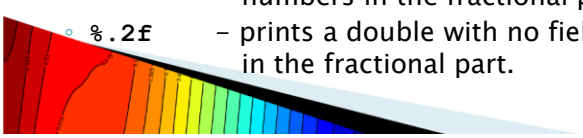


On Formatting Output

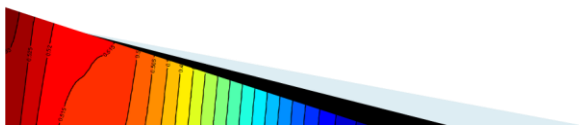
- The following shows examples on formatting numbers to help
  - Align them within fields to create tables
  - Determine how many decimal numbers to use in the fractions of real numbers.
- Simple presentation: **`%w.pc`** where
  - **`w`** defines a minimum field width (minimum amount of space the value occupies)
  - **`p`** precision (defines the number of decimal numbers in the fraction of a real number)
  - **`c`** the conversion character (e.g. f, d, etc.)

See C Reference card for details

- Examples:
  - **`%10d`** – prints an integer value in a field of 10 characters,
  - **`%10.2f`** – prints a double value in a field of 10 characters with 2 numbers in the fractional part.
  - **`%.2f`** – prints a double with no field definition and 2 numbers in the fractional part.



Topic 4 Math Library



Standard Math Functions

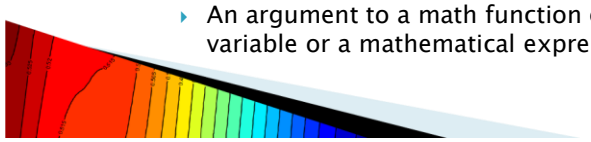
- The following slide provides a list standard C math functions.
  - The most popular math functions are available.
  - Google them to get details or consult the appendix of the recommended text.
- The preprocessor directive `#include <math.h>` provides the C compiler with definitions used by the standard math functions in the program. This directive is normally placed after the directive `#include <stdio.h>` before `main()`.
- All standard math functions except three (`frexp`, `ldexp` and `modf`) require arguments of type `double` and all standard math functions return a number of type `double`.
- The value returned by a math function can be
  - assigned to a variable for future use:

```
x = sin(angle);
```

  - used in an arithmetic expression, or as an argument sent in to another function:

```
c = sqrt(PI + pow(z, 3.0) );
```

    - An argument to a math function can be a symbolic constant, a variable or a mathematical expression.



- The rules of promotion are applied to the arguments of math functions if they are not of type `double`. These same rules are applied to the value returned by the function if it is not stored into a variable of type `double`.
- Some of the most commonly encountered math functions:

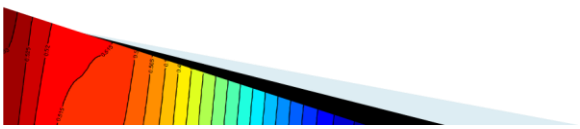
Math Function	C Math Function	Note on Usage
$\sqrt{x}$	`sqrt(x)`	x must be ≥ 0.0
x^y	`pow(x,y)`	
e^x	`exp(x)`	
$\log_e x$	`log(x)`	
$\log_{10} x$	`log10(x)`	
$\lvert x \rvert$	`fabs(x)`	
sin(x)	`sin(x)`	x is in radians.
cos(x)	`cos(x)`	
tan(x)	`tan(x)`	
arcsin(x)	`asin(x)`	$-1 \le x \le 1$ and $-\pi/2 \le$ return $\le \pi/2$, in radians.
arccos(x)	`acos(x)`	$-1 \le x \le 1$ and $0 \le$ return $\le \pi$, in radians.
arctan(x)	`atan(x)`	$-\pi/2 \le$ return $\le \pi/2$, in radians.
sinh(x)	`sinh(x)`	Hyperbolic functions.
cosh(x)	`cosh(x)`	
tanh(x)	`tanh(x)`	

- The function `ceil(x)` rounds `x` to the smallest integer that is greater than or equal to `x`.
  - E.g.:
    - `ceil(9.0)` yields 9.0
    - `ceil(9.2)` yields 10.0
    - `ceil(9.5)` yields 10.0
    - `ceil(9.8)` yields 10.0
    - `ceil(-9.8)` yields -9.0
    - `ceil(-9.5)` yields -9.0
    - `ceil(-9.2)` yields -9.0
    - `ceil(-9.0)` yields -9.0

  Returns an integer as a `double`.
- The function `floor(x)` rounds `x` to the largest integer that is less than or equal to `x`.
  - E.g.:
    - `floor(9.0)` yields 9.0
    - `floor(9.2)` yields 9.0
    - `floor(9.5)` yields 9.0
    - `floor(9.8)` yields 9.0
    - `floor(-9.8)` yields -10.0
    - `floor(-9.5)` yields -10.0
    - `floor(-9.2)` yields -10.0
    - `floor(-9.0)` yields -9.0

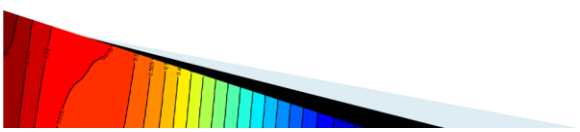
  Returns an integer as a `double`.

Topic 5: Debugging



Types of Errors Encountered in Computing

- Syntax errors.
  - They are C "grammar and spelling" errors.
  - They occur during compilation.
  - The compiler does not understand what you are trying to do.
  - The compiler usually provides a clue as to where the error is.
  - Re-compile your code after correcting a few syntax errors since a single error often causes many others to occur.
- Execution and logical errors.
  - The program compiles without error but it does not run as expected.
  - May have a fatal error during execution such as a division by zero
    - E.g.: `c = a / b;` with `b` containing `0`.
  - May have logical errors in instructions.
  - Debugging is the art of finding and correcting logical errors



Debugging Logical Errors

- To find and correct logical errors, be systematic
  - Trace the program (e.g. using programming model) to see how variables change during the execution of the program.
  - Add `printf` instructions in your program to output values of variables to trace how they change.
  - Use the debugger if the programming environment has one (ours does).
- Finding where variables have unexpected results allows you to zero in on statements that are causing the errors
- Golden Rule: when debugging, make changes for a reason to your code to correct the error.

Next Module

- Topic 1: Structures
- Topic 2: Arrays
- Topic 3: More Standard C Functions