ESC101N
Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

Variables

- Variables signify data that may be modified
- Name of a variable can contain letters, digits and underscore (_)
- Example: `i`, `y2k`, `big_name`, `bigger_name_2`
- Case-sensitive: `camel`, `CAMEL` and `CaMeL` are different

Variables

- Variables signify data that may be modified
- Name of a variable can contain letters, digits and underscore (_)
- Example: `i`, `y2k`, `big_name`, `bigger_name_2`
- Case-sensitive: `camel`, `CAMEL` and `CaMeL` are different
- Name cannot start with a digit
- Example: `1d` is not valid
- Name can start with an underscore, but do not do so
- Example: avoid valid names such as `_bad`

Variables

- Variables signify data that may be modified
- Name of a variable can contain letters, digits and underscore (_)
- Example: `i`, `y2k`, `big_name`, `bigger_name_2`
- Case-sensitive: `camel`, `CAMEL` and `CaMeL` are different
- Name cannot start with a digit
- Example: `1d` is not valid
- Name can start with an underscore, but do not do so
- Example: avoid valid names such as `_bad`
- Certain keywords are special
- They are reserved and cannot be used
- Example: `main`, `if`

Types of variables

- Each variable has a type that signifies the domain of values

Domain	Type
integer	int, char
real	double, float
character	char
boolean	int

Types of variables

- Each variable has a type that signifies the domain of values

Domain	Type
integer	int, char
real	double, float
character	char
boolean	int

- Initial values of variables are specified as constants of the same type
- Examples:
  - `int i = 0;`
  - `double d = 1.4;`
  - `char c = 'A';`

Types of variables

- Each variable has a type that signifies the domain of values

Domain	Type
integer	int, char
real	double, float
character	char
boolean	int

- Initial values of variables are specified as constants of the same type
- Examples:
  - `int i = 0;`
  - `double d = 1.4;`
  - `char c = 'A';`
- Types are *not* mathematically equivalent to domain
- They capture only a subset
- Real numbers of arbitrary precision cannot be represented
  - `double` is more accurate than `float`
  - 1/3 is printed as 0.3333334326.. as a float, but 0.3333333333.. as a double

More on types

- There is no *boolean* or *truth type* in C
- Integers are treated as booleans
- Value 0 represents *false*
- Any non-negative value (typically 1) represents *true*
- Examples:
  - `(3 > 5)` is printed as 0 whereas `(3 < 5)` is printed as 1

More on types

- There is no *boolean* or *truth type* in C
- Integers are treated as booleans
- Value 0 represents *false*
- Any non-negative value (typically 1) represents *true*
- Examples:
  - `(3 > 5)` is printed as 0 whereas `(3 < 5)` is printed as 1
- Characters are special integers of much shorter size
- 8 bits are used
- *Only* 256 characters can be represented
- Unicode includes characters from all languages of the world
- ASCII specifies a standard that maps characters to integers (between 0 and 255)
- Examples:
  - `'a'` is equivalent to 97, `'A'` to 65, `'0'` to 48, `'.'` to 46
  - Look up ASCII chart for complete list

```
printf(``%d'', 'a');
printf(``%c'', 97);
printf(``%c'', 353);
```

Input and output of variables

- Correct type specification must be used

Type	Specification
int	%d
double	%f
float	%f
char	%c

Input and output of variables

- Correct type specification must be used

Type	Specification
int	%d
double	%f
float	%f
char	%c

- `scanf` is for input
- Format: `scanf(``<specification>'', &<name>);`
- Examples:
  - `i` is an `int`: `scanf(``%d'', &i);`
  - `c` is a `char`: `scanf(``%c'', &c);`
  - `d` is a `double`: `scanf(``%f'', &d);`

Input and output of variables

- Correct type specification must be used

Type	Specification
int	%d
double	%f
float	%f
char	%c

- `scanf` is for input
- Format: `scanf(‘‘<specification>’’, &<name>);`
- Examples:
  - `i` is an `int`: `scanf(‘‘%d’’, &i);`
  - `c` is a `char`: `scanf(‘‘%c’’, &c);`
  - `d` is a `double`: `scanf(‘‘%f’’, &d);`
- `printf` is for output
- Format: `printf(‘‘<specification>’’, <name>);`
- Examples:
  - `i` is an `int`: `printf(‘‘%d’’, i);`
  - `c` is a `char`: `printf(‘‘%c’’, c);`
  - `d` is a `double`: `printf(‘‘%f’’, d);`

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` = -3 (non-standard)
- `33.0 / 5.0` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` = -3 (non-standard)
- `33.0 / 5.0` = 6.6
- `'a' + 2` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` = -3 (non-standard)
- `33.0 / 5.0` = 6.6
- `'a' + 2` = 'c'
- `'a' - 2` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` = -3 (non-standard)
- `33.0 / 5.0` = 6.6
- `'a' + 2` = 'c'
- `'a' - 2` = '_'
- `'A' + '1'` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` = -3 (non-standard)
- `33.0 / 5.0` = 6.6
- `'a' + 2` = 'c'
- `'a' - 2` = '_'
- `'A' + '1'` = 65 + 49 = 114 = 'r' (avoid)
- `'1' * 2` =

Arithmetic operators

Operator	Meaning	`int`	`double`, `float`	`char`
+	addition	yes	yes	restricted
-	subtraction	yes	yes	restricted
*	multiplication	yes	yes	best to avoid
/	division	integer	yes	best to avoid
%	modulus	yes	no	best to avoid

- `33 / 5` = 6
- `33 % 5` = 3
- `-33 / 5` = -6 (non-standard)
- `-33 % 5` = -3 (non-standard)
- `33.0 / 5.0` = 6.6
- `'a' + 2` = 'c'
- `'a' - 2` = '_'
- `'A' + '1'` = 65 + 49 = 114 = 'r' (avoid)
- `'1' * 2` = 49 * 2 = 98 = 'b' (avoid)

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- 33 > 5 =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` = 0
- `'a' == 97` =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` = 0
- `'a' == 97` = 1
- `'a' == 353` =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` = 0
- `'a' == 97` = 1
- `'a' == 353` = 0
- `'a' == 97.0` =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` = 0
- `'a' == 97` = 1
- `'a' == 353` = 0
- `'a' == 97.0` = 1
- `96.0 == 97` =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` = 0
- `'a' == 97` = 1
- `'a' == 353` = 0
- `'a' == 97.0` = 1
- `96.0 == 97` = 0
- `97.0 == 97` =

Relational operators

Operator	Meaning	`int`	`double`, `float`	`char`
>	greater than	yes	yes	yes
>=	greater than or equal to	yes	yes	yes
<	lesser than	yes	yes	yes
<=	lesser than or equal to	yes	yes	yes
==	equal to	yes	yes	yes
!=	not equal to	yes	yes	yes

- `33 > 5` = 1
- `'a' > 'b'` = 0
- `'a' == 97` = 1
- `'a' == 353` = 0
- `'a' == 97.0` = 1
- `96.0 == 97` = 0
- `97.0 == 97` = 1 (avoid such automatic type conversions)

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- !4 =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` = 0
- `4 || 0` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` = 0
- `4 || 0` = 1
- `!'a'` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` = 0
- `4 || 0` = 1
- `!'a'` = 0
- `'\0'` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` = 0
- `4 || 0` = 1
- `!'a'` = 0
- `'\0'` = 0
- `'0'` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` = 0
- `4 || 0` = 1
- `!'a'` = 0
- `'\0'` = 0
- `'0'` = 1
- `'a' || '\0'` =

Logical or boolean operators

Operator	Meaning	`int`	`double`, `float`	`char`
!	not	yes	yes	yes
&&	logical and	yes	yes	yes
||	logical or	yes	yes	yes

- `!4` = 0
- `!4.0` = 0
- `4 && 5` = 1
- `4.0 && 5.0` = 1
- `4 && 0` = 0
- `4 || 0` = 1
- `!'a'` = 0
- `'\0'` = 0
- `'0'` = 1
- `'a' || '\0'` = 1

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- `45 + -33 =`

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- 45 + -33 = 12
- 45 + +33 =

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- `45 + -33` = 12
- `45 + +33` = 78
- `i = 33; i++;` :

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- `45 + -33` = 12
- `45 + +33` = 78
- `i = 33; i++;` : i becomes 34
- `i = 33; i--;` :

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- `45 + -33` = 12
- `45 + +33` = 78
- `i = 33; i++;` : i becomes 34
- `i = 33; i--;` : i becomes 32
- `c = 'g'; ++c;` :

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- `45 + -33` = 12
- `45 + +33` = 78
- `i = 33; i++;` : i becomes 34
- `i = 33; i--;` : i becomes 32
- `c = 'g'; ++c;` : c becomes 'h' (for ASCII standard)
- `c = 'g'; --c;` :

Unary operators

Operator	Meaning	`int`	`double`, `float`	`char`
(unary) +	positive	yes	yes	best to avoid
(unary) -	negative	yes	yes	best to avoid
++	increment by 1	yes	yes	yes
--	decrement by 1	yes	yes	yes
=	assignment	yes	yes	yes

- `45 + -33` = 12
- `45 + +33` = 78
- `i = 33; i++;` : i becomes 34
- `i = 33; i--;` : i becomes 32
- `c = 'g'; ++c;` : c becomes 'h' (for ASCII standard)
- `c = 'g'; --c;` : c becomes 'f' (for ASCII standard)

Expressions

- An expression is any legal combination of variables, constants and operators
- $12 + 6 \ / \ 3 =$

Expressions

- An expression is any legal combination of variables, constants and operators
- $12 + 6 / 3 = 14$
  - / evaluated earlier than + due to higher precedence
- $24 / 6 * 2 =$

Expressions

- An expression is any legal combination of variables, constants and operators
- $12 + 6 / 3 = 14$
  - / evaluated earlier than + due to higher precedence
- $24 / 6 * 2 = 8$
  - / is of same precedence as $*$ and order of evaluation is left to right
- $24 / 6 / 2 =$

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 =

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 = 3
  - - is evaluated from left to right

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 = 3
  - - is evaluated from left to right
- Brackets () are needed to enforce particular order (remember BODMAS rule)
  - (12 + 6) / 3 =

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 = 3
  - - is evaluated from left to right
- Brackets () are needed to enforce particular order (remember BODMAS rule)
  - (12 + 6) / 3 = 6
  - 24 / (6 * 2) =

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 = 3
  - - is evaluated from left to right
- Brackets () are needed to enforce particular order (remember BODMAS rule)
  - (12 + 6) / 3 = 6
  - 24 / (6 * 2) = 2
  - 24 / (6 / 2) =

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 = 3
  - - is evaluated from left to right
- Brackets () are needed to enforce particular order (remember BODMAS rule)
  - (12 + 6) / 3 = 6
  - 24 / (6 * 2) = 2
  - 24 / (6 / 2) = 8
  - 12 - (6 - 3) =

Expressions

- An expression is any legal combination of variables, constants and operators
- 12 + 6 / 3 = 14
  - / evaluated earlier than + due to higher precedence
- 24 / 6 * 2 = 8
  - / is of same precedence as * and order of evaluation is left to right
- 24 / 6 / 2 = 2
  - / is evaluated from left to right
- 12 - 6 - 3 = 3
  - - is evaluated from left to right
- Brackets () are needed to enforce particular order (remember BODMAS rule)
  - (12 + 6) / 3 = 6
  - 24 / (6 * 2) = 2
  - 24 / (6 / 2) = 8
  - 12 - (6 - 3) = 9
- Best, of course, is to bracket up anyway to avoid confusion

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6 =`

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6` =
  `(((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)))`

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6` $=$
  `(((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)))` $= 1$
- How did we decide?

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6` =
  `(((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)))` = 1
- How did we decide?
- By using rules of operator precedence and associativity

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6` =
  `(((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)))` = 1
- How did we decide?
- By using rules of operator precedence and associativity
- Precedence: among multiple operators, precedence rules guide the order in which they will be evaluated
  - Operators having higher precedence are evaluated earlier
  - 12 + 6 / 3 is really 12 + (6 / 3)

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6` =
  `(((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)))` = 1
- How did we decide?
- By using rules of operator precedence and associativity
- Precedence: among multiple operators, precedence rules guide the order in which they will be evaluated
  - Operators having higher precedence are evaluated earlier
  - 12 + 6 / 3 is really 12 + (6 / 3)
- Associativity: among multiple instances of the same operator, associativity rules guide the order in which they will be evaluated
  - If an operator is left-to-right associative, the leftmost instance of the operator is evaluated first, and so on
  - 12 - 6 - 3 is really (12 - 6) - 3

Evaluation of expressions

- `- 6 / 3 * - 5 < 8 || 0 > 5 - 6` =
  `(((((-6) / 3) * (-5)) < 8) || (0 > (5 - 6)))` = 1
- How did we decide?
- By using rules of operator precedence and associativity
- Precedence: among multiple operators, precedence rules guide the order in which they will be evaluated
  - Operators having higher precedence are evaluated earlier
  - 12 + 6 / 3 is really 12 + (6 / 3)
- Associativity: among multiple instances of the same operator, associativity rules guide the order in which they will be evaluated
  - If an operator is left-to-right associative, the leftmost instance of the operator is evaluated first, and so on
  - 12 - 6 - 3 is really (12 - 6) - 3
- It is best to avoid using such expressions without bracketing

Operator precedence and associativity

- Unary operators: right-to-left
- Arithmetic operators: left-to-right
- Relational operators: left-to-right
- Logical operators: left-to-right

Operator precedence and associativity

- Unary operators: right-to-left
- Arithmetic operators: left-to-right
- Relational operators: left-to-right
- Logical operators: left-to-right

Operator	Precedence	Associativity
! ++ -- (unary)+ (unary)-	Highest	right-to-left
* / %		left-to-right
+ -		left-to-right
< <= > >=		left-to-right
== !=		left-to-right
&&		left-to-right
||		left-to-right
=	Lowest	right-to-left