CMPT 102
Introduction to Scientific Computer Programming

Expressions and Operators
Program Style

© Janice Regan, CMPT 102, Sept. 2006

Components of a C Program

- More types of tokens (smallest individual units of a program) include:
  - ***Other separators*** blanks, tabs, line feed, blank lines …
  - ***Operators*** actions used to combine constants
    - = and + in the expression

    y = x + THREE;

© Janice Regan, CMPT 102, Sept. 2006

C: Binary Arithmetic Operators

- A Binary Operator operates on two arguments
  - \+ addition
  - \- subtraction
  - * multiplication
  - / division
  - % modulus or remainder (only for integers, 5%2=1)
- Evaluated left to right

© Janice Regan, CMPT 102, Sept. 2006

Unary Arithmetic Operators in C

- A Unary Operator operates on one argument
  - \+ positive
  - \- negative
  - ++ increment
  - -- decrement
  - ~ ones complement
- Evaluated right to left

© Janice Regan, CMPT 102, Sept. 2006

Expressions in C

- An expression can be a single variable, or can be a series of variables combined using operators
- Arithmetic expressions manipulate numeric variables following the rules of algebra
- Relational expressions compare numerical values and give a logical answer (more later)
- The two variables operated on by a binary arithmetic or relational operator should both have the same type
  - If they have different types one must have it's type converted to the type of the other

© Janice Regan, CMPT 102, Sept. 2006

Precedence of operators in C

1. ( ) [] . innermost first ++ -- (post)
2. ++ -- (pre) + - ! ~ & *(unary operators)
3. * / %
4. + -
5. = += -= *= /= %=

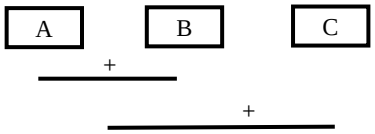
- Evaluate 2 and 5 right to left
- Evaluate 1, 3, and 4 left to right

© Janice Regan, CMPT 102, Sept. 2006

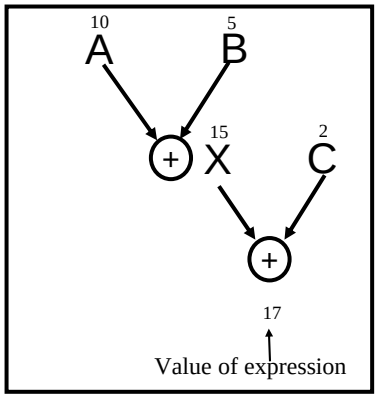
Expressions: arithmetic operators

- A + B + C

  X + C



- Let A=10, B=5, C=2

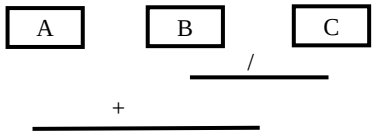


© Janice Regan, CMPT 102, Sept. 2006

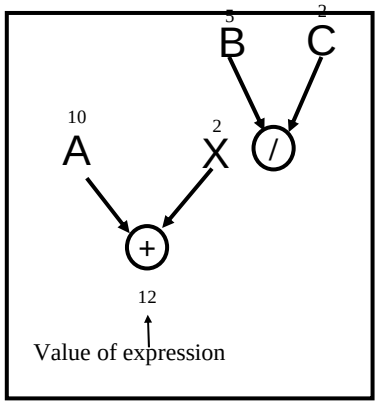
Expressions: arithmetic operators

- Order of operations is determined by operator precedence rules / before +
- A + B / C

  A + X



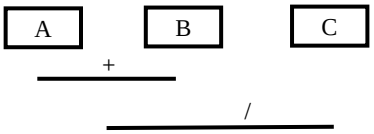
- Let A=10, B=5, C=2



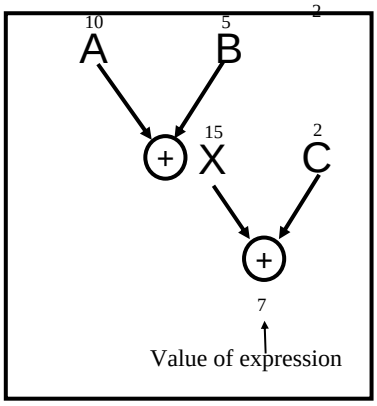
© Janice Regan, CMPT 102, Sept. 2006

Importance of order of operations

- Order of operations is determined by operator precedence rules () before /
- (A + B) / C
  X / C



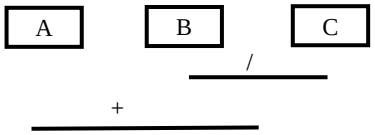
- Let A=10, B=5, C=2



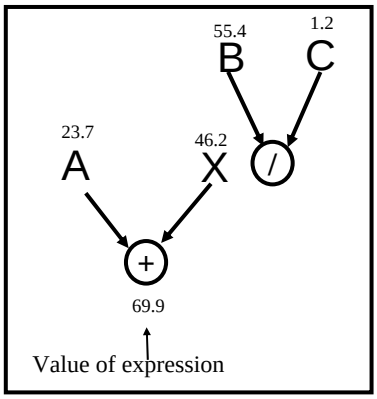
© Janice Regan, CMPT 102, Sept. 2006

Expressions: arithmetic operators

- Types of operands: float vs. integer divide
- An operator always operates on two operands of the same type
- A + B / C
  A + X



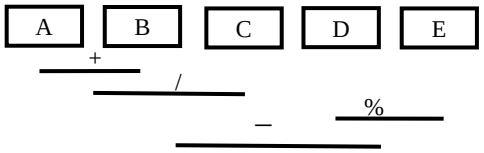
- Let A=23.7, B=55.4, C=1.2

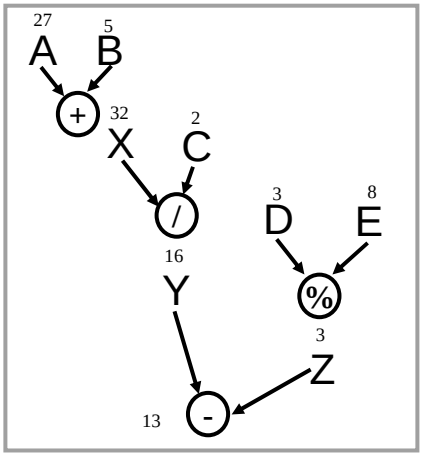


© Janice Regan, CMPT 102, Sept. 2006

Expressions: arithmetic operators

- Let A=27, B=5, C=2, D=3, E=8
- ((A + B) / C) – D % E
  (X / C) – D % E
  Y – D % E
  Y – Z

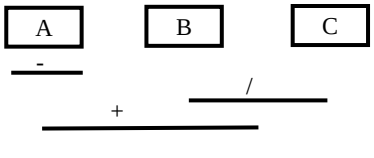




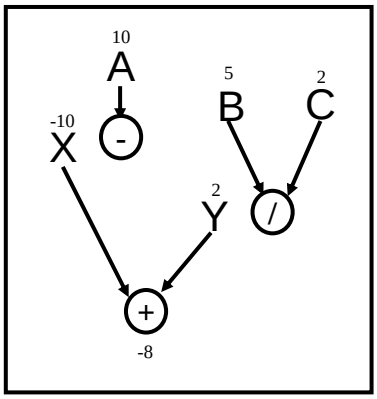
© Janice Regan, CMPT 102, Sept. 2006

Expressions: arithmetic operators

- Order of operations is determined by operator precedence rules unary -, before /, before +
- -A + B / C
  X + B / C
  X + Y



- Let A=10, B=5, C=2



© Janice Regan, CMPT 102, Sept. 2006

Expressions: arithmetic operators

- ++A – B pre increment
- --B + C pre decrement
- Pre-increment: increment variable then use in expression
- A * C++ post increment
- A / C-- post decrement
- Post increment: use variable in expression then increment the variable

Assignment Statements

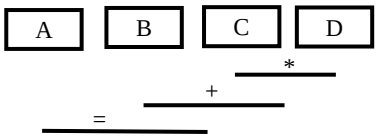
- Basic statement used to perform calculations
- Form: *result* = *expression*;
- Example: A = B + C * D;
- NOT the same as an = in an equation
  - Each variable is associated with a location in memory
  - Evaluate the expression on the left (B+C*D)
    - Multiply the value in the memory location associated with variable C by the value in the memory location associated with variable C
    - Add the product to the value of the in the memory location associated with variable B
    - The sum is the value of the expression
  - The value of the expression on the right hand side is placed in the memory location associated with variable A

Expressions: arithmetic operators

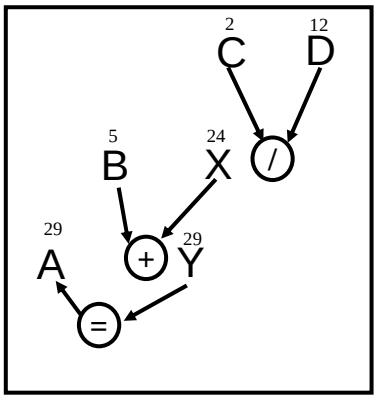
- Order of operations is determined by operator precedence rules * before + before =
- A = B + C * D

  A = B + X

  A = Y



- A=10, B=5, C=2, D=12



© Janice Regan, CMPT 102, Sept. 2006

Assignment Statements:

- Form: *result* = *expression*;
- Example: X = X * Y;
  - Each variable is associated with a location in memory
  - Evaluate the expression on the left (X*Y)
    - Multiply the value in the memory location associated with variable X by the value in the memory location associated with variable Y
    - The product is the value of the expression
  - The product is placed in the memory location associated with variable X overwriting the previous value of X

© Janice Regan, CMPT 102, Sept. 2006

Assignment operators

- A = B assign value of expression B to variable A, store result in A
- A += B add the value of expression B to the value of variable A, store result in A
- A -= B subtract the value of expression B from the value of variable A, store result in A
- A *= B multiply the value of expression B by the value of variable A, store result in A
- A /= B divide the value of expression A by the value of variable B, store result in A

Numerical Values and Expressions

- When you evaluate an expression in C you may combine two values (operands) according to a binary operation.
- Two operands A and B combine with operation +
  - A+B
- Both operands of a binary operation should have the same type (int, float, double ….)
- Conversions can be done automatically or can be done explicitly by the programmer

Numerical Values and Expressions

- In C if you use two types of operands with the same binary operator, one of the operands will be converted to the same type as the other operand.
  - When evaluating expressions such conversions and promotions are automatically performed according to a defined set of rules, ***the usual arithmetic conversions***.
- To avoid unexpected results you should understand how these conversions are done.
- In some cases to be sure you get the results you want you should do conversions explicitly yourself

Explicit conversion: The cast operation

- In C you can explicitly convert the type of a variable or expression within a larger expression using a cast operator
  - The value of the variable or expression is not changed
  - The value used in the larger expression is converted to the requested type
- Sample expressions including casts
  - Integerone + (int)(Floatone+Floattwo)
  - (float)Integerone + Float1 + Float2
  - (double)unsigned1+(double)unsigned2 * double2

Conversions

- When evaluating an arithmetic expression not including an assignment statement the usual arithmetic conversions are used
- When executing an assignment statement (A=B) the value of expression B is placed in location A in memory.
  - In this case the type of variable B must be converted to the type of variable A
  - This can result in a loss of accuracy

Usual arithmetic conversions

- If either operand is long double the other is converted to a long double
- Otherwise, If either operand is a double the other is converted to a double
- Otherwise, If either operand is a float the other is converted to a float
- Otherwise, Integral promotions are performed on both operands
  - short integer and character variables are converted to integers if all their values can be represented as integers
  - Otherwise they are converted to unsigned integers

Usual arithmetic conversions

THEN

- If either operand is an unsigned long int the other is converted to an unsigned long int
- Otherwise, If one operand is a long int and the other is an unsigned int then
  - If the long int can represent all possible values of the unsigned integer then the unsigned int is converted to a long int
  - Otherwise both are converted to unsigned long int
- Otherwise, If one operand is a long int the other is converted to a long int
- Otherwise, If one operand is a unsigned int the other is converted to a unsigned int
- Otherwise, both operands have type int

How are conversions done 1

- An integer being converted to floating point number will take on the closest number to the value of the integer with a representable value. This value may not be exactly equal to the integer value.
- For example if the closest representable values to 2745 are 2745.00014 and 2744.99973 Then the converted value of 2745 will be 2745.00014
- A long double being converted to a double will similarly take on the closest representable double value
- A double converted to a long double has the same value.
  - All representable double values are representable long double values
  - Some long double values are exactly respresentable as double values

How are conversions done 2

- When a float is converted to an integer the fractional part is discarded. (the result is not defined if the result can't be represented as an integer of the specified type)
- When an integer is converted to a float the value is the next higher of lower representable value.
- An integer is converted to an unsigned type by "finding the smallest non-negative value that is congruent to that integer modulo one more than the largest value"
- An unsigned integer converted to a signed integer is unchanged so long as it can be represented
- A higher precision float converted to a lower precision float the result is the closest representable value
- A lower precision float converted to a higher precision float is unchanged

© Janice Regan, CMPT 102, Sept. 2006

Program Style

- The language being used, in our case C++ has its own syntax (structure and rules)
- Projects usually set up a standard style for program layout
  - Capitalization, spacing, form of comments
  - Specifics of chosen style not as important as the fact that there *is* a common style
  - Common style is usually more restrictive than the syntax of the language
- Common style for consistency, clarity, ease of understanding and modification

© Janice Regan, CMPT 102, Sept. 2006

Program Style

- Bottom-line: Make programs easy to read and modify
- Comments, two methods:
  - Comment on separate line (usually explains a block of code)
    - /*Delimiters indicates everything between is ignored*/
  - Comment after line of code
    - A = B + C; /* Comment explaining this line */
  - Both methods commonly used
- Identifier naming
  - ALL_CAPS for constants
  - lowerToUpper for variables
  - Most important: MEANINGFUL NAMES!

© Janice Regan, CMPT 102, Sept. 2006

Program Style for CMPT 102 (1)

- One aspect of common style used for this course and in many computing based workplaces is to begin any class or method with a block of comments explaining what the class or method does, the variables and input and output, and program authorship and date.
- Comments are also used to explain what each block or section within the code does.
  - Provide additional information in comments
  - Explain why, explain how this helps solve the overall problem
  - Do not give and English 'translation' of the code in the block of code

© Janice Regan, CMPT 102, Sept. 2006

Program Style for CMPT 102 (2)

- When writing programs, use comments throughout for clarity
  - Use the C++ form of comments
  - Each comment must be preceded by a //
- Identifiers for Constants should include only UPPER CASE letters.
  - GRAVITATIONAL_CONSTANT
- Identifiers for variables should begin with a lower case letters
  - sumOfSquares

© Janice Regan, CMPT 102, Sept. 2006

Program Style for CMPT 102 (3)

- Identifiers for methods should begin with an upper case letter
  - CalculateMean
- Identifiers containing more than one word should have the second and each successive word capitalized
  - squareRootOfSum
- All identifiers should be meaningful names that indicate what the variable they identify represents

© Janice Regan, CMPT 102, Sept. 2006