Expressions

Expressions

- Variables and constants linked with operators
  - Arithmetic expressions
    - Uses arithmetic operators
    - Can evaluate to any value
  - Logical expressions
    - Uses relational and logical operators
    - Evaluates to 1 or 0 (true or false) only
  - Assignment expression
    - Uses assignment operators
    - Evaluates to value depending on assignment

Arithmetic Operators

- Binary operators
  - Addition: **+**
  - Subtraction: **–**
  - Division: **/**
  - Multiplication: *****
  - Modulus: **%**
- Unary operators
  - Plus: **+**
  - Minus: **–**

Examples

2*3 + 5 – 10/3
–1 + 3*25/5 – 7
distance / time
3.14* radius * radius
a * x * x + b*x + c
dividend / divisor
37 % 10

Contd.

- Suppose x and y are two integer variables, whose values are 13 and 5 respectively

x + y	18
x – y	8
x * y	65
x / y	2
x % y	3

- We will see why x / y is 2 and not 2.6 a little later

- All operators except % can be used with operands of all of the data types int, float, double, char (yes! char also! We will see what it means later)
- % can be used only with integer operands

Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ( )
  2. Unary minus :: –5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the same priority, evaluation is from left to right as they appear
- Parenthesis may be used to change the precedence of operator evaluation

Examples:
Arithmetic Expressions

a + b * c – d / e ➔ a + (b * c) – (d / e)

a * – 10 + d % e – f ➔ a * (– 10) + (d % e) – f

a – b + c + 5 ➔ (((a – b) + c) + 5)

x * y * z ➔ ((x * y) * z)

a + 2.34 + c * d * e ➔ (a + 2.34) + ((c * d) * e)

Example: Centigrade to Fahrenheit

Output

```
Enter centigrade: 36.5
36.500000 C equals 97.699997 F
```

```
#include <stdio.h>
int main()
{
    float cent, fahr;
    printf(“Enter Centigrade: “);
    scanf(“%f”,&cent);
    fahr = cent*(9.0/5.0) + 32;
    printf( “%f C equals %f F\n”, cent, fahr);
    return 0;
}
```

- **Caution:** Since floating-point values are rounded to the maximum number of significant digits permissible, the final value is an approximation of the final result. This can cause strange results sometimes in comparisons.

```
#include <stdio.h>
int main()
{
        float f1;
        printf("Enter a no: ");
        scanf("%f", &f1);
        printf("No. entered is %f\n", f1);
        if(f1 == 23.56) printf("True\n");
        else printf("False\n");
}
```

```
Enter a no: 23.56
No. entered is 23.559999
False
```

- Can be handled in many cases by using double instead of float (as it allows more number of digits)
- See the same program below, just with double. Now you get correct result

```
#include <stdio.h>
int main()
{
        double f1;
        printf("Enter a no: ");
        scanf("%lf", &f1);
        printf("No. entered is %lf\n", f1);
        if(f1 == 23.56) printf("True\n");
        else printf("False\n");
}
```

```
Enter a no: 23.56
No. entered is 23.560000
True
```

Type of Value of an Arithmetic Expression

- If all operands of an operator are integer (int variables or integer constants), the value is always integer
  - Example: 9/5 will be printed as 1, not 1.8
- But if at least one operand is real, the value is real
  - So 9/5.0 will be correctly printed as 1.8

```c
#include <stdio.h>
int main()
{
    int a, b;
    float c;
    a = 9; b = 5;
    printf ("a/b is %d\n", a/b) ;
    c = 5.0;
    printf ("a/c is %f\n", a/c) ;
    return 0;
}
```

Output

```
a/b is 1
a/c is 1.800000
```

This is a problem!!

```
int a=10, b=4, c;
float x;
c = a / b;
x = a / b;
```

The value of c will be 2

The value of x will be 2.0

But we want 2.5 to be stored in x

We will take care of this a little later

Assignment Expression

- Uses the assignment operator (=)
- General syntax:

  variable_name = expression

- Left of = is called l-value, must be a modifiable variable
- Right of = is called r-value, can be any expression
- Examples:

  velocity = 20

  b = 15; temp = 12.5

  A = A + 10

  v = u + f * t

  s = u * t + 0.5 * f * t * t

Contd.

- An assignment expression evaluates to a value same as any other expression
- Value of an assignment expression is the value assigned to the l-value
- Example: value of
  - a = 3 is 3
  - b = 2*4 – 6 is 2
  - n = 2*u + 3*v – w is whatever the arithmetic expression 2*u + 3*v – w evaluates to given the current values stored in variables u, v, w

Contd.

- Several variables can be assigned the same value using multiple assignment operators

```
a = b = c = 5;
flag1 = flag2 = 'y';
speed = flow = 0.0;
```

- Easy to understand if you remember that
  - The assignment expression has a value
  - Multiple assignment operators are right-to-left associative

Example

- Consider a = b = c = 5
  - Three assignment operators
  - Rightmost assignment expression is c=5, evaluates to value 5
  - Now you have a = b = 5
  - Rightmost assignment expression is b=5, evaluates to value 5
  - Now you have a = 5
  - Evaluates to value 5
  - So all three variables store 5, the final value the assignment expression evaluates to is 5

Types of l-value and r-value

- Usually should be the same
- If not, the type of the r-value will be internally converted to the type of the l-value, and then assigned to it
- Example:

```
double a;
a = 2*3;
```

Type of r-value is int and the value is 6

Type of l-value is double, so stores 6.0

This can cause strange problems

```
int a;
a = 2*3.2;
```

- Type of r-value is float/double and the value is 6.4
- Type of l-value is int, so internally converted to 6
- So a stores 6, not the correct result
- But an int cannot store fractional part anyway
- So just badly written program
- Be careful about the types on both sides

More Assignment Operators

- +=, -=, *=, /=, %=
- Operators for special type of assignments
- a += b is the same as a = a + b
- Same for -=, *=, /=, and %=
- Exact same rules apply for multiple assignment operators

Contd.

- Suppose x and y are two integer variables, whose values are 5 and 10 respectively.

x += y	Stores 15 in x Evaluates to 15
x –= y	Stores -5 in x Evaluates to -5
x *= y	Stores 50 in x Evaluates to 50
x /= y	Stores 0 in x Evaluates to 0

Logical Expressions

- Uses relational and logical operators in addition
- Informally, specifies a condition which can be true or false
- Evaluates to value 0 if the condition is false
- Evaluates to some non-zero value if the condition is true

Relational Operators

- Used to compare two quantities

<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Examples

10 > 20 is false, so value is 0

25 < 35.5 is true, so value is non-zero

12 > (7 + 5) is false, so value is 0

32 != 21 is true, so value is non-zero

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared

  a + b > c – d is the same as (a + b) > (c – d)

- Note: The value corresponding to true can be any non-zero value, not necessarily 1
  - Will print 1 in most cases, but should not assume it will

Logical Operators

- Logical AND (&&)
  - Evaluates to true (1) if both the operands are non-zero
- Logical OR (||)
  - Evaluates to true (1) if at least one of the operands is non-zero

X	Y	X && Y	X || Y
0	0	false	false
0	non-0	false	true
non-0	0	false	true
non-0	non-0	true	true

Contd.

- Unary negation operator (!)
  - Single operand
  - Value is 0 if operand is non-zero
  - Value is 1 if operand is 0

Examples of Logical Expressions

- (count <= 100)
- ((math+phys+chem)/3 >= 60)
- ((sex == ’M’) && (age >= 21))
- ((marks >= 80) && (marks < 90))
- ((balance > 5000) || (no_of_trans > 25))
- (! (grade == ’A’))

- a = 3 && (b = 4)
  - b = 4 is an assignment expression, evaluates to 4
  - && has higher precedence than =
  - 3 && (b = 4) evaluates to true as both operands of && are non-0, so final value of the logical expression is true
  - a = 3 && (b = 4) is an assignment expression, evaluates to 1 (true)
- Note that changing to b = 0 would have made the final value 0

Example: AND and OR

Output

```
3 0
3 AND 0 = 0, 3 OR 0 = 1
```

```
#include <stdio.h>
int main ()
{
        int i, j;
        scanf("%d%d",&i,&j);
        printf ("%d AND %d = %d, %d OR %d=%d\n", i, j, i&&j, i, j, i||j) ;
        return 0;
}
```

- Logical expressions are used in control statements
- We will see more examples of logical expressions when we study control statements next

More on Arithmetic Expressions

Recall the earlier problem

```
int a=10, b=4, c;
float x;
c = a / b;
x = a / b;
```

The value of c will be 2

The value of x will be 2.0

But we want 2.5 to be stored in x

Solution: Typecasting

- Changing the type of a variable during its use
- General form

  (type_name) variable_name

- Example

  x = ((float) a)/ b;

Now x will store 2.5 (type of a is considered to be float for this operation only, now it is a mixed-mode expression, so real values are generated)

- Not everything can be typecast to anything
  - float/double should not be typecast to int (as an int cannot store everything a float/double can store)
  - int should not be typecast to char (same reason)
- General rule: make sure the final type can store any value of the initial type

Example: Finding Average of 2 Integers

Wrong program

```
int a, b;
float avg;
scanf("%d%d", &a, &b);
avg = (a + b)/2;
printf("%f\n", avg);
```

average-1.c

```
int a, b;
float avg;
scanf("%d%d", &a, &b);
avg = ((float) (a + b))/2;
printf("%f\n", avg);
```

Correct programs

```
int a, b;
float avg;
scanf("%d%d", &a, &b);
avg = (a + b)/2.0;
printf("%f\n", avg);
```

average-2.c

More Operators: Increment (++) and Decrement (--)

- Both of these are unary operators; they operate on a single operand
- The increment operator causes its operand to be increased by 1
  - Example: a++, ++count
- The decrement operator causes its operand to be decreased by 1.
  - Example: i--, --distance

Pre-increment versus post-increment

- Operator written before the operand (++i, --i))
  - Called pre-increment operator (also sometimes called prefix ++ and prefix --)
  - Operand will be altered in value before it is utilized in the program
- Operator written after the operand (i++, i--)
  - Called post-increment operator (also sometimes called postfix ++ and postfix --)
  - Operand will be altered in value after it is utilized in the program

Examples

<u>Initial values :: a = 10; b = 20;</u>

x = 50 + ++a;	a = 11, x = 61
x = 50 + a++;	x = 60, a = 11
x = a++ + --b;	b = 19, x = 29, a = 11
x = a++ – ++a;	??

Called side effects (while calculating some values, something else gets changed)

Precedence among different operators (there are many other operators in C, some of which we will see later)

<table>
<tr><th>Operator Class</th><th>Operators</th><th>Associativity</th></tr>
<tr><td>Unary</td><td>postfix++, --</td><td>Left to Right</td></tr>
<tr><td>Unary</td><td>prefix ++, --
— ! &</td><td>Right to Left</td></tr>
<tr><td>Binary</td><td>* / %</td><td>Left to Right</td></tr>
<tr><td>Binary</td><td>+ —</td><td>Left to Right</td></tr>
<tr><td>Binary</td><td>< <= > >=</td><td>Left to Right</td></tr>
<tr><td>Binary</td><td>== !=</td><td>Left to Right</td></tr>
<tr><td>Binary</td><td>&&</td><td>Left to Right</td></tr>
<tr><td>Binary</td><td>||</td><td>Left to Right</td></tr>
<tr><td>Assignment</td><td>= += —=
*= /= %=</td><td>Right to Left</td></tr>
</table>

Doing More Complex Mathematical Operations

- C provides some mathematical functions to use
  - perform common mathematical calculations
  - Must include a special header file

    `#include <math.h>`
- Example
  - `printf ("%f", sqrt(900.0));`
    - Calls function `sqrt`, which returns the square root of its argument
- Return values of math functions are of type `double`
- Arguments may be constants, variables, or expressions
- Similar to functions you have seen in school maths

Math Library Functions

double acos(double x)	– Compute arc cosine of x.
double asin(double x)	– Compute arc sine of x.
double atan(double x)	– Compute arc tangent of x.
double atan2(double y, double x)	– Compute arc tangent of y/x.
double cos(double x)	– Compute cosine of angle in radians.
double cosh(double x)	– Compute the hyperbolic cosine of x.
double sin(double x)	– Compute sine of angle in radians.
double sinh(double x)	– Compute the hyperbolic sine of x.
double tan(double x)	– Compute tangent of angle in radians.
double tanh(double x)	– Compute the hyperbolic tangent of x.

Math Library Functions

double ceil(double x)	– Get smallest integral value that exceeds x.
double floor(double x)	– Get largest integral value less than x.
double exp(double x)	– Compute exponential of x.
double fabs (double x)	– Compute absolute value of x.
double log(double x)	– Compute log to the base e of x.
double log10 (double x)	– Compute log to the base 10 of x.
double pow (double x, double y)	– Compute x raised to the power y.
double sqrt(double x)	– Compute the square root of x.

Computing distance between two points

```
#include <stdio.h>
#include <math.h>
int main()
{
        int x1, y1, x2, y2;
        double dist;
        printf(“Enter coordinates of first point: “);
        scanf(“%d%d”, &x1, &y1);
        printf(“Enter coordinates of second point: “);
        scanf(“%d%d”, &x2, &y2);
        dist = sqrt(pow(x1 – x2, 2) + pow(y1 – y2, 2));
        printf(“Distance = %lf\n”, dist);
        return 0;
}
```

Output

```
Enter coordinates of first point: 3  4
Enter coordinates of second point: 2  7
Distance = 3.162278
```

Practice Problems

1. Read in three integers and print their average
2. Read in four integers a, b, c, d. Compute and print the value of the expression

   a+b/c/d*10*5-b+20*d/c

   - Explain to yourself the value printed based on precedence of operators taught
   - Repeat by putting parenthesis around different parts (you choose) and first do by hand what should be printed, and then run the program to verify if you got it right
   - Repeat similar thing for the expression a&&b||c&&d>a||c<=b
3. Read in the coordinates (real numbers) of three points in 2-d plane, and print the area of the triangle formed by them
4. Read in the principal amount P, interest rate I, and number of years N, and print the compound interest (compounded annually) earned by P after N years