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

- 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

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 1, not 1.8
  - Example:

```
int a=9, b=5;
printf("%d", a/b)
```

will print 1 and not 1.8

- If at least one operand is real, the value is real
  - **Caution:** Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result
  - Example: 1/ 3.0 * 3.0 may have the value 0.99999 and not 1.0

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

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

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

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

Type Casting

- Convert a variable from one data type to another data type
- Cast operator: `(type_name) expression`

Cont.

What is the output?

```
#include <stdio.h>
int main() {
    int a = 25, b = 10;
    float result;
    result = a/b;
    printf("The result is %f\n", result);
    return 0;
}
```

Cont.

```
#include <stdio.h>
int main() {
    int a = 25, b = 10;
    float result;
    result = a/b;
    printf("The result is %f\n", result);
    return 0;
}
```

What is the output?

The result is 2.000000

Cont. (Apply Type Cast)

What is the output?

```
#include <stdio.h>
int main() {
    int a = 25, b = 10;
    float result;
    result = (float) a/b;
    printf("The result is %f\n", result);
    return 0;
}
```

Cont. (Apply Type Cast)

```
#include <stdio.h>
int main() {
    int a = 25, b = 10;
    float result;
    result = (float) a/b;
    printf("The result is %f\n", result);
    return 0;
}
```

What is the output?

The result is 2.500000

Implicit casting

- Compiler can implicitly cast the values in an expression based on the declared variable type

- Implicit casting sequence priority:
  - `int -> long -> float -> double`

Implicit Casting

int a = 10;

float b = 2.0

- The implicit type for (a*b) is float

Rule: The final type of an expression is the highest priority type among all the variable types in the expression

Cont.

What is the output?

```
#include <stdio.h>
int main() {
    int a = 25;
    float b = 10.0;
    float result;
    result = a/b;
    printf(“The result is %f\n”, result);
    return 0;
}
```

Cont.

```
#include <stdio.h>
int main() {
    int a = 25;
    float b = 10.0;
    float result;
    result = a/b;
    printf("The result is %f\n", result);
    return 0;
}
```

What is the output?

The result is 2.500000

Logical Expressions

- Uses relational and logical operators in addition
- Informally, specifies a condition which can be true or false
- Evaluates to value 0 or 1
  - 0 implies the condition is false
  - 1 implies 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

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’))

Examples

10 > 20	is false, so value is 0
25 < 35.5	is true, so value is 1
12 > (7 + 5)	is false, so value is 0
32 != 21	is true, so value is 1

- 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)

Logical Operators

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

X	Y	X && Y	X || Y
0	0	0	0
0	non-0	0	non-0
non-0	0	0	non-0
non-0	non-0	non-0	non-0

Contd

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

Example

- (4 > 3) && (100 != 200)
  - 4 > 3 is true, so value 1
  - 100 != 200 is true so value 1
  - Both operands 1 for &&, so final value 1
- (!10) && (10 + 20 != 200)
  - 10 is non-0, so value !10 is 0
  - 10 + 20 != 200 is true so value 1
  - Both operands NOT 1 for &&, so final value 0
- (!10) || (10 + 20 != 200)
  - Same as above, but at least one value non-0, so final value 1

A Special Operator: AddressOf (&)

- Remember that each variable is stored at a location with an unique address
- Putting & before a variable name gives the address of the variable (where it is stored, not the value)
- Can be put before any variable (with no blank in between)

```
int a =10;
printf("Value of a is %d, and address of a is %d\n", a, &a);
```

Statements in a C program

- Parts of C program that tell the computer what to do
- Different types
  - Declaration statements
    - Declares variables etc.
  - Assignment statement
    - Assignment expression, followed by a ;
  - Control statements
    - For branching and looping, like if-else, for, while, do-while (to be seen later)
  - Input/Output
    - Read/print, like printf/scanf

Example

Declaration statement

```
int a, b, larger;
scanf("%d %d", &a, &b);
larger = b;
if (a > b)
   larger = a;
printf("Larger number is %d\n", larger);
```

Control statement

Assignment statement

Input/Output statement