Foundations of C Programming (Structured Programming) - Expressions

Outline

- Arithmetic expressions
- Relational expressions (conditions)
- Logical expressions (decisions)
- Special operators
- Macro definition

Arithmetic Expressions

- Arithmetic operators
  - Binary operators
    - +, -, *: for all integer and float
      - E.g, a + b; 10 – 4, -5, 2.0 * 10
    - /
      - integer: give the int quotient. E.g., 10 / 3 = 3
      - Float: give the float quotient. E.g., 10.0 / 3 = 3.333333
    - % (modulus)
      - integer: give the remainder. E.g., 5 % 3 = 2
      - not applicable to float

Arithmetic Expressions

- An expression is a sequence of operands (constants or variables) and operators that reduces to a single value
  - E.g., a*b-c, (m+n)*(x+y), 6*2/3
- An expression is evaluated from left to right using the rule of precedence of operators
  - Precedence
    - Highest priority: ( )
    - High priority: * / %
    - Low priority: + -
  - What are the results of x and y
    - x = 9-12/(3+3*2-1);
    - y = 9-12/3+3*2-1;

Class Exercises

Compare the results of y

```
int i = 5, j = 1;
float x = 1.0, y;
y = x / i;
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = j / i;
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = (float)j / i;
```

Class Exercises

Compare the results of y

```
int i = 5, j = 1;
float x = 1.0, y;
y = x / i;              /* y = 1.0/5 = 0.2  */
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = j / i;              /* y = 1/5 = 0.0 */
```

```
int i = 5, j = 1;
float x = 1.0, y;
y = (float)j / i;       /* y = 1.0/5 = 0.2  */
```

Relational Expressions (Conditions)

- Expressions with relational operators: <, <=, >, >=, ==, !=
- Allows you to compare variables and values
- The value of a relational expression is either 0 (false) or 1 (true)
- 4 < 5: true (1); 4 > 5: false (0); 4!=5: true (1).
- x > 10 : unknown, depending on the value of x

Relational Expressions

Operator	Description	Example
>	greater than	5 > 4
>=	greater than or equal to	mark >= score
<	less than	height < 75
<=	less than or equal to	height <= input
==	equal to	score == mark
!=	not equal to	5 != 4

Logical Expressions (Decisions)

- Comprise relational expressions (conditions) and logical operators
  - Logical operators
    - && (two ampersands): means *and*.
      - *C1* && *C2* is true only when both *C1* and *C2* are true
    - || (two vertical bars): means *or*.
      - *C1* || *C2* is true when at least one of *C1* and *C2* is true
    - ! (an exclamation point): means *not*.
      - *!C*: true if *C* is false, false if *C* is true

Logical Expressions (Decisions)

- Logical operations allows you to verify more than one condition
- A logical expression is also called a decision
  - E.g., (x >= 0.0) && (x <= 1.0)
- When there is no logical operator in a decision, it is a condition.
- In this lecture, we will not differentiate the concepts of condition and decision strictly.

Logical Expressions

- What are the values of these Boolean variables

```
int x = 5, y = 10,  z = 20;
_Bool   P = (x < y);  //P = 1
_Bool   Q = (y > z);   //Q = 0
_Bool   R = 1;
_Bool   S1 = P && Q;
_Bool   S2 = ((x<y) && (y<z));
_Bool   T = !Q || R;
_Bool   W = -10;
```

Tip: relational expressions and logical expressions are usually used directly without need to assign to Boolean variables.

Class Exercise

- Translate the following English questions into C decisions.
  - The height is not equal to zero
  - The temperature is greater than 32 and less than 212
  - The absolute value of *pos* is greater than 5

Class Exercise

- Translate the following English questions into C decisions.
  - The height is not equal to zero (*height* != 0)
  - The temperature is greater than 32 and less than 212 (*temperature* > 32 && *temperature* < 212)
  - The absolute value of *pos* is greater than 5

    (*pos* > 5 || *pos* < -5)

After class

- Think about the following interesting question:

Are the following three expressions equivalent (assume x, y are of short int type, and z = x – y, u = -x, v = -y)?

- `x < y`
- `z < 0`
- `v > u`

Assignment Operators

- Assignment operators
  - =
    - E.g., a = 10; c = 'c';
  - *op*=
    - *op* can be any of +, -, *, /, and %
    - a *op*= b is equivalent to a = a *op* b
    - E.g.,
      - a += 10 is equivalent to a = a + 10
      - a /= b + c is equivalent to a = a / (b + c)

Examples

- Example 1

```
x = ( y = 3 ) +1;     /* y is assigned 3 */
                      /* the value of (y = 3) is 3 */
                      /* x is assigned 4 */
```

- Example 2

```
x = 5;                /* x is assigned 5 */
y = 3;                /* y is assigned 3 */
x += y + 1;           /* x = x + (y+1) */
                      // x = 9
```

Class Exercise

- Can you explain these expressions
  - x = (y = 5) + 3
  - x = y = 5 + 3
  - x == (y =5)

Arithmetic Expressions

- Arithmetic operators
  - unary operators
    - -
      - E.g., -5, x = -y
    - ++ (increment), -- (decrement)
      - Has only one operand. Only applicable to integers
      - Unary operators that require variable as their operand
      - E.g.,
        - » x++, ++x , x--, --x

Rules for ++ and --

- When x++ appears as a single independent statement, e.g., `x++;` it is equivalent to `x = x + 1;` similarly,
  - ++x ; is equivalent to x = x + 1;
  - x--; is equivalent to x = x – 1;
  - --x; is equivalent to x = x – 1;
- When ++, or -- is used in an expression with other operators or a statement, some rules must be followed.
  - postfix ++ (or --) (e.g., x++): the original value of the variable is used first in that expression or statement and then the variable is incremented (or decremented) by one
  - prefix ++ (or --) (e.g., ++x): the variable is incremented (or decremented) by one first and result value of the variable is then used in that expression or statement.

An Example

```
m = 5;
n = 10;
m++;  // m = 6
y = ++m;  // m = 7, y = 7
x = n++;  // x = 10, n = 11
printf("%d %d %d %d\n", m, n, y++, ++x);
// 7 11 7 11

printf("%d %d\n", y, x);
//8 11
```

What is the output?

Use of ++ and --

- It is not encouraged to use ++ and - - on a variable which appears at multiple places
  - It makes program hard to read, and
  - different compilers may give different results.

```
#include<stdio.h>
int main(void) {
    int a = 9;
    int b;
    b = ++a + a--;
    printf("%d %d ", a, b);
    b = a-- + ++a;
    printf("%d %d\n", a, b);
    return 0;
}
```

Results on different compilers:

```
Windows, Pelles C:  9 20 9 18
Windows, VC++:      9 20 9 20
Linux, gcc:         9 19 9 18
Linux, g++:         9 19 9 18
OpenBSD, gcc:       9 20 9 20
OpenBSD, g++:       9 20 9 20
```

Attention:
It is very unreadable and unreliable to include a variable with ++, -- more than once in an expression.
Simply use i++ as the simplification of $i = i + 1$;

Conditional Operators

- Format
  - exp1 ? exp2 : exp3
  - If exp1 is true,
    - Value of exp1 ? exp2 : exp3 is exp2 (exp3 is not evaluated)
  - If exp1 is false,
    - Value of exp1 ? exp2 : exp3 is exp3 (exp2 is not evaluated)

Examples

Example 1

```
x = 4;
y = 5;
z = (x > y) ? x : y;
```

Example 2

```
x = 5;
y = 4;
z = (x > y) ? x : y;
```

What is the value of z in these two examples?

Bitwise Operators

- Work on binary system of all integer types
  - Operator Meaning

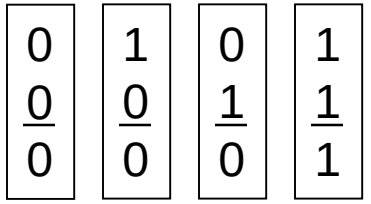
Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise Exclusive OR
~	bitwise complement
<<	shift left
>>	shift right

Bitwise Operators

```
x = 0xFFF0;
y = 0x002F;
```

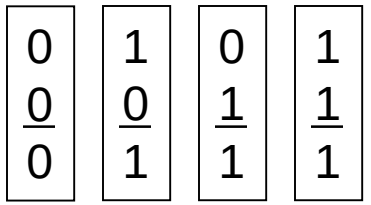
```
x & y = 0x0020

x:    1111 1111 1111 0000
y:    0000 0000 0010 1111
      ---------------------
      0000 0000 0010 0000
```




```
x | y = 0xFFFF

x:    1111 1111 1111 0000
y:    0000 0000 0010 1111
      ---------------------
      1111 1111 1111 1111
```

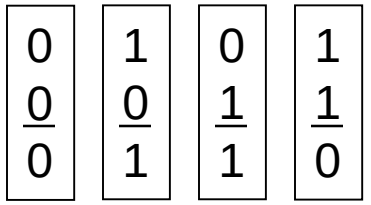


Bitwise Operators

```
x = 0xFFF0;
y = 0x002F;
```

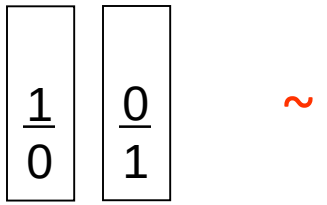
```
x ^ y = 0xFFDF

x:    1111 1111 1111 0000
y:    0000 0000 0010 1111
      ---------------------
      1111 1111 1101 1111
```




```
~y = 0xFFD0

y:    0000 0000 0010 1111
      ---------------------
      1111 1111 1101 0000
```



Shift, Multiplication and Division

- 14: 0000 1110 ($2^3+2^2+2^1$)
  - 14<<1 (shift one bit left: 0001 1100) ($2^4+2^3+2^2 = 28$)
  - 14>>1 (shift one bit right: 0000 0111) ($2^2+2^1+2^0 = 7$)
- Multiplying 2 can be replaced by shifting 1 bit to the left

```
n = 10
printf("%d = %d" , n*2, n<<1);
printf("%d = %d", n*4, n<<2);
```

- Division by 2 can be replaced by shifting 1 bit to the right

```
n = 10
printf("%d = %d" , n/2, n>>1);
printf("%d = %d", n/4, n>>2);
```

- Multiplication and division are often slower than shift.

Comma Operator

- An expression can be composed of multiple subexpressions separated by commas.
  - Subexpressions are evaluated left to right.
  - The entire expression evaluates to the value of the rightmost subexpression.

```
x = (a++, b++);
```

Evaluation steps:

1. a is incremented
2. b is assigned to x
3. b is incremented

Attention:
This sentence is not readable. It is better to rewrite it as:

```
a++;
x = b;
b++;
```

Operator Precedence

Operator	Precedence level
( )	1
~, ++, --, unary - , !	2
*, /, %	3
+, -	4
<<, >>	5
<, <=, >, >=	6
==, !=	7
&	8
^	9
|	10
&&	11
||	12
=, +=, -=, etc.	14
,	15

Class Exercise

```
#include <stdio.h>
int main (){
    int temp1, temp2;
    int x = 20, y1 = 30, y2 = 30, z = 40;

    temp1 = x * x /++y1 + z / y1;
    temp2 = x * x /y2++ + z / y2;
     printf("temp1 = %d; temp2 = %d; y1 = %d; y2 = %d\n",
          temp1, temp2, y1, y2);
   return 0;
}
```

What is the output of the program?

Use of Macro

- A constant may be used in several places in a program.
- To make the program readable and easy to modify, a macro can be defined

```
#include <stdio.h>
#define PI 3.14
int main(){
  float r, c, a;
  r = 13.14;
  c = 2 * PI * r;
  a = PI * r * r;
  printf("c=%f, a=%f", c, a);
  return 0;
}
```

```
#include <stdio.h>
int main(){
  float r, c, a;
  r = 13.14;
  c = 2 * 3.14 * r;
  a = 3.14 * r * r;
  printf("c=%f, a=%f\n", c, a);
    return 0;
}
```

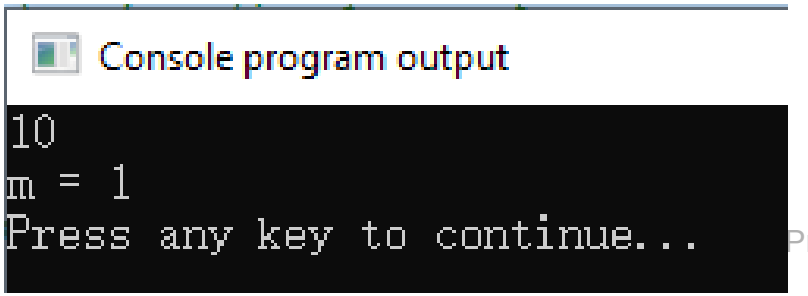
Compare these two programs:
Which one is convenient to modify if we want to use 3.14159 as the PI value?

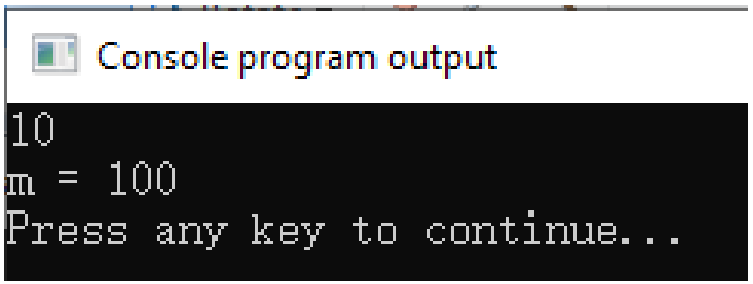
Use of Macro

- Macro can also use variable, but parentheses must be added

```
#include <stdio.h>
#define SQUARE(X) (X*X)
int main(){
  int m, n;
  scanf("%d", &n);
  m = 100/SQUARE(n);
  printf("m = %d\n", m);
  return 0;
}
```

```
#include <stdio.h>
#define SQUARE(X) X*X
int main(){
  int m, n;
  scanf("%d", &n);
  m = 100/SQUARE(n);
  printf("m = %d\n", m);
  return 0;
}
```





Summary

- Arithmetic expression
- Logical expression
- Bitwise operation
- Precedence of operations