Introduction to Computer Engineering
0702111

2. Expressions

YACOUB SABATIN
MUNTASER ABULAFI
OMAR QARAEEN

1

Introduction

- A typical programming scenario:
  - ***Read*** **input** data using ***scanf*** function,
  - ***Perform*** *some calculations involving:*
    - *Evaluating expressions;*
    - *Storing results in certain variables by executing assignment statements*, and
  - **Output result** using ***printf*** function;
- Read & Output➔ Covered previously;
- **Will focus on the calculation point** by **combining constants** and **variables** with operators (**+,-,/, *, %**) to form **expressions**;

2

- C language has rich and powerful set of operators:
  - **Arithmetic Operators:**
    - ***( )*** ***Brackets;***
    - ***++*** ***Increment;***
    - ***--*** ***Decrements;***
    - ******* ***Multiplication;***
    - ***/*** ***Division;***
    - ***+*** ***Addition (plus);***
    - ***-*** ***Subtraction (minus);***
    - ***%*** ***Remainder (Mod);***
    - ***=*** ***Assignment.***

3

  - **Relational Operators:**
    - *>* *Greater than;*
    - *>=* *Greater than or equal to;*
    - *<* *Less than;*
    - *<=* *Less than or equal to;*
    - *==* *Equal to;*
    - *!=* *Not equal to.*
  - **Logical/Boolean Operators:**
    - && *AND;*
    - || *OR;*
    - ! NOT.

4

- The operators **precedence** (**priority**):
  1. Example: **a=10.0 + 2.0 * 5.0 - 6.0 / 2.0**;
     What is the answer?! (**Answer=17**)
     ➔ ***Multiplication*** *&* ***Division*** *parts will be* ***evaluated first*** *and then the* ***addition*** *and* ***subtraction*** *parts….*
  - ***Note:*** *To avoid confusion use brackets*.
    The following are two different calculations:
    → a=10.0 + (2.0 * 5.0) - (6.0 / 2.0)
    → a=(10.0 + 2.0) * (5.0 - 6.0) / 2.0
  - **Level 1: ();**
  - **Level 2: *, /, %;**
  - **Level 3: +, -.**

5

- **Evaluate the expression: 5 + 3 * 2 + 12 / 6?**

  5 + 6 + 2

  ➔ **13**;
- Evaluate:
  - **7 / 3 = 2**
  - **7 % 3 = 1**
  - **20.0 / 3 * 20 / 3 = 44.44443**
  - **20 / 3 * 20.0 / 3 = 40.0**
  - **20.0 / 3 + 20 / 3 = 12.666667**
- Evaluate:
  - **- 4 + 23 * (2 / 3)**
  - **3 * 17 – 189.0 / 18.0**

6

Example: Universal Product Code (**UPC)** is a bar code for the product, to be able to identify it and know its price. It consists of **12 digits** (**1**, **5**, **5**, and **1**) the **1st** (**d**) identifies the type of the item, the next **5** identify the **manufacturer**, the next **5** identify the **product**, then the **final digit** is 'check digit' which can tell if the previous **11 digits** were read correctly or not, so to re-read them by the scanner once again if there is something wrong. The formula to calculate this digit:

- ***first_sum = d + i2 + i4 + j1 + j3 + j5***
- ***second_sum = i1 + i3 + i5 + j2 + j4***
- ***total = 3 * first_sum + second_sum***
- ***check digit=9 - ((total-1) % 10)***

- The following example implements this algorithm:

7

```
/* Computes a Universal Product Bar Code check
   digit */
#include <stdio.h>
main()
{
  int d, i1, i2, i3, i4, i5, j1, j2, j3, j4,
     j5, first_sum, second_sum, total;
  printf("Enter the first (single) digit: ");
  scanf("%1d", &d);
  printf("Enter first group of five digits: ");
  scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4,
     &i5);
```

8 >>

```
  printf("Enter second group of five digits: ");
  scanf("%1d%1d%1d%1d%1d", &j1, &j2, &j3, &j4,
     &j5);

  first_sum = d + i2 + i4 + j1 + j3 + j5;
  second_sum = i1 + i3 + i5 + j2 + j4;
  total = 3 * first_sum + second_sum;

  printf("Check digit: %d\n", 9-( (total - 1) %
     10 ));
  return 0;
}
```

Try: 3, 1, 2, 3, 4, 5, 0, 6, 7, 8, 9:
Check Digit = ??

9

Assignment Statements

- Once you've declared a variable you can use it, ***but not until it*** *has been declared* - attempts to use a variable that has not been defined will cause a compiler error!
- Using a variable means storing something in it
- You can store a value in a variable by using assignment statements of the form:
  Variable *name (v) = value or an expression(e);*

- Example:
  a=10; */*stores the value 10 in the int variable a */*

10

– Expression, ***e,*** is evaluated and its value is assigned to ***v,*** (**overwriting** the previously stored value; if any)
– The assignment produces a result which is the value of ***v*** after assignment
– **e** can be:
  • *Constant* *i = **5**;* /* i is 5 */
  • *variable* *j = **i**;* /* j is 5 */
  • *expression* *k = **10 * i + j**;* /* k is 55 */

- Generally, ***v*** is an expression that have an **lvalue** (*a location in the machine's memory).*
  So an assignment *i + j = 0;* is illegal, because *i + j* has no **lvalue** or no place to store the Zero!

11

- BE CAREFUL WITH ARITHMATIC!!!
  What is the answer to this simple calculation? a=10/3
- The answer depends upon the data type of the division sides (see the next example).
  →If *a* declared as type ***int*** the answer will be 3;
  →If *a* is of type ***float*** then the answer will be 3.000000
- Two points to note from the above calculation:

1. C ignores fractions when doing **integer division**!
2. When doing float calculations integers will be **converted** into float.

→ We will see later how C handles type conversions.

12

2 examples:

```
float a=10/3;
printf("The answer is: %f",a);
```

Would print:

```
The answer is: 3.000000
```

```
float a=10/3.0;
printf("The answer is: %f",a);
```

Would print:

```
The answer is: 3.333333
```

13

Compound Assignments

- *i = i + 3; A*n assignment statement that **adds 3** to the **previous value stored in *i***
- **Compound assignment operators:**
  (+=, -=, *=, /=, %=)
  – Allows to shorten statements: (*i = i + 3;*) to (*i += 3;)*
  ➔ ***+=*** *adds value of right* ***operand*** *to* ***variable*** *to left*
  – **Other compound assignments:**
    - ***v += e*** ➔ Adds ***e*** to ***v***, storing result in **v**;
    - ***v -= e*** ➔ *S*ubtracts **e** from **v**, storing result in **v**
    - ***v *= e*** ➔ Multiplies **v** by **e**, storing result in **v**
    - ***v /= e*** ➔ Divides **v** by **e**, storing result in **v**
    - ***v %= e*** ➔ Computes the remainder when **v** is divided by **e**, storing result in **v;**

14

Example

```
int a, b, c, d, e;
a = 14;
b = a + 1;              //Now, b=15
b *= b;                 //b=225
c = b / 10 - 2;         //c=20
d = a + 15 - c;         //Now, d=9
c += d;                 //c=29
e = c + d;              //e=38
e /= 2;                 //e=19
printf("%d%d%d%d%1.0f DNA is a Main-belt
Asteroid discovered by C. Juels & P. Holvorcem
at Fountain Hills.\n",a-9,b/45,c-24,d-4,e/4.0);
```

Asteroid: *The belt is the region of the Solar System located roughly between the orbits of the planets Mars and Jupiter. It is occupied by numerous irregularly shaped bodies called asteroids or minor planets.*

15

What's the output??

- **What are the final values of X, Y, & Z:**

```
X = Y = Z = 10;/* The = operator is right
  associative, so this assignment is equivalent to:
  X=(Y=(Z=10)); */
   X += 15;
   Y -= 15;
   Z %= 3;
```

- `X = ?, Y = ?, Z = ?`
- **Watch out an unexpected results in chained assignments and type conversion:**

```
  int i;
  float f;
  f=i=33.3;
```

- `i` **is assigned the value** `33` **(because:** `int`**), then** `f` **is assigned** `33.0` **(**`not 33.3`**, as you might think)!**

16

Increment & Decrement

- Another way to increment variables is using:
  - **++ (increment)** ***(add 1 to operand)*** and
  - **-- (decrement)** ***(subtract 1 from operand)***
- **++** and **--** can be used as:
  - ***prefix*** **operators** (**++i** and **--I**) ➔ **Variable is changed *before* the expression is evaluated.**
  - ***postfix*** **operators** (**i++** and **i--**) ➔ **Variable is not changed until *after* it is used in the expression.**

Simple Assignment	Compound	Increment
➔ i = i + 1	i += 1	++i
➔ j = j - 1	j - = 1	--j

17

- **That is:**
  - A **prefix ++i** , the variable is incremented (or decremented) and then used:
    - ➔ *int i = 10;*
      int j = ++i;

      i is incremented to 11, & then j is set to 11 (the value of i)
  - A **postfix i++** , the variable is used and then incremented. *(increments i, but returns the prior, non-incremented value)*
    - ***int i = 10;***
    - ***int j = i++;***

      j is set to 10 (the value of i) and then i is incremented to 11.

18

- The expressions:
  - *k = 2 * ++i* ➔
    - Add one to *i*,
    - Store the result back in *i*,
    - Multiply *i* by 2, and
    - Store that result in *k*.
  - *k = 2 * i++* ➔
    - Take *i*'s old value and multiply it by *2*,
    - Increment *i*, and
    - Store the result of the multiplication in *k*.
- The -- operator works in the same way.

19

Example 1:

```
i = 1;
printf ("i is %d \n", ++i); /* prints➔ i is 2 */
printf ("i is %d \n", i);   /* prints➔ i is 2 */
```

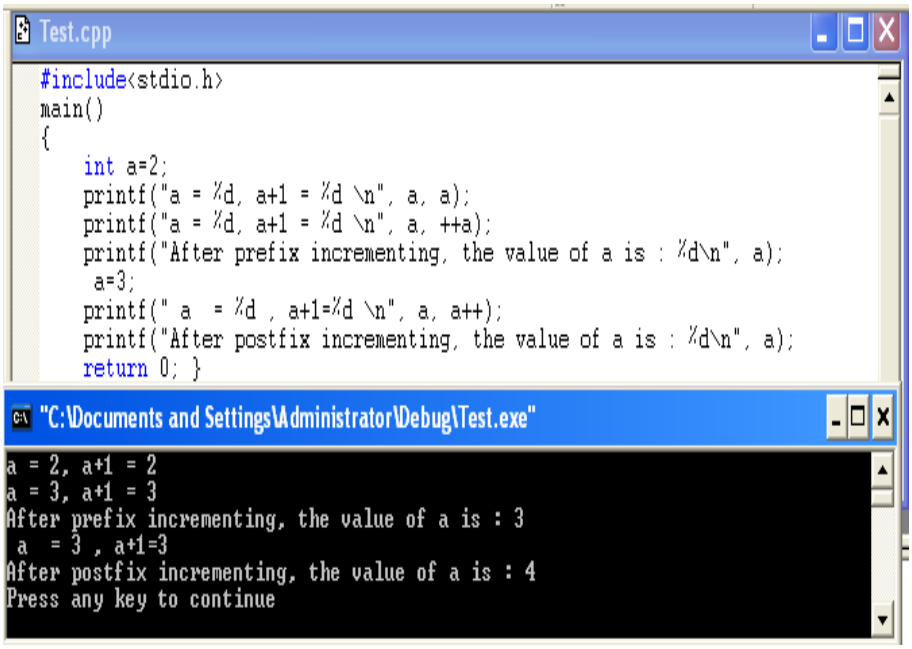
Example 2:

```
i = 1;
printf ("i is %d \n", i++); /* ➔prints i is 1 */
printf ("i is %d \n", i);   /* ➔prints i is 2 */
```

Example 3:

```
a= 3;
printf("a=%d, a+1=%d \n", a, ++a);
/* incremented before evaluation, then passed */
/* will print out ➔ a=4,  a+1=4 */
```

20



21

- *If the increment operator comes after the named variable, then the value of the statement is calculated after the increment occurs:*
  - *So for the statement:* **a= 3;**

    `printf("a=%d, a+1=%d \n", a, a++);`

    - would display `a=3  a+1=3`
  - Now, a will be set to 4, Try

    `printf("a=%d", a);`

    - The result will be: `a=4`

22

Operator Precedence & Associativity

- If Expression contains more than One operator and NO parentheses are used to clarify expression
- Then can USE operator Precedence & Associatively to clarify any ambiguity!
- Example:

  Does x+y*k equals x+(y*k) or (x+y)*k ?

23

- **Recalling Precedence**:
  - Highest - + (unary)
  - * / %
  - Lowest + - (binary)
- **Unary operation:** An operation with only **one operand:**
  - **Operand** ➔ One of the inputs (such as 3, 6) of an **operator** (such as +);

  Ex.:

  3 + 6 = 9

  i.e. an operation with a single input.

24

- In C language, the following operators are **unary**:
  - **Increment / Decrement :** `++x, x++, --x, x--`
    (Doesn't work on expressions; needs l-value)
  - **Address:** `&x`
  - **Indirection:** `*x`
  - **Positive / Negative:** `+x, -x`
  - **One's complement:** `~x`
  - **Logical negation:** `!x`
    (ex. printf("%d",!!3); would print 1)
  - **Sizeof operator:**
    `sizeof x, sizeof(type-name)`
  - **Cast:** `(type-name) cast-expression`
    Ex. `printf("%f",(float)(10/3));` ➔ `3.000000`

25 **Ex.** `printf("%f",(float)10/3);` ➔ `3.333333`

Associativity

- If expression contains operators of the same precedence, then we have to use Associativity
- **Left Associativity:** Operator groups from **L** to **R**
- Ex:
  - Binary Arithmetic Operators **(+ - * / %)**:
    i – j + k ➔ (i – j) + k
    I * j / k ➔ (I * j) / k
- **Right Associativity:** Operator groups from **R** to **L**
  Ex:
  - Unary Arithmatic Operators **(- +)**
    -+i ➔ -(+i)

26

Notes

- In compound assignment operators:
  - **i+=j;** equivalent to **i=i+j;**
  - **i=+j;** compiles **BUT** equivalent to **i=(+j)** which merely copies the value of j into i!
- Compound assignment operators have the same properties as the = operator (associativity)
- **i+=j+=k;** means **i+=(j+=k);**

27

- **Equality & Assignment:**

```
x = 5;          /* sets value of x to 5 */
if (x == 5)     /* returns true/false */
x += 3;           /* x is now is 8   */
if (x != 5)
{
   Printf ("x is not 5 \n");
}
x *= 2;           /* x is now 16   */
```

28

Examples

1. What is the value of `x` in each of the following:
   1. `x = 3 + 4 % 6 / 2 + 5`?
   2. `x = y = 4;  x *=3 + y;`
2. What is the output in each case:
   1. `printf("%d", 3 * 17 – 189/18);`
   2. `printf("%f",193/19/pow(3.0, 2.0));`
   3. `printf("%d", -4 + 23 *( 2 / 3));`
   4. `printf("%f", 3 * 17 – 189.0 /18.0);`
   5. `printf("%f",193/19.0/pow(3.0, 2.0));`
   6. `printf("%d", 5 % 2 + 14 / 3 - 8);`

29