Chapter Three

Arithmetic Expressions and Assignment Statements

1

Expressions

- An *expression* consists of *operators* and *operands*
- An *operator* indicates a *computational operation* such as *addition* or *subtraction*
- An *operand* represents a *data value* such as an *integer number* or a *real number*

2 * 3.14159

2

Operands

- An *operand* can be
  - a *constant*
  - a *variable*
  - a *function call*
  - an *expression*
  - an *expression in parentheses*

$$- b + \sqrt{b^2 - 4 * a * c}$$

- b + sqrt((b * b) – (4 * a * c))

3

Data Types

- *Data values* are classified into different *types*
- Each data type has two properties: a set of *values* and a set of *operations*
- The type *int* denotes the set of *integers* and the set of operations {+, -, *, /, %}
- The type *double* denotes the set of *double-precision floating point numbers* and the set of operations {+, -, *, /}

4

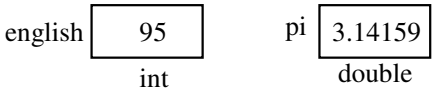
Constants

- Integer numbers
  - 2, 0, -2
- Double-precision floating point numbers
  - 2.0, 0.0, -2.0, 2.345E+8, 2.345E-8

5

Variables

- Variables denote *memory locations* that store *data values* used during the computation
- Each variable has three attributes: an *identifier*, a *value*, and a *type*



6

Identifiers

- An *identifier* is a string of characters consisting of *letters*, *digits*, and *underscores* that does not begin with a digit
- Letters in an identifier is *case-sensitive*
- Use identifiers of 31 characters or fewer to ensure *portability*
- Identifiers are used to name *variables*, *functions* and *types*

7

Variable Declarations

- All variables must be *declared before* they are *used*
- Variable declarations are *usually* written *before any statements* in the function
- A variable declaration announces the *attributes* of variables; it consists of a *type name*, a list of variables, and a semicolon

```
int  english, chinese, total;
```

8

An Example

```
main( )
{
   int  english, chinese, total;

   english = 95;
   chinese = 98;
   total = english + chinese;
}
```

Variable declarations

Statements

9

Keywords

- A *keyword* is a word that is *reserved* for special purposes
- A keyword cannot be used as an *identifier*
- Examples of keywords are int and double

10

Assignment Statements

- An *assignment* statement
  variable = *expression*
  evaluates the value of *expression* and *assigns* that value to *variable*
- The operator = is called the *assignment operator*
- Before any assignment to a variable, the value of the variable is *undefined*

11

An Example

```
main( )
{
   int  english, chinese, total;

   english = 95;
   chinese = 98;
   total = english + chinese;
}
```

english	chinese	total
95		
95	98	
95	98	193

12

Update

- An *assignment* to a variable uses the value of the expression to *update* the value of the variable
- The evaluation of an *expression* does *not update* the value of the variables in the expression

13

An Example

```
main( )
{
   int  english, chinese, temp;

   english = 95;
   chinese = 98;
   temp = english;
   english = chinese;
   chinese = temp;
}
```

english	chinese	temp
95		
95	98	
95	98	95
98	98	95
98	95	95

14

Operators

- Most operators can *only* manipulate operands of *one* specific data type
- The *integer addition* adds two integers and produces an integer
- The *floating point addition* adds two floating points and produces a floating point
- The data types of *expression* and *variable* in an assignment must be *compatible*

15

An Example

```
main( )
{
   double english, chinese, total;

   english = 95.0;
   chinese = 98.0;
   total = english + chinese;
}
```

english	chinese	total
95.0		
95.0	98.0	
95.0	98.0	193.0

16

Three Phases of Programs

- The *input phase* asks the user to enter the input data
- The *computation phase* uses the input data to compute the output data
- The *output phase* displays the output data to the user

17

An Example

```
#include  <stdio.h>

main( )
{
   int  english, chinese, total;

   printf("English score? "); scanf("%d", &english);
   printf("Chinese score? "); scanf("%d", &chinese);
   total = english + chinese;
   printf( "Total score = %d\n", total );
}
```

18

Output Statements

- The function *printf* is a general-purpose output formatting function
- Its first argument is a string of characters to be printed, with each % indicating *where* one of the other arguments is to be substituted, and *in what form* it is to be printed
- The most common formatting specifications

  %d int
  %lf double

19

Input Statements

- The function *scanf* is a general-purpose input formatting function
- Its formatting specifications in the first argument are almost the same as ones for printf
- The other arguments in scanf must be *variables* and must be applied the operator *&*

20

Precedence of Operators

- Apply first any *unary minus operators*
- Apply then the *multiplicative operators* (*, /, and %). If two of these operators apply to the same operand, the *leftmost* one is performed first
- Apply last the *additive operators* (+ and -). If two of these operators apply to the same operand, the *leftmost* one is performed first

21

An Example

```
#include  <stdio.h>

main( )
{
   double  english, chinese, average;

   printf("English score? "); scanf("%lf", &english);
   printf("Chinese score? "); scanf("%lf", &chinese);
   average = (english + chinese) / 2.0;
   printf( "Average score = %lf\n", average );
}
```

22

Overloaded Operators

- The notation '+' is said to be *overloaded* because it denotes *both* the integer and floating point additions; the notations '-', '*', and '/' are the same

```
celsius = 5.0 / 9.0;

celsius = 5 / 9;
```

23

Automatic Type Conversion

- If one operand of an operator is of type int and the other is of type double, then the int operand will be *automatically* converted to a double and a *double operation* is applied

9 / 4	int
9.0 / 4	double
9 / 4.0	double
9.0 / 4.0	double

24

An Example

```
#include  <stdio.h>

main( )
{
   double  fahr, celsius;

   printf("degrees in Fahrenheit? ");
   scanf("%lf", &fahr);
/* celsius = 5 / 9 * (fahr – 32); */
   celsius = 5 * (fahr – 32) / 9;
   printf("degrees in Celsius = %lf\n", celsius);
}
```

25

Type Cast

- The user can use a unary *type cast* operator to specify an *explicit* type conversion

```
quotien = num / den;
remainder = num % den;
ratio = num / (double) den;
ratio = (double) num / den;
```

26

Simple Statements

- A *simple statement* consists of an *expression* followed by a *semicolon*
- A *function call* is an *expression*; hence, input and output statements are simple statements
- An *assignment* is also an *expression*; hence, assignment statements are also simple statements

27

An Example

```
#include  <stdio.h>

main( )
{
   int  english, chinese, total;

   printf("English score? "); scanf("%d", &english);
   printf("Chinese score? "); scanf("%d", &chinese);
   total = english + chinese;
   printf( "Total score = %d\n", total );
}
```

6 simple statements

28

Assignment Expressions

- The = in an assignment is an operator called the *assignment operator*
- An *assignment* is therefore an *expression* and also produces a *value*
- The *value* produced by an assignment is the value assigned to the *variable*

29

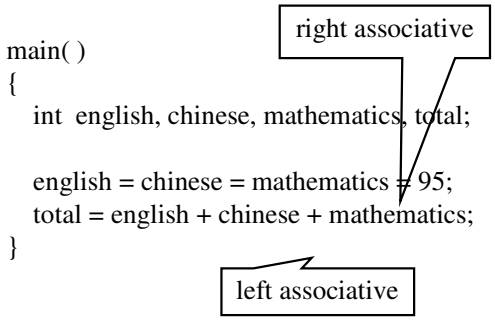
Embedded Assignments

```
main( )
{
   int  english, chinese, total;

   total = (english = 95) + (chinese = 98);
}
```

30

Multiple Assignments



31

Programming Idioms

- Idioms are *common* and *concise phrases* that are easy to remember
- Each programming language has its own idioms
- An example is the input of a data value
  printf("*prompt string*");
  scanf("*format string*", &*variable*);

32

Shorthand Assignment Idioms

```
main( )
{
   int  balance, deposit;

   balance = 1000;
   deposit = 100;
   balance = balance + deposit;
   balance += deposit;      /* -=, *=, /=, %= */
}
```

33

Increment & Decrement Operators

```
main( )
{
   int  balance;

   balance = 1000;
   balance = balance + 1;
   balance += 1;
   balance++;          /* increment */
   balance = balance - 1;
   balance -= 1;
   balance--;          /* decrement */
}
```

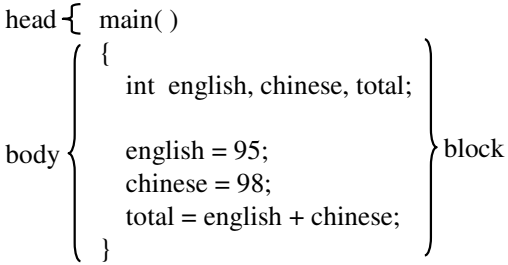
34

Compound Statements

- A *compound statement* or a *block* is a sequence of statements enclosed in *curly braces*
- The *body* of a function is a *block*
- *Variable declarations* can appear at the beginning of blocks
- A *block* can appear in any place where a *statement* can appear
- A block is *not* followed by a *semicolon*

35

An Example



36

An Example

```
main( )
{
   int  english, chinese, total1, total2;

   english = 95; chinese = 98;
   total1 = english + chinese;
   {    int temp;
        temp = english; english = chinese; chinese = temp;
   }
   total2 = english + chinese;
}
```

block

37

Programming Style

- Programs are *read* more often than they are *written*
- Good style and program readability are *critical* for program maintenance
- You should *proofread* your own program for style, just as a writer would proofread an article

38

Stylistic Guidelines

- Use *comments* to tell your readers what they need to know
- Use *indentation* to mark the various levels of program control
- Use *meaningful identifiers* and develop a *convention for identifiers*
- Use standard *idioms* and *conventions* when appropriate
- *Avoid* unnecessary complexity

39

Comments

- Each comment begins with **/*** and ends with ***/**
- Use comments to explain something that is *complicated* or *difficult to understand*
- Avoid comments to explain something that is *obvious*
- Be sure to *update comments* when you *update programs*

40

Conventions for Indentations

- Set a convention for the *size* of indent (3 or 4 spaces)
- Place a *space* after each comma ‘,’
- Place *spaces* on either side of an operator
- Place a *blank line* between declarations and statements

41

Conventions for Identifiers

- Identifiers of *variables* and *data types* begin with a *lowercase letter*
- Identifiers of *functions* begin with an *uppercase letter*
- Whenever an identifier consists of more than one word, the *first* letter in each word is *capitalized*

42