Chapter 1 Review

- **What are the three types of language translators?**
- A: assemblers, compilers, and interpreters.
- **Please describe the basic structure of C program.**

```
# preprocessing directives

void main(void)
{
  declaration statements;
  executable statements;
} /* any text, number, or character */
```

- **What are the three types of errors in C source code?**
- A: syntax errors, run-time errors, and logic errors.

Chapter 2

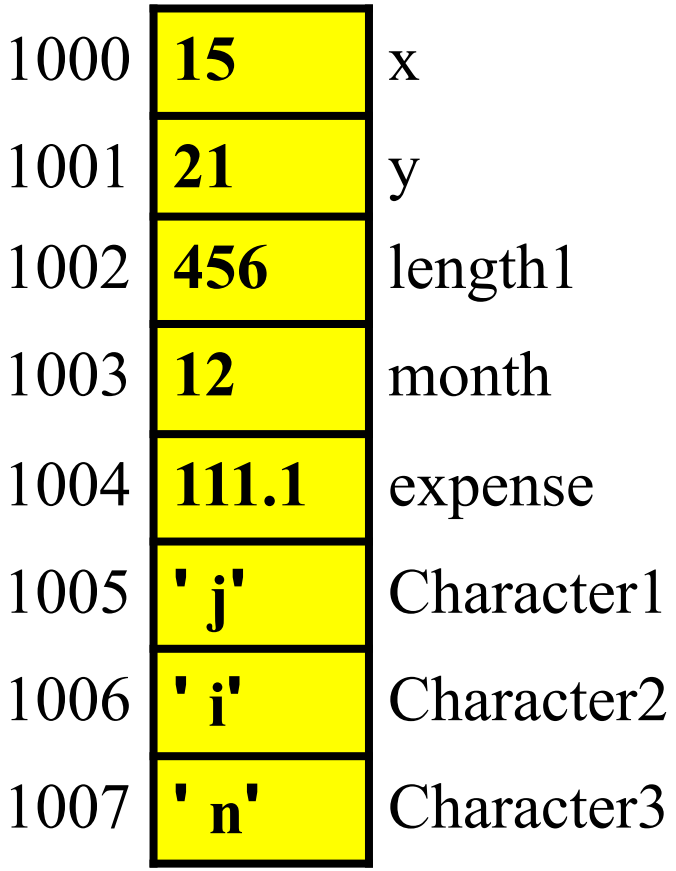
Variables, Arithmetic Expressions and Input/Output

Topics

- Naming variables
- Declaring data types
- Using assignment statements.
- Displayng variable values
- Elementary assignment statements

2.1 Variables: Naming, Declaring, Assigning and Printing Values

- Variables
- variable names
  - consist of entire words rather than single characters
  - Why?
  - easier to understand your programs if given very *descriptive* names to each variable



2.1 Variables: Naming, Declaring, Assigning and Printing Values

```
#include <stdio.h>
void main(void)
{
    int    month;
    float    expense, income;
    month = 12;
    expense = 111.1;
    income = 100.;
    printf ("Month=%2d", month);
    return 0;
}
```

Result?

Concepts

- Variable names must be ***declared before used***
- "Declare" all your variable names near the beginning of your program
- Variable names are classified as *identifiers*
  - first character must be non-digit characters a–z, A–Z, or _
  - other characters must be non-digit characters a–z, A–Z, _, or digit 0–9
- Valid examples
  - apple1 interest_rate xfloat Income one_two
- Invalid
  - 1apple interest_rate% *float* In come one.two

Some Constraints on Identifiers

- Use of uppercase or mixed-case is allowed
- However, many programmers use lowercase characters for variable names and uppercase for constant names. Differentiate your identifiers by using different characters rather than different cases

- More in Table 2.1 (page 42)

Hungary Notation(匈牙利表示)

- Hungarian Notation is the practice of including a prefix in identifiers to encode some metadata about the parameter, such as the data type of the identifier

```
int       nMonth;
long      lTotal;
float     fSalary,fAverage;
```

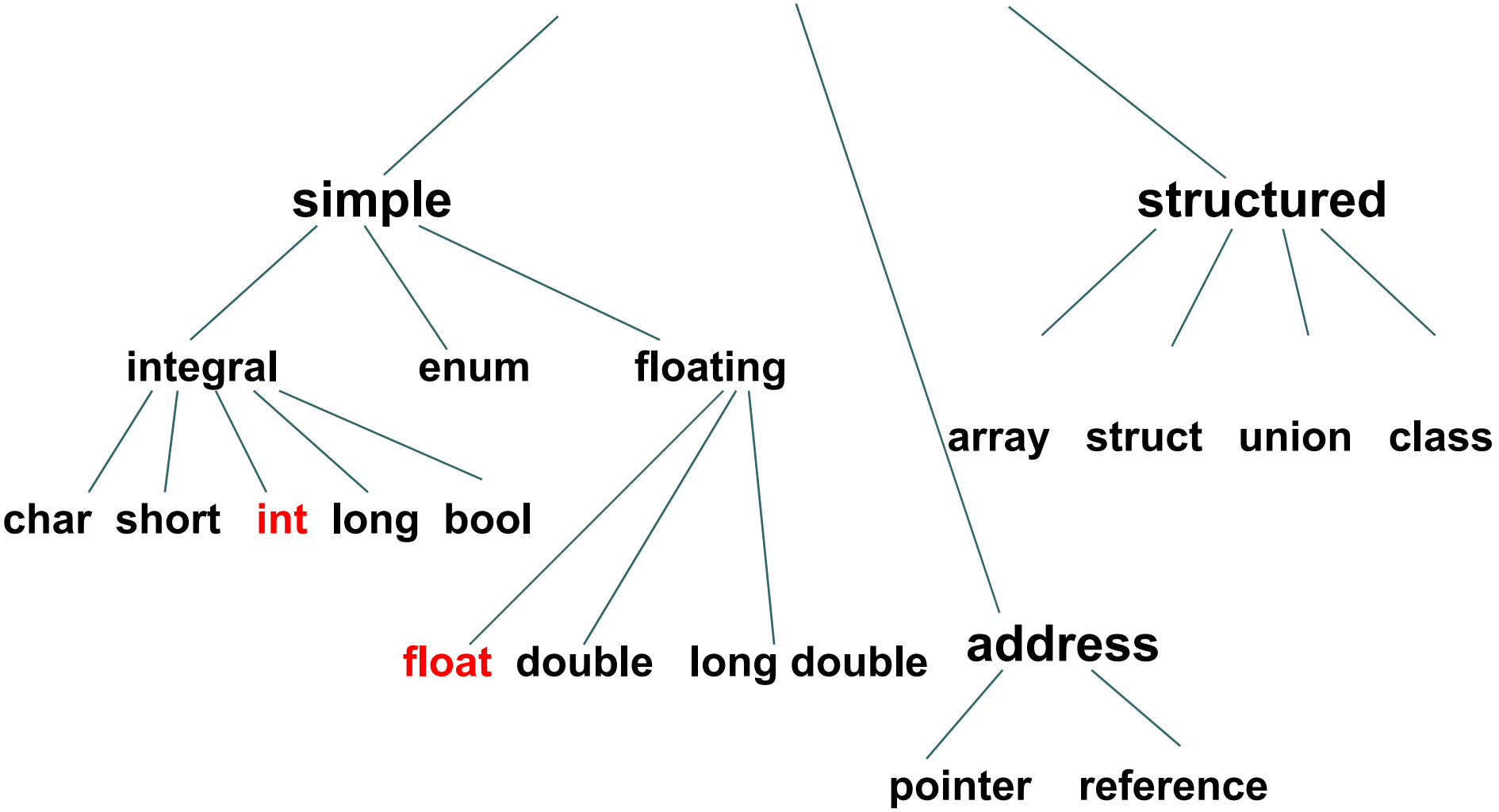
Prefix	Implicit Type
i,n	integer
f	float
l	long
c	char
by	BYTE
p	pointer
s	string
…	…

Keyword(Reserved Word)

- A keyword is an identifier type token for which C has a defined purpose
- Cannot use them as variable names
- Number of keywords in C is very small, just 32
- Refer to Table 2.1(Page42)

auto break case char const void union default
do double else enum extern float for static
if int long return short signed struct typedef
goto switch while volatile unsigned continue
register sizeof

C++ Data Types



Assignment Statement(Page41)

- General form

```
variable_name = value;
```

month = 12;

- Assigns a value to a variable
- Causes a value to be stored in the variable's memory location
- Equal sign(=) does **not really mean equal**

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{  /*C2_0*/
int     nYear,n;
float   pi;
float   a,b,c,d,e,f;

    nYear=2015;
    pi=3.141592653589793;

    return 0;
}
```

Exercises (Page46)

2. Which of the following are incorrect C variable names and why?

   enum, KNUM, lotus123, A+B123, A(b)c, AaBbCc,

   Else, αβχ, pi, Л

3. Which of the following are incorrect C assignment statements and why?

   year=1967

   1967 =oldyear;

   day= 24 hours;

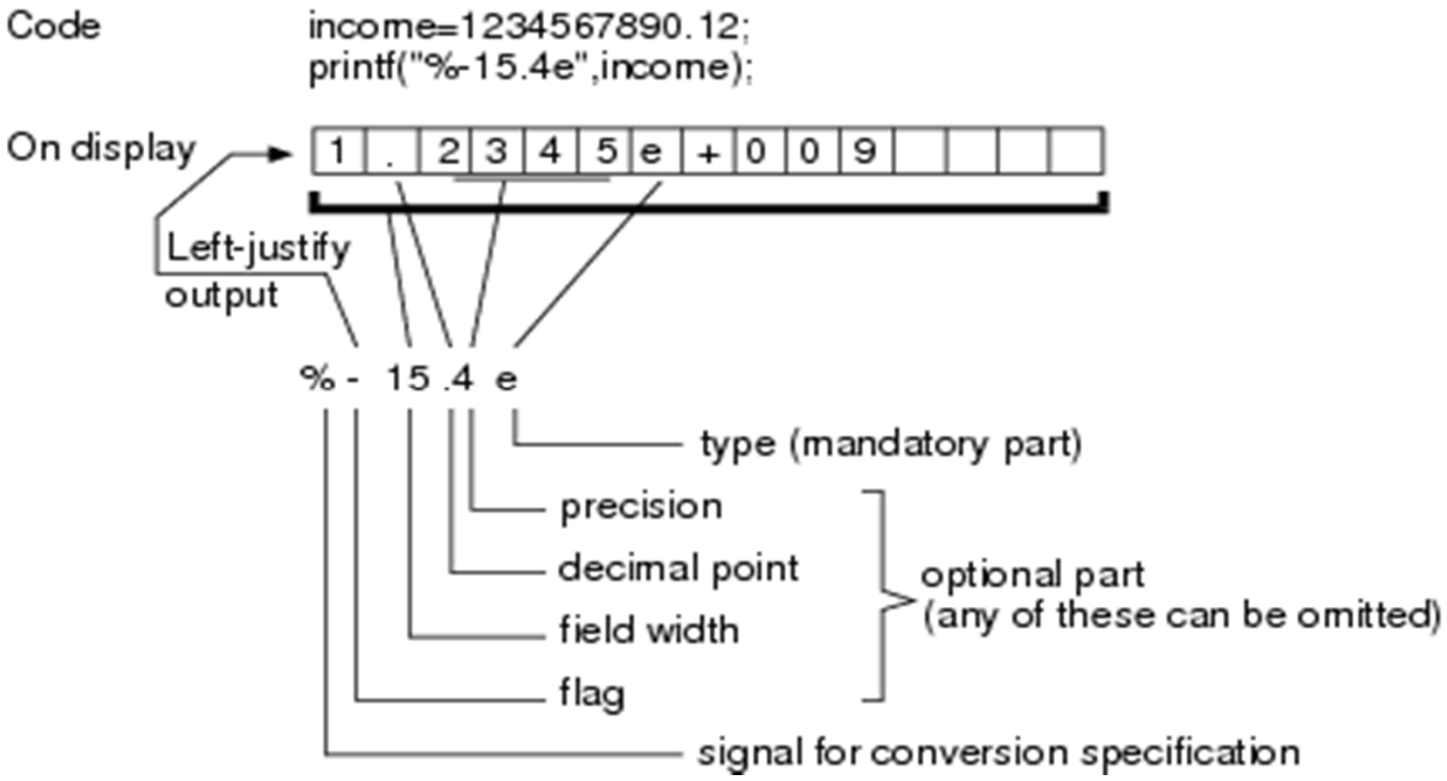
   while=32;

printf (page 42-43)

- To display the value of a variable or constant on the screen
  ***printf*(format_string, argument_list);**
- **format_string**
  - ***plain characters*** – displayed directly unchanged on the screen
    printf("This is C ");
  - ***conversion specification*(s)** – used to convert, format and display argument(s) from the argument_list
  - ***escape sequences*** – control the cursor, for example,
  - the newline '\n'
- Each argument must have a format specification. For example,
  ***printf*("pi=%f\n", 3.14);**

Conversion Specification(Page 43-44)

- Complete structure of format specifications is **%[flag][field width][.precision]type**
- [ ] (square brackets) meaning *optional*
- Page62



Conversion Specification(Page44)

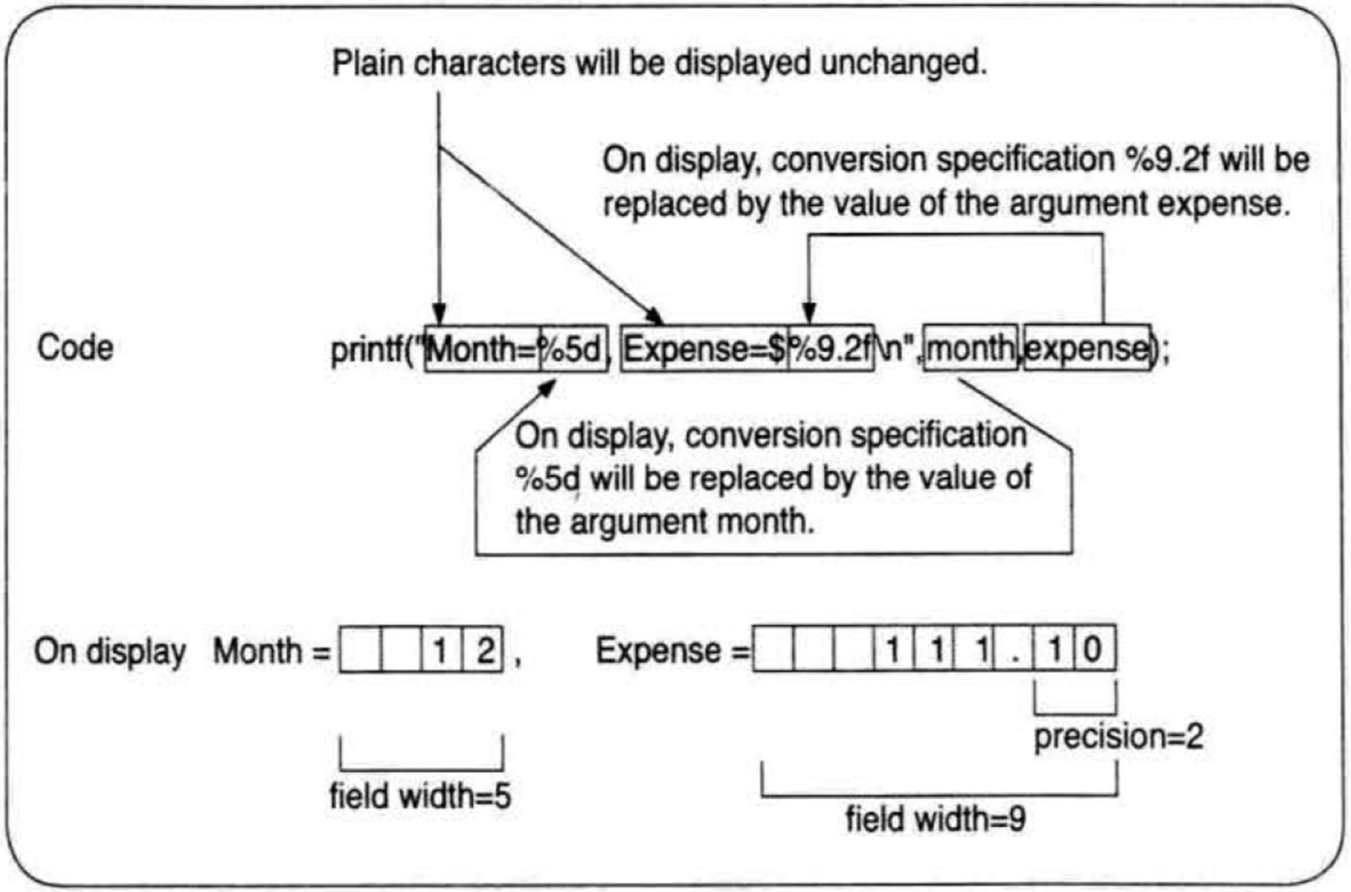


Fig. 2.1 *Format specifications in printf*

Conversion Specification

- If the precision specified for a real is
  - less than actual, displays only the number of digits in the specified precision
  - greater than actual, adds trailing zeros to make the displayed precision equal to the precision specified
  - not specified, makes the precision equal to 6
- The flag –
  - left-justifies a value that is put in a field width that is greater than the actual.

Table2.2 Flags and Types(Page61)

Component	Use
flag = -	This flag causes the output to be left justified within the given field width
flag = +	This flag causes the output to be right justified within the given field width and a plus sign displayed if the result is positive
flag = 0	This flag causes leading zeros to be added to reach the minimum field width; the flag is ignored if the 2 flag is used simultaneously
field width	This integer represents the minimum number of character spaces reserved to display the entire output (including the decimal point, digits before and after the decimal point, and the sign). If the specified field width is not given or is less than the actual field width, the field width is automatically expanded on print out to accommodate the value being displayed. The field width and precision are used together to determine how many digits before and after a decimal point will be displayed
precision	For floating data types, precision specifies the number of digits after the decimal point to be displayed. The default precision for float type (e, E, or f) data is six. Precision also can be used for integer type data, where it specifies the minimum number of digits to be displayed. If the data to be displayed has fewer digits than the specified precision, the C compiler adds leading zero(s) on the left of the output
type = d	For int type data
type = f	The output is converted to decimal notation in the form of [sign]ddd.dddd, where the number of digits after the decimal point is equal to the specified precision
type = e or E	The output is converted to scientific notation in the form of [sign]d.dddd e[sign]ddd, where the number of digits before the decimal point is one, the number of digits after the decimal point is equal to the specified precision, and the number of exponent digits is at least two. If the value is zero, the exponent is 0.

Table2.3 Flags and Types(Page65)

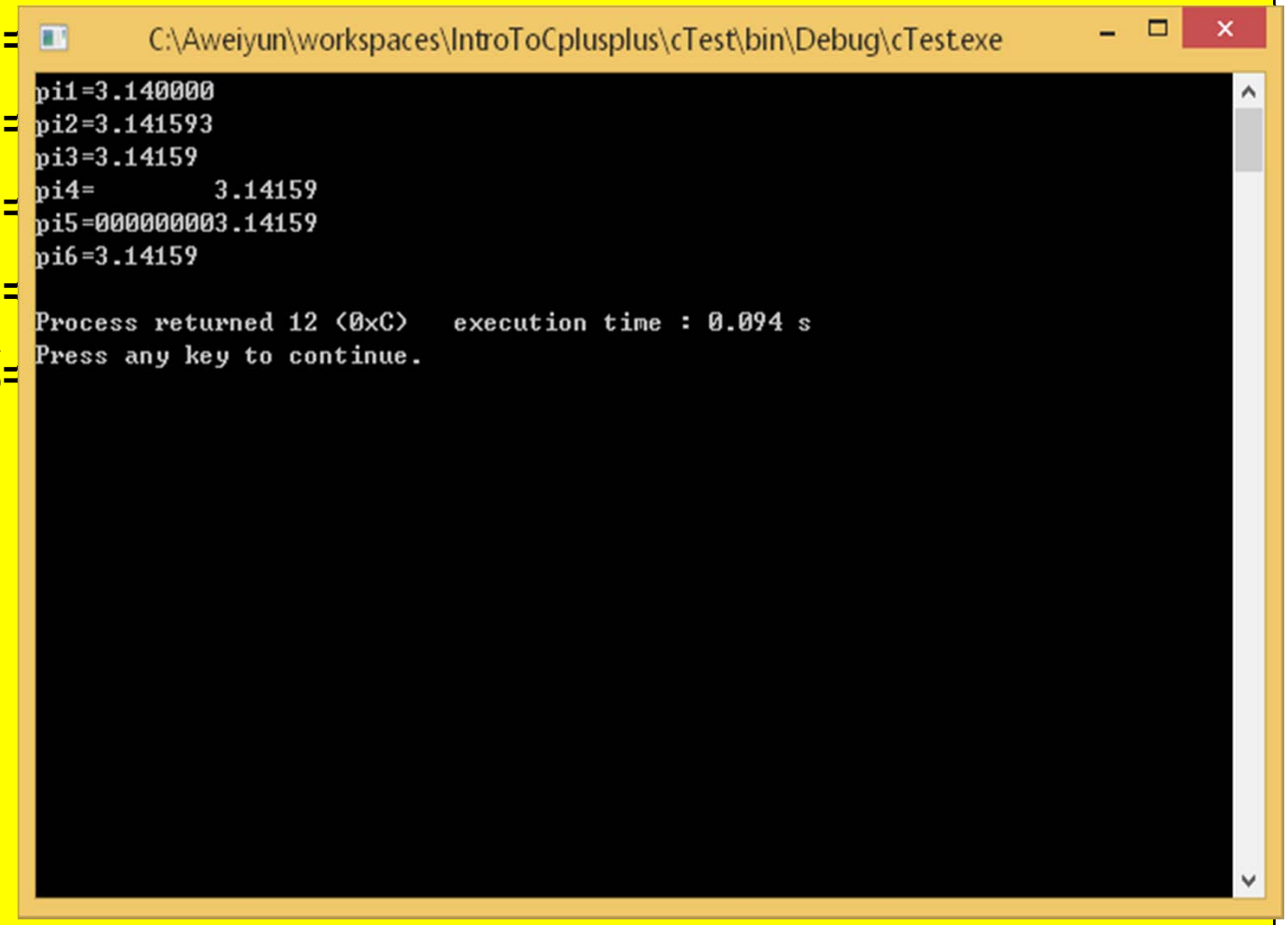
		Field			Display	Conversion Flag
width	Type	Precision		Note	(␣ means blank)	
%+5d	+	5	d	none	␣+365	Right-justified output, 1 sign added, total characters displayed is five
%-5d	-	5	d	none	365␣␣	Flag is 2, so output is left justified
%1d	none	1	d	none	365	Specified field width is less than the actual width, all characters in the value are displayed, no truncation occurs
%0.5d	zero	0	d	5	00365	Flag is 0, so output is prefixed with zeros, precision is 5, so the number of characters to be printed is five
%d	none	none	d	none	365	Field width is undefined, all characters in the value are displayed, no truncation occurs; no blanks are added; value is left justified
%+9.5f	+	9	f	5	␣+3.14160	Total digits, including blanks, is nine
%-9.5f	-	9	f	5	3.14160␣␣	Flag is -, left-adjusted output
%1.3f	none	1	f	3	3.142	Uses precision 3, note the result is 3.142, not 3.141
%f	none	none	f	none	3.141600	Uses the default precision, 6
%+12.4e	1	15	e	4	␣1.2346e+009	output, total digits is 12, field width of 15 accommodates the e and 1, precision is 4
%-12.4e	2	15	e	4	1.2346e+009␣	Same as previously, but flag - , so output is left justified
%5.2e	none	5	e	2	1.23e+009	Precision is 2; field width is too short, so C uses minimum field width for output
%E	none	none	E	none	1.234568E1009	C uses default precision of 6; field width is too short, so C uses minimum field width for output

```
float   pi=3.141592653589793;

    printf ("pi1=%f\n",3.14);
    printf ("pi2=%f\n",pi);
    printf ("pi3=%1.5f\n",pi);
    printf ("pi4=%15.5f\n",pi);
    printf ("pi5=%015.5f\n",pi);
    printf ("pi6=%.5f\n",pi);
    return 0;
```

```
float   pi=3.141592653589793;

    printf ("pi1=%f\n",3.14);
    printf ("pi2=
    printf ("pi3=
    printf ("pi4=
    printf ("pi5=
    printf ("pi6=
    return 0;
```



Exercises (Page47)

4. Supposing year is an *int* variable and salary is a *float* variable, which of the following *printf()* statements are unacceptable and why?

```
printf("My salary in 2007 is $2000",salary);
printf("My salary in 2007 is %d\n",salary);
printf(In year %d, my salary is %f\n"),year,salary;
printf("My salary in %d year is %f \n,salary,year");
printf("My salary in %5d year is %10.2f \n\n",year,salary);
```

Exercises (Page47)

5. The price of an apple is 50 cents, a pear is 35 cents and a melon is 2 dollars. Write a program to display the prices as follows:

```
***** ON SALE *****
Fruit type          Price
Apple               $ 0.50
Pear                $ 0.35
Melon               $ 2.00
```

2.2 Arithmetic Operators and Expressions(page47)

- Consists of a sequence of *operand*(s) and *operator*(s) that specify the computation of a value
- Look much like algebraic expressions that you write
  d = x/y; // x divided by y
- Assigns the value of the arithmetic expression(division) on the right to the variable on the left

Variable Initialisation

- How do we initialise variables?
  - Uses an assignment statement, e.g.
    e=3;
  - Initialises in a declaration statement, e.g.
    float a=7, b=6;

```
int     i,j,k,p;                          /*C2_2*/
float   x,y;

i=5;   j=5;
k=11;  p=3;
x=3.0; y=4.0;
printf("..... Initial values .....\n");
printf("i=%4d, j=%4d\nk=%4dn", i,j,k);
printf("p=%4d\nx=%4.2f, y=%4.2f\n\n",p,x,y);
```

```
/*Section1*/                              /*C2_2*/
float    a,b,c,d,e,f;

a=x+y;
b=x-y;
c=x*y;
d=x/y;
e=d+3.0;
f=d+3;
i=i+1;
j=j+1;

printf("..... Section 1 output .....\n");
printf("a=%5.2f, b=%5.2f\nc=%5.2f, d=%5.2f\n", a,b, c,d);
printf("e=%5.2f, f=%5.2f\ni=%5d, j=%5d \n\n",e,f,i,j);
```

2.2 Arithmetic Operators and Expressions(Page49)

Operators ++, --

++

- **same as i=i+1;**
- ***increment* operator, can be placed before or after a variable**

```
int   i=10;
i++;
++i;
```

- **Only difference between i++ and ++i is in order of increment**

```
int  i=10,x,y;
x=i++;
y=++i;
```

i--; or --i;

- **same as i=i-1;**

2.2 Arithmetic Operators and Expressions(Page49)

Operator %

%

- **is a *remainder* operator**
- **must be placed between two *integer* variables or constants**
- **11%3 return**

  2

```
/*Section2*/
int       m,n,u;

  u=k%p;
  i++;
  ++j;
  m=i++;
  n=++j;
  printf("..... Section 2 output .....\n");
  printf("m=%4d, n=%4d\ni=%4d, j=%4d\n",m,n, i,j);
  printf("e=%4.2f, f=%4.2f\n", e,f);
```

Arithmetic Operators and Expressions

- Cannot write
```
x/y = d;
i + 1 = i;
```
- Left side of assignment statement can have only *single* variable

- lvalues Vs. rvalues (page 51)

2.3 Reading Data from The Keyboard(Page52)

- We can instruct the computer to retrieve data from various input devices
  - the keyboard
  - a mouse
  - the hard disk drive
- Programs that have input from the keyboard usually create a dialogue between the program and the user during execution

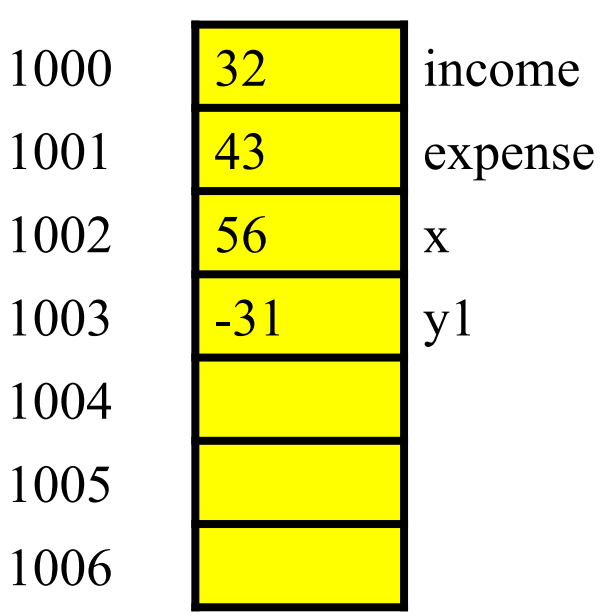
Addresss of a Variable(Page55)

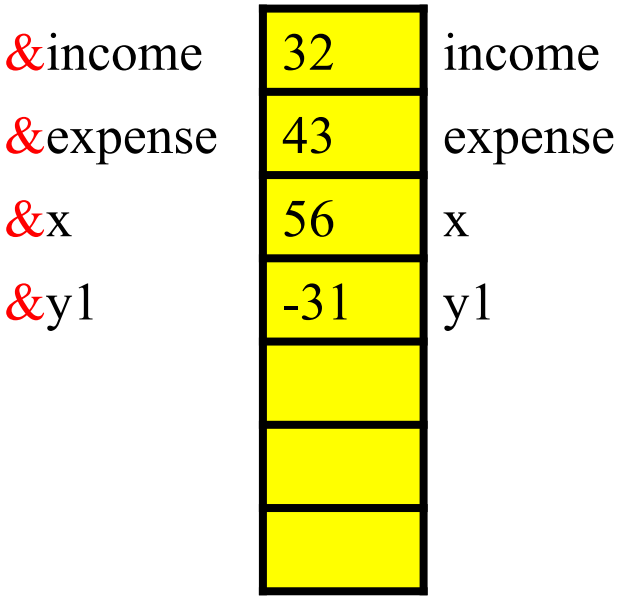
&

stands for the address of a memory

& : “address of” operator

&*income*=address of variable *income*





scanf() function (page54)

- ***scanf***() function format

  ***scanf* (format_string, argument_list);**

- format_string converts characters in the input into values of a specific type
- argument_list contains the ***address*** of the variable(s)

  ***scanf*("%f",&income);**

  ***scanf*("%f%lf",&income,&expense);**

  - 1st keyboard input data converted to float (%f) => income
  - 2nd keyboard input converted to double (%lf) => expense
  - By giving scanf the address, the program knows where in memory to put the value typed

Reading Data from the Keyboard

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
int                 month;          /*C2_3*/
    printf ("What month is it?\n");
    scanf ("%d", &month);
    printf ("You have entered month=%5d\n",month);
    return 0;
}
```

Reading Data from the Keyboard

```
float          income;
double         expense;

 printf ("Please enter your income and expenses\n");
 scanf ("%f %lf",&income,&expense);
 printf ("Entered income=%8.2f, expenses=%8.2lf\n",
          income,expense);
```

Reading Data from the Keyboard

```
int             hour,minute;

 printf ("Please enter the time, e.g.,12:45\n");
 scanf ("%d:%d",&hour,&minute);
 printf ("Entered Time = %2d:%2d\n",hour,minute);
```

2.4Constant Macros宏 (Page 57)

- To create a constant macro
  - begin with the symbol # (which must begin the line)
  - semicolon must **not** be used at the end
- General form

  #define **DAYS_IN_YEAR** **365**
- Prior to translation into machine code, the preprocessor replaces ***every*** symbolic_name(**DAYS_IN_YEAR**) in the program with the given replacement(**365**)

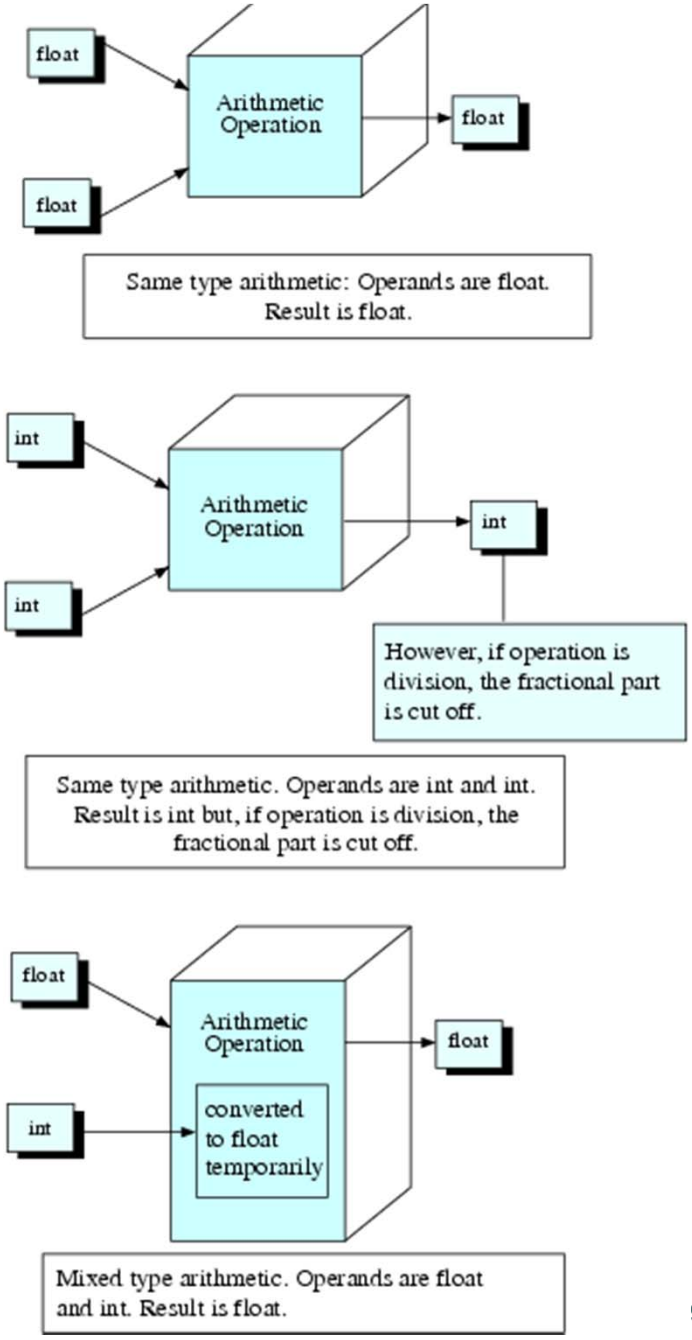
  printf("Days in year=%5d\n",**DAYS_IN_YEAR**);

  after preprocessing becomes

  printf("Days in year=%5d\n",**365**);

2.5Mixed Data Type Calculation(Page69)

- 6.0/4.0 => ?
- 6/4 => ?
- 6/4.0 =>?
- C converts integer to real temporarily and perform the operation



Try!（Cont.）

```
fY7=3;
printf("fY1=%3.0f,fY2=%3.0f,fY3=%3.0f,"
       "fY4=%3.1f,fY7=%3.1f\n",fY1,fY2,fY3,fY4,fY7);
fY8=fY1+fY2-fY3/fY4*fY7;
printf("fY8=%10.3f \n",fY8);
fY9=fY1+(fY2-fY3)/fY4*fY7;
printf("fY9=%10.3f \n",fY9);
fY10=((fY1+fY2)-fY3/fY4)*fY7;
printf("fY10=%10.3f\n",fY10);
```

*cast*强制转换*Operators*(page 74)

- Change the type of an expression temporarily
- General form

(type) expression

Force a floating
Point division

```
int     a=5, b=2;
float  x;
   x = (float) a / b;
```

x = 2.5, otherwise x = 2.0

Try!

```
fY5=6/4;
fY6=(float)6/4;
fY7=(int)((float)6/4);
fY8=(int)(float)6/4;
printf("fY5=%3.1f,fY6=%3.1f,“
          "fY7=%3.1f,fY8=%3.1f\n",fY5,fY6,fY7,fY8);
```

*Compound Assignment*复合赋值

(Page75)

```
x+=2;
```

```
x=x+2;
```

```
y*=2;
```

```
y=y*2
```

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])                    /*C2_4*/
{
int         nX1,nX2,nX3=10,nX4=20,nX5=30,nX6=40,nX7=50;
float       fY1=7.0,fY2=6,fY3=5,fY4=4;
float       fY7,fY5,fY6,fY8,fY9,fY10;

  nX1=6/4;
  nX2=6/4.0;
  fY5=6/4;
  fY6=6/4.0;
  printf("nX1=%2d,nX2=%2d\nfY5=%3.1f,fY6=%3.1f\n\n",nX1,nX2,fY5,fY6);

  printf("Original nX3=%2d,nX4=%2d,nX5=%2d"
          "nX6=%2d,nX7=%2d\nX2",nX3,nX4,nX5,nX6,nX7);
  nX3+= 2;
  nX4-=2;
  nX5*=(8/4);
  nX6/=2.0;
  nX7%=2;
  printf("New nX3=%2d,nX4=%2d,nX5=%2d,"
          "nX6=%2d,nX7=%2d\n\n",nX3,nX4,nX5,nX6,nX7);
}
```

*Controlling Precedence*优先

- Arithmetic operators located within the parentheses() always have highest precedence
- Example

```
z = ((a+b)*c/d);
```

- a+b evaluated first

Associativity 结合性 (pp. 77)

- Specifies the *direction* of *evaluation* of the operators with the same precedence

Side Effect 副产品 (pp. 78)

- Primary effect of evaluating an expression is arriving at a value for that expression
- Anything else occurs during evaluation of expression is considered a ***side effect***
- eg. Assume i=7,

  j = i++;
  - **j=7** is primary effect
  - i is changed to 8, is side effect
- eg.

  j = (i=4) + (k=3) - (m=2);
  - j = 5 is primary effect (4 + 3 – 2)
  - Three side effects:
    - Set i equal to 4
    - Set k equal to 3
    - Set m equal to 2

Try!

```
nX1=1;
nX2=2;
nX6=20;
nX3=nX1+nX2++;
nX4=nX1+--nX2;
nX5=nX1+nX6%3;
```

Home Work

1. Page57-2
2. Page67-2
3. Page68-3
4. Page68-4
5. Page82-2
6. Page82-3
7. Page82-5 (Program)
8. Page82-6