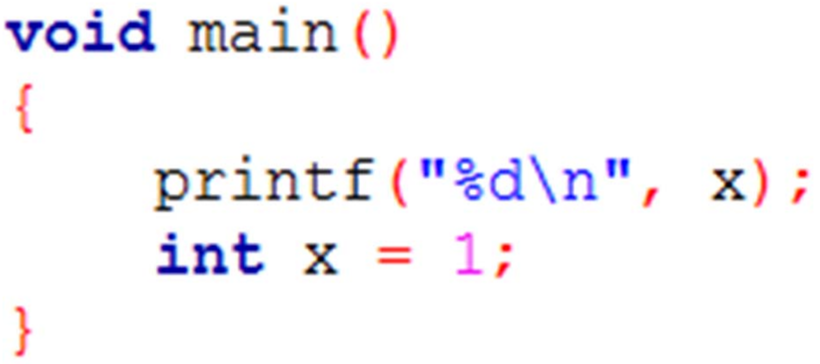
Chapter 2 Review

- Correct?

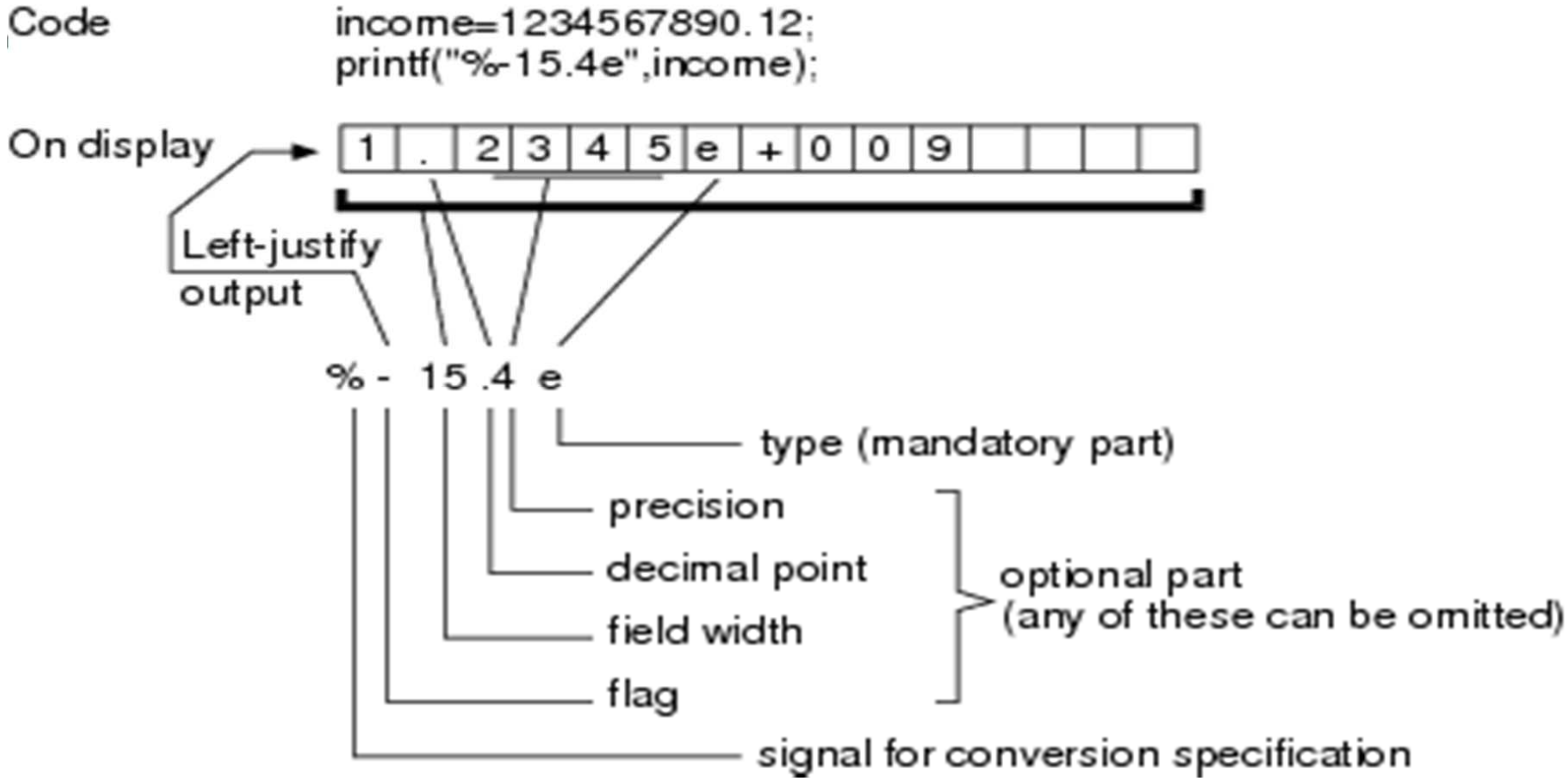


```
void main()
{
     printf("%d\n", x);
     int x = 1;
}
```

- A variable must be declared before used!

Chapter 2 Review

- To display the value of a variable or constant on the screen: ***printf*(format_string, argument_list);**
- The complete structure of **format_string** is **%[flag][field width][.precision]type**



Code

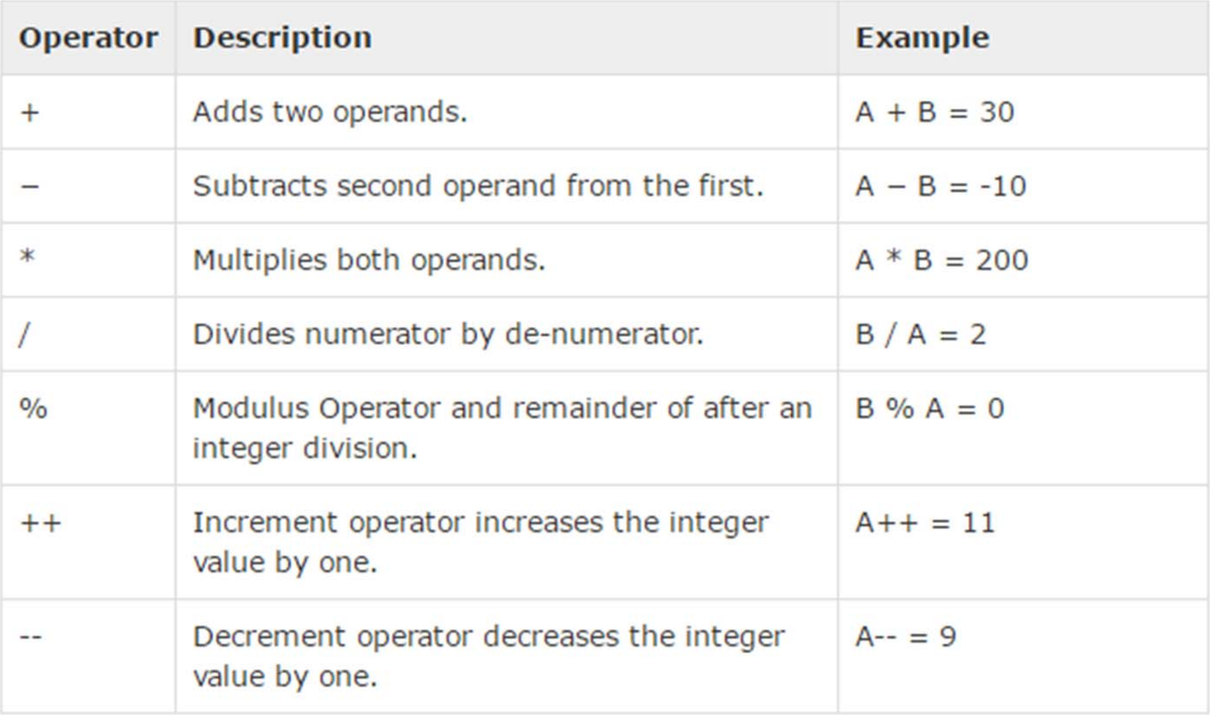
```
income=1234567890.12;
printf("%-15.4e",income);
```

On display

1	.	2	3	4	5	e	+	0	0	9				

Chapter 2 Review

- Arithmetic operators (suppose A = 10 and B = 20)



Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Chapter 2 Review

- Correct?

```
int cat, dog;
float weight;
scanf (“%d, %d”, cat, weight);
```

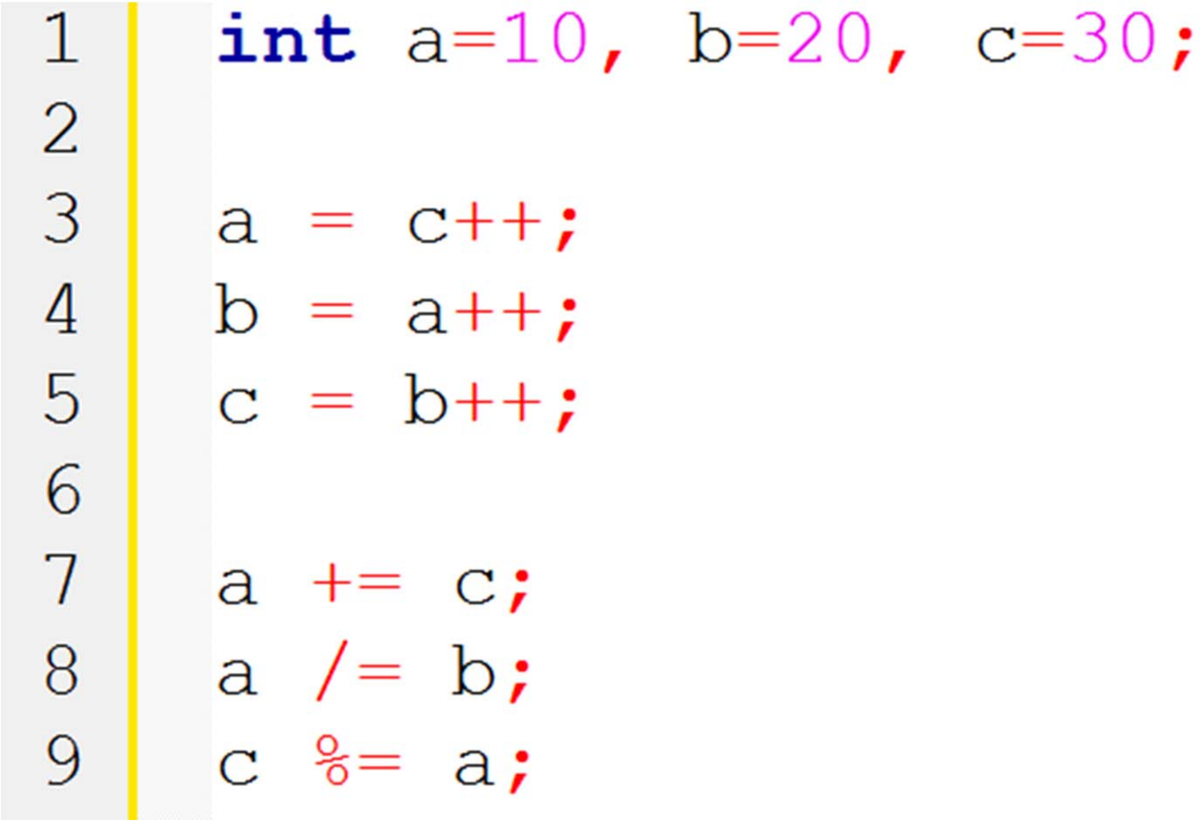
Chapter 2 Review

- Correct macro?

```
#DEFINE PI 3.1416;
```

Chapter 2 Review

- Values of a, b, c from Line 3 to 9?

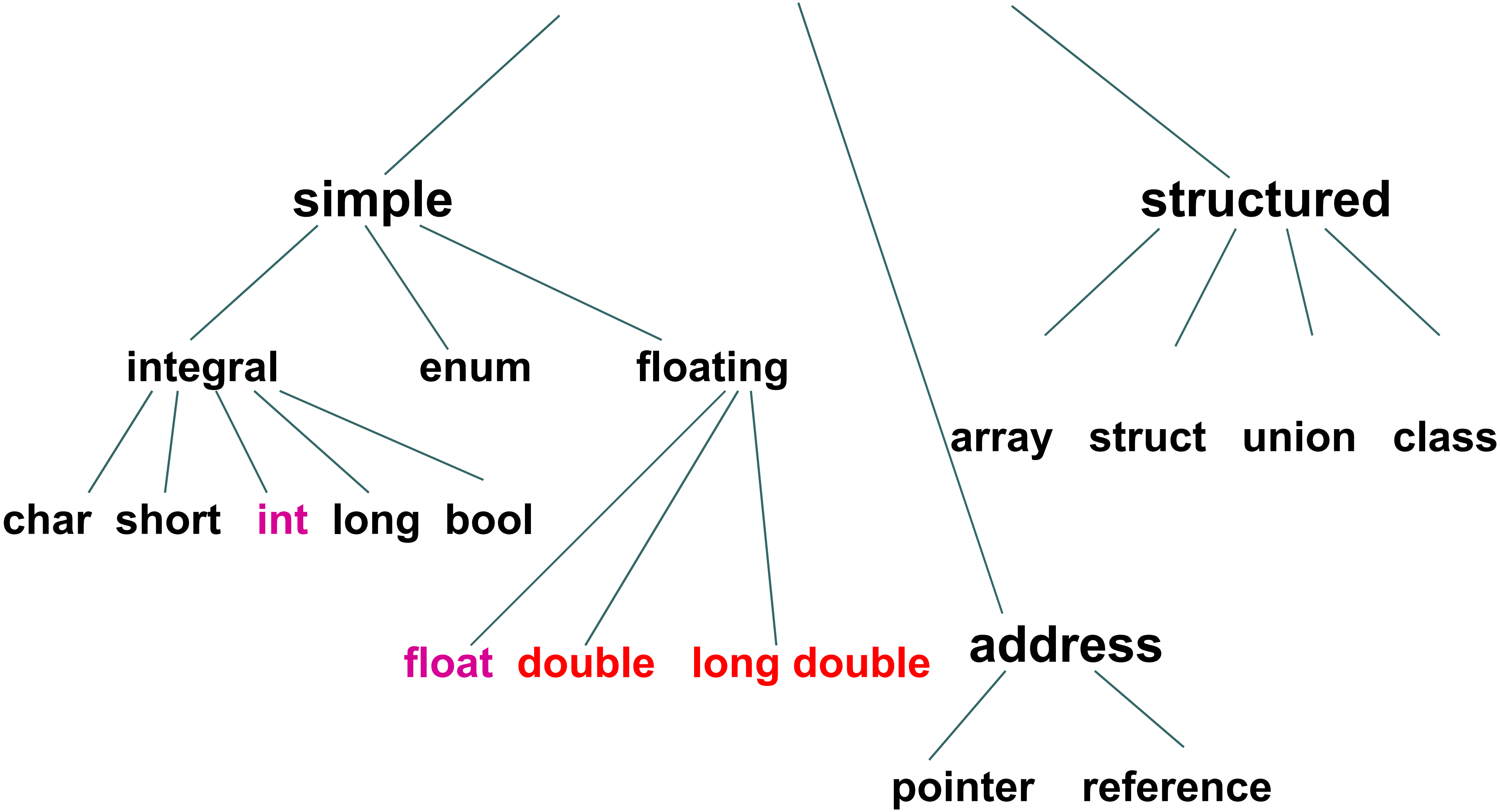


```
1  int a=10, b=20, c=30;
2
3  a = c++;
4  b = a++;
5  c = b++;
6
7  a += c;
8  a /= b;
9  c %= a;
```

Chapter 3

The Basics of C-Math Functions and Character File Input Output

C++ Data Types



simple

structured

integral

enum

floating

array struct union class

char short int long bool

float double long double

address

pointer reference

C data types

- Integers: ***char, int, short, long, long long***
- Unsigned integers: ***unsigned char, unsigned int, unsigned long, unsigned long long***
- Floating point numbers: ***float, double, long double***
- Note that: C does **NOT** have ***bool*** until C99 by including <stdbool.h> (more in Chapter 4)
- Structures (more in Chapter 8)

Use of double **Data Type**

- ***double*** can carry a large number of digits
  - important when doing a large number of calculations
  - more memory is required

  double **x=3.0, y=4.0, a,b,c,d,e,f;**

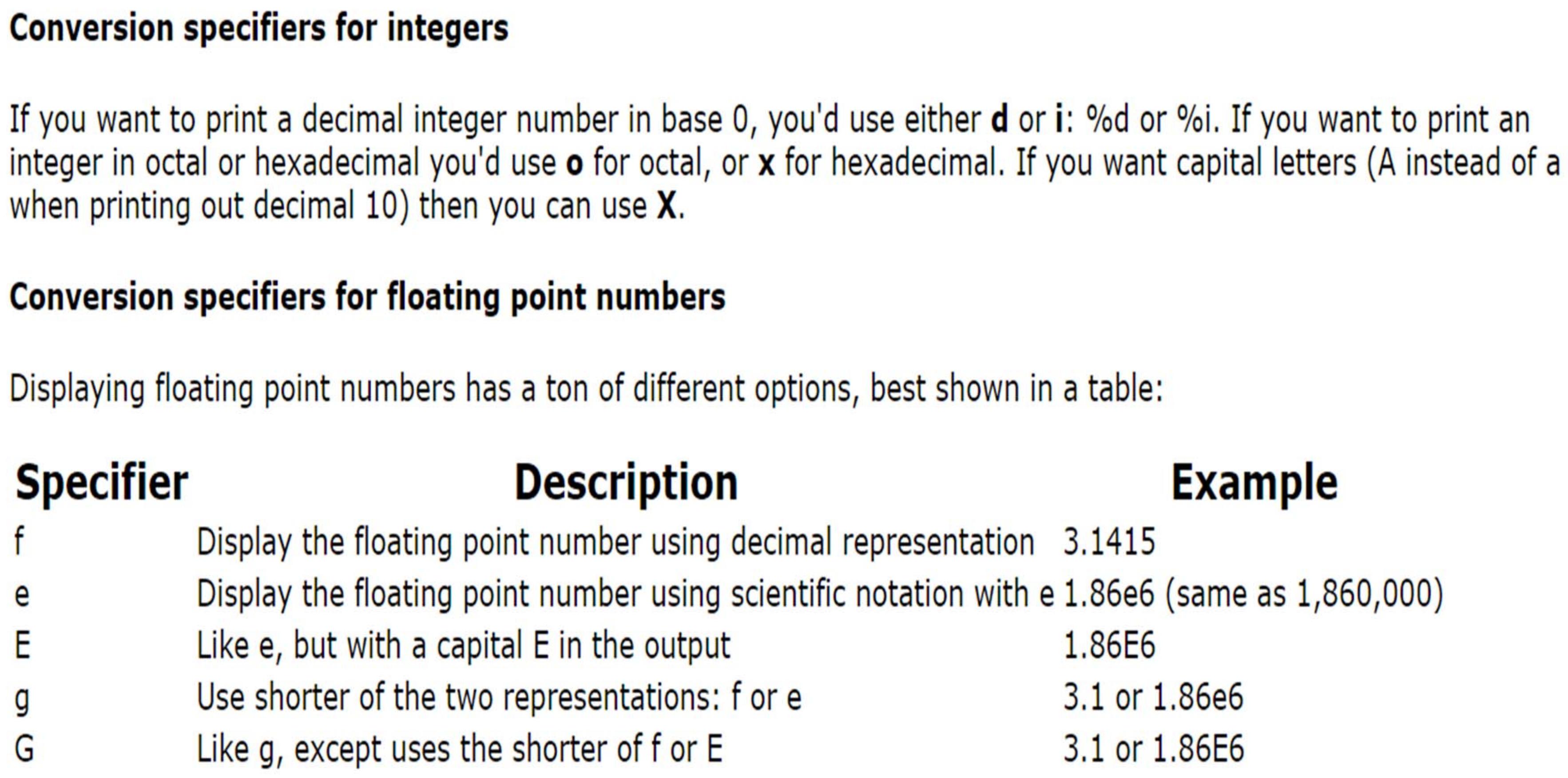
Different Types for Real Number(page87)

Item	***float***	***double***	***long double***
Memory used	4 bytes = 32 bits	8 bytes = 64 bits	10 bytes = 80 bits
Range of values	1.1754944E - 38 to 3.4028235E + 38	2.2250738E - 308 to 1.7976935E + 308	Approximately 1.0E - 4931 to 1.0E + 4932
Precision	6	15	19
printf format	%f, %e, %E, %g, %G	%lf, %e, %E, %g, %G	%Lf, %Le, %LE, %Lg, %LG

Different Types for Integers(page87)

	short int		***int, long int***	
Item	***signed short int***	***unsigned short int***	***signed int, signed long int***	***unsigned int , unsigned long int***
Memory used	2 bytes = 16 bits	2 bytes = 16 bits	4 bytes (32 bits)	4 bytes (32 bits)
Range of values	-32768 to 32767	0 to 65535	-2147483648 to 2147483647	0 to 4294967295
Simplest format	**%d**, may need to use %hd for short int	**%d**, may need to use %hd for unsigned short int	**%d** (int) %ld (long)	**%d** (int) %ld (long)

Format types for integer and floats



Conversion specifiers for integers

If you want to print a decimal integer number in base 0, you'd use either **d** or **i**: %d or %i. If you want to print an integer in octal or hexadecimal you'd use **o** for octal, or **x** for hexadecimal. If you want capital letters (A instead of a when printing out decimal 10) then you can use **X**.

Conversion specifiers for floating point numbers

Displaying floating point numbers has a ton of different options, best shown in a table:

Specifier	Description	Example
f	Display the floating point number using decimal representation	3.1415
e	Display the floating point number using scientific notation with e	1.86e6 (same as 1,860,000)
E	Like e, but with a capital E in the output	1.86E6
g	Use shorter of the two representations: f or e	3.1 or 1.86e6
G	Like g, except uses the shorter of f or E	3.1 or 1.86E6

Try it!

3.1 *C Mathematical Library Functions* (page88)

FUNCTION NAME	CALCULATING
sin(x)	sine of x, x is in radians
exp(x)	natural exponential of x
log(x)	natural logarithm of x
sqrt(x)	square root of x
pow(x, y)	x (raised) to the power of y

- Input argument(s) x or y and return values are of ***double*** type
- For sin/cos/tan/…, arguments are in **radians**, not degrees

Math Library Functions

```
#include "stdafx.h"
#include "math.h"  //must include header file for math functions
#define          PI       3.1415926
int _tmain(int argc, _TCHAR* argv[])      //C3_1
{
double           x,a,b,c,d;


  x=30.0;
  a=sin(x);
  b=sqrt(9.0);
  c=pow(x,2.0);
  d=log10(1000.0);

  return 0;
}
```

Function name	Example	Description
abs(x)	y=abs(x);	absolute value of an int type argument, x and y are of type int (Note: needs #include <stdlib.h> not math.h)
fabs(x)	y=fabs(x);	absolute value of a double type argument, x and y are of type double (Note: needs #include <stdlib.h> not math.h)
sin(x)	y=sin(x);	sine of an angle in radians, x and y are of type double
sinh(x)	y=sinh(x);	hyperbolic sine of x, x and y are of type double
cos(x)	y=cos(x);	cosine of an angle in radians, x and y are of type double
cosh(x)	y=cosh(x);	hyperbolic cosine of x, x and y are of type double
tan(x)	y=tan(x);	tangent of an angle in radians, x and y are of type double
tanh(x)	y=tanh(x);	hyperbolic tangent of x, x and y are of type double
log(x)	y=log(x);	natural logarithm of x, x and y are of type double
log10(x)	y=log10(x);	logarithm to the base 10 of x, x and y are of type double

rand() Library Functions

- The ***int rand(void)*** function returns a pseudorandom伪随机 integer in the range 0 to **RAND_MAX** (32767). Header: <**stdlib.h**>
- Use the ***void srand(unsigned int)*** function to seed the pseudorandom-number generator before calling ***rand()***.

```
#include "stdafx.h"
#include <stdlib.h>
#include <time.h>
int _tmain(int argc, _TCHAR* argv[])      // C3_1_Rand
{
int nRandNum,a,b,c,nMin,nMax;
  srand((unsigned)time(NULL));
  nRandNum=rand();
  a=rand();
  b=rand();
  printf("%d+%d=",a,b);
  return 0;
}
```

rand() Library Functions

```
#include "stdafx.h"
#include <stdlib.h>
#include <time.h>
int _tmain(int argc, _TCHAR* argv[])      // C3_1_Rand
{
int nRandNum,a,b,c,nMin,nMax;
  srand((unsigned)time(NULL));
  nRandNum=rand();
  a=rand();
  b=rand();
  printf("%d+%d=",a,b);
  scanf("%d",&c);
  nMin=400;
  nMax=1000;
  nRandNum=(double)rand()/(RAND_MAX+1)*(nMax-nMin+1)+nMin;
  nRandNum=(double)rand()/(RAND_MAX+1)*(nMax-nMin+1)+nMin;
  nRandNum=(double)rand()/(RAND_MAX+1)*(nMax-nMin+1)+nMin;
  return 0;
}
```

3.2 Single Character Data(page90)

- Character type, ***char***, refers to more than just lowercase and uppercase letters
- Graphic characters such as ! (exclamation/sigh), # (sharp), and ^(caret).
- Escape sequences (like \n and \r) also are regarded as single characters

Character(page92)

- Declared with
  char variable1, variable2, variable3, . . .;

  ***char* c1, c2;**

- Assignment of ***char***
  enclose the value(constant) in **single quotes**

  c1 = 'g';

- Complete list of ANSI C character called ASCII (American Standard Code for Information Interchange, “askey”) Page94

ASCII Table(page94)

Chara cter	ASCII value	Chara cter	ASCII value	Chara cter	ASCII value	Chara cter	ASCII value	Chara cter	ASCII value	Chara cter	ASCII value
\a	7	+	43	,	60	N	78	_	95	p	112
\b	8	,	44	=	61	O	79	‘	96	q	113
\t	9	-	45	.	62	P	80	a	97	r	114
\n	10	.	46	?	63	Q	81	b	98	s	115
\v	11	/	47	A	65	R	82	c	99	t	116
\f	12	0	48	B	66	S	83	d	100	u	117
\r	13	1	49	C	67	T	84	e	101	v	118
space	32	2	50	D	68	U	85	f	102	w	119
!	33	3	51	E	69	V	86	g	103	x	120
“	34	4	52	F	70	W	87	h	104	y	121
#	35	5	53	G	71	X	88	i	105	z	122
%	37	6	54	H	72	Y	89	j	106	{	123
&	38	7	55	I	73	Z	90	k	107	|	124
,	39	8	56	J	74	[	91	l	108	}	125
(	40	9	57	K	75	\	92	m	109	~	126
)	41	:	58	L	76	]	93	n	110		
*	42	;	59	M	77	^	94	o	111		

Character Processing(page93)

- C regards escape sequences as a single character

  c3 = '\n';

- To print characters

  char c1='x',c2='p';

  printf ("%**c**,%c \n", c1, c2);

- C treats characters with their integer values

  printf ("%**c**,%d\n", c1, c1);

  the ASCII value 120 is printed

putchar **Function**(page95)

- Prints the character(its argument), to the standard output device (the screen)

putchar **(character);**

putchar ('y');

putchar(32);

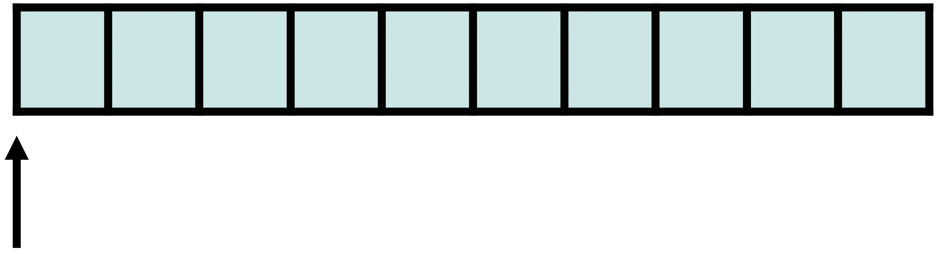
```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{        //C3_2
char   c1='x',c2='y';
  printf("%c,%c\n",c1,c2);
  printf("%c,%d\n",c1,c1);
  putchar(c1);
  putchar('\n');
  putchar(121);
  return 0;
}
```

Read Characters from Keyboard(page96)

- Using the ***scanf*** with **%c**
  scanf **("%c%c", &c3, &c4);**
- Using ***getchar*** (header file <conio.h>)
  c5 = ***getchar*****();**
- Note that <conio.h> is actually out of scope of ANSI C
- Both functions works with the ***input buffer*** to get the information typed at the keyboard

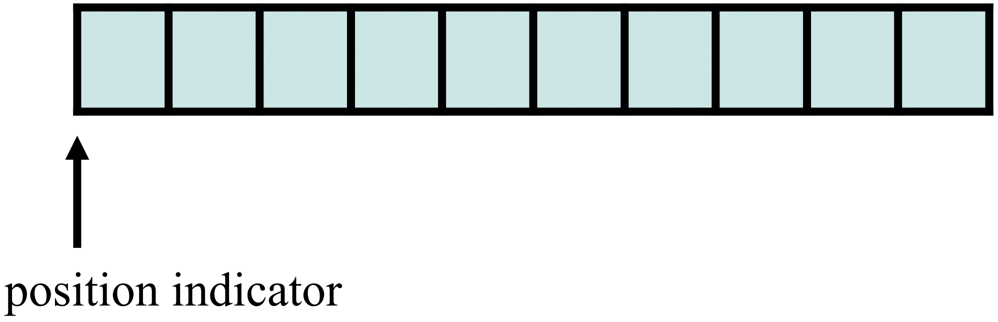
Read Characters from Keyboard(page96)

- A *buffer* is a portion of memory reserved for temporarily holding information
  - Accessed sequentially, i.e., one memory cell after another is read
  - A position indicator keeps track of the point at which no further information has been read



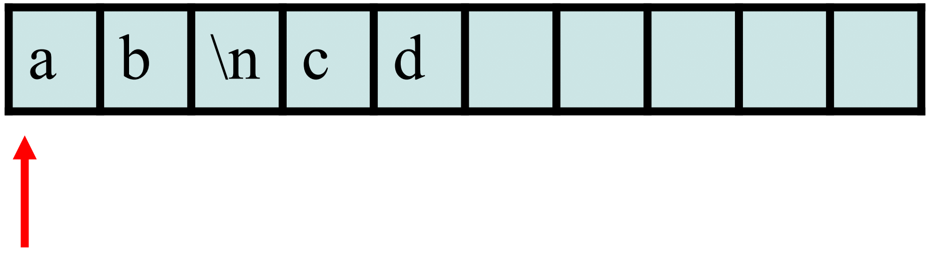
Read Characters from Keyboard

- On reading a cell, position indicator advances one cell
- ***getchar*** function works with buffer position indicator to retrieve next character in buffer
- When input function is called, it either reads item in next cell or it stops execution and waits
- input function is reactivated when ***Return*** or ***Enter*** key is pressed



Read Characters from Keyboard

- The first keys pressed were **ab**<*return* >
- The buffer will have



a	b	\n	c	d					

position indicator

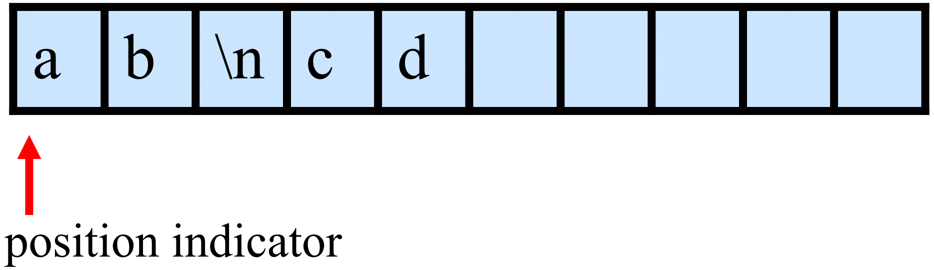
- As ***scanf*** function has read both **a** and **b** so the position indicator is located as shown

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{               // C3_2.cpp
char       c3,c4,c5,c6,c7,c8;

  scanf("%c%c",&c3,&c4);
  c5=getchar();
  c6=getchar();

  fflush(stdin);
  c7=getchar();
  c8=getchar();

  return 0;
}
```



a	b	\n	c	d					

position indicator

Read Characters from Keyboard

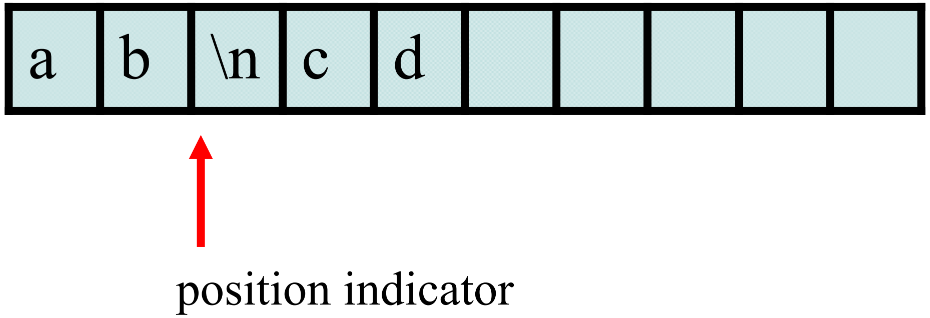
- Then next statements are

  c5 = ***getchar***();

  c6 = ***getchar***();

- ***getchar***(); is executed – reads \n from the buffer
- ***fflush()*** can flush or empty the buffer after obtaining the character(s) of interest

  fflush**(stdin);**



a	b	\n	c	d					

position indicator

scanf and Numeric Input Data

- White space in the input stream is treated differently with numeric input

  scanf **("%d%d%f", &a1, &a2, &a3);**

  would accept the input

  1 2
  3.14159

  as a1=1, a2=2, a3=3.14159.

- The input

  1 2 3.14159

- Would produce the same assignments

scanf and Numeric Input Data

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{       //C3_2
int     nX1,nX2;
float   fY1;

  scanf("%d%d%f",&nX1,&nX2,&fY1);
  printf("nX1=%d,nX2=%d,fY1=%f\n",nX1,nX2,fY1);
  return 0;
}
```

More about how scanf() works and problems when dealing with character data (Page 100-101)

3.3 File Processing(page105)

- Typical file processing procedures:
  - Opening a file with a given file name
    - When program is being executed, it will search for a file with that name. If that file does not exist, your program won't execute
  - Reading/Writing data from/to a file
    - Read using the ***fscanf***( ) function
    - Write using the ***fprintf***()
  - Close the file

(1) *Open File*

- First, we need to declare a file pointer variable holding address to the file(page107)

  FILE *pFilePointer;

  - **FILE** is a data type
  - pFilePointer is the file pointer

- Then, use ***fopen()*** to open a file, meaning to create a link between a disk file and a file pointer (page108)

  file_pointer = *fopen* (file_name, access_mode);

  - **file_name** and the ***access_mode*** are in string literals (page110)

  pFilePointer = ***fopen*** ("D:\\text1.txt","r");

  - "D:\\text1.txt" is file name to be opened
  - "**r**" means that we want to read data. We can use "**w**" for write access

(2) ***Reading from File***(page107)

- Use *fscanf*() to read data from a file
  ***fscanf*(file_pointer, format_string, argument_list);**
  fscanf(pFilePointer,"%d,%f",&nX1,&fY1)**;**
- Reads the contents of the file indicated by file_pointer according to the conversion specifications in format_string.This is similar to ***scanf()***.

(3) ***Close File*** (page109)

- Although C will automatically close all open files after execution, it is still recommended to close a file manually, using the ***fclose()***

  fclose**(file_pointer);**

  example

  fclose**(**pFilePointer**);**

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{//C3_3_FileProcess
int       nX1,nX2,nX3,nX4;
float     fY1,fY2,fY3,fY4;
FILE*     pFilePointer;
pFilePointer=fopen ("D:\\text1.txt","r");
fscanf(pFilePointer,"%d,%f",&nX1,&fY1);
fscanf(pFilePointer,"%d,%f",&nX2,&fY2);
fscanf(pFilePointer,"%d,%f",&nX3,&fY3);
fscanf(pFilePointer,"%d,%f",&nX4,&fY4);
fclose(pFilePointer);
printf("%-8d%-10.6f\n",nX1,fY1);
printf("%-8d%-10.6f\n",nX2,fY2);
printf("%-8d%-10.6f\n",nX3,fY3);
printf("%-8d%-10.6f\n",nX4,fY4);
return 0;
}
```

D:\text1.txt

```
101,314.15
02,3.145
11653,31415.3986
-104,0.3145
```

Writing to File (page113)

- Use the ***fprintf***() function to write data to a file.
  fprintf**(file_pointer, format_string, argument_list);**

  example
  fprintf**(pFilePointer,"Week=%5d,Year=%5d\n",week, year);**

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{//C3_3_FileProcess
int         nX1,nX2,nX3,nX4;
float       fY1,fY2,fY3,fY4;
FILE*       pFilePointer;
  pFilePointer=fopen ("D:\\text2.txt","w");
  nX1=201;fY1=3.14;
  fprintf(pFilePointer,"nX1=%-8d,fY1=%-10.6f\n",nX1,fY1);
  nX1++;fY1++;
  fprintf(pFilePointer,"nX1=%-8d,fY1=%-10.6f\n",nX1,fY1);
  nX1++;fY1++;
  fprintf(pFilePointer,"nX1=%-8d,fY1=%-10.6f\n",nX1,fY1);
  nX1++;fY1++;
  fprintf(pFilePointer,"nX1=%-8d,fY1=%-10.6f\n",nX1,fY1);

  fclose(pFilePointer);

  return 0;
}
```

D:\text2.txt

```
nX1=201     ,fY1=3.140000
nX1=202     ,fY1=4.140000
nX1=203     ,fY1=5.140000
nX1=204     ,fY1=6.140000
```

Home Work

- Page111-2
- Page112-3(Program)
- Page114-2
- Page114-3(Program)
- Page119-3.1(Program)
- First, generate 10 Random integer numbers from 10 to 20 to a file. Then read these numbers from the file and print them out.