

Basic Computation

Objectives

After studying this section, you should be able to:

- ◆ Understand what is a data type
- ◆ Declare constants and variables of a program
- ◆ Express operations on data

Contents

◆ Variables and Data types

- ◆ Data Types
- ◆ Integral Types
- ◆ Floating-Point Types
- ◆ Declarations

◆ Basic Memory Operations

- ◆ Literals
- ◆ Constants
- ◆ Assignment Operator
- ◆ Output
- ◆ Input

◆ Expressions

- ◆ Arithmetic,
- ◆ Relational,
- ◆ Logical
- ◆ Bit Operators
- ◆ Shorthand Assignment Operators
- ◆ Mixing Data Types
- ◆ Casting
- ◆ Precedence

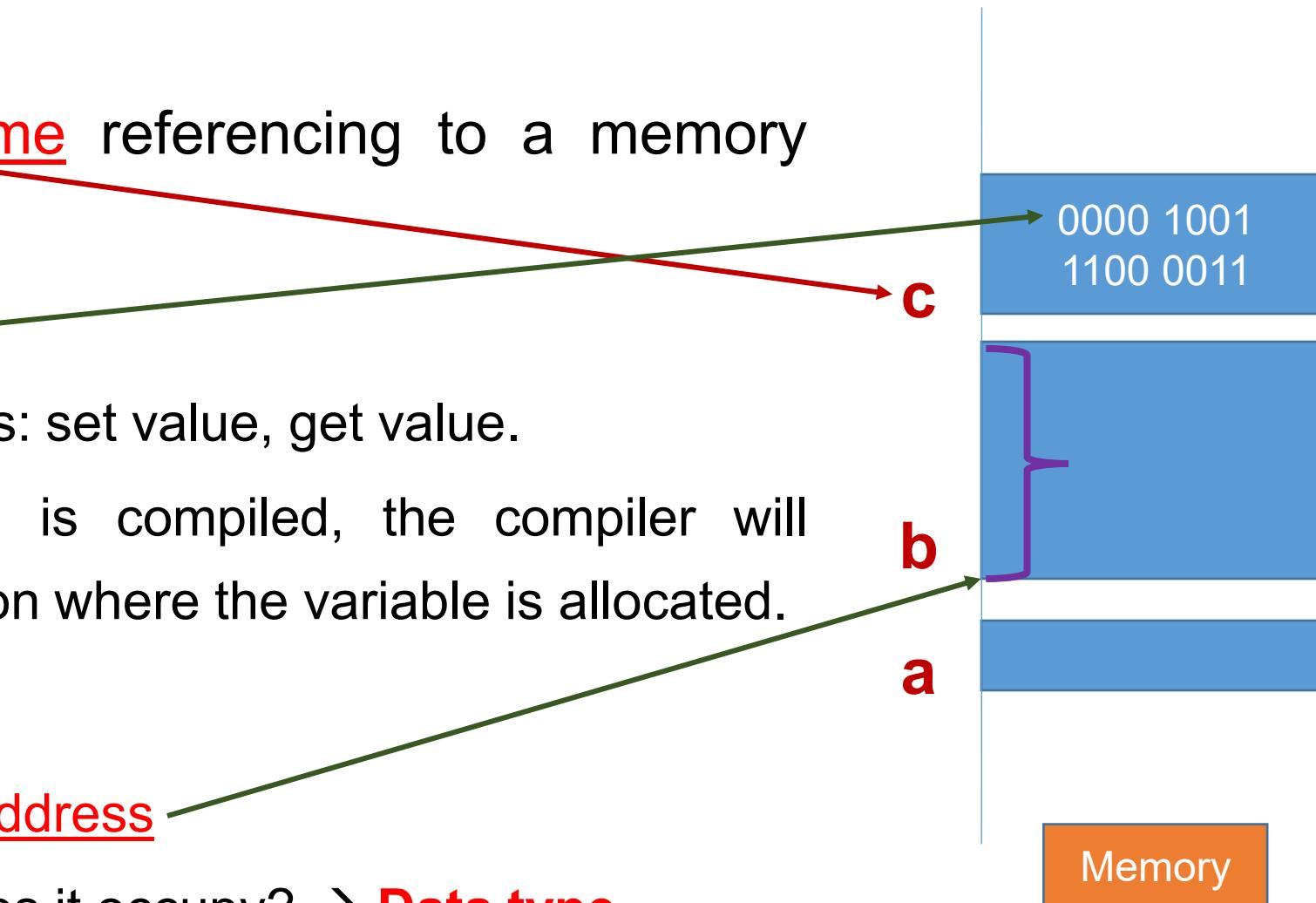
Variables and Data types

Introduction

- ◆ Instruction: A task that hardware must perform on data.
- ◆ Data can be: **constants, variables**.
- ◆ **Constants**: Fixed values that can not be changed when the program executes.
- ◆ **Variables**: Values can be changed when the program execute.
- ◆ Data must be stored in the main memory (RAM).
- ◆ 2 basic operations on data are **READ** and **WRITE**.
- ◆ Numerical data can participate in expressions.

Variables

- ◆ A variable is a name referencing to a memory location (address)
 - Holds binary data
 - Two basic operations: set value, get value.
 - When the program is compiled, the compiler will determine the position where the variable is allocated.
- ◆ Questions:
 - Where is it? → It's Address
 - How many bytes does it occupy? → Data type



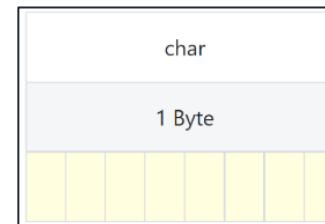
Data Types

- ◆ The C language associates a data type with each variable. Each data type occupies a compiler-defined number of bytes.
- ◆ A data type defines:
 - How the values are stored and
 - How the operations on those values are performed.
- ◆ Typed languages defined some primitive data types.

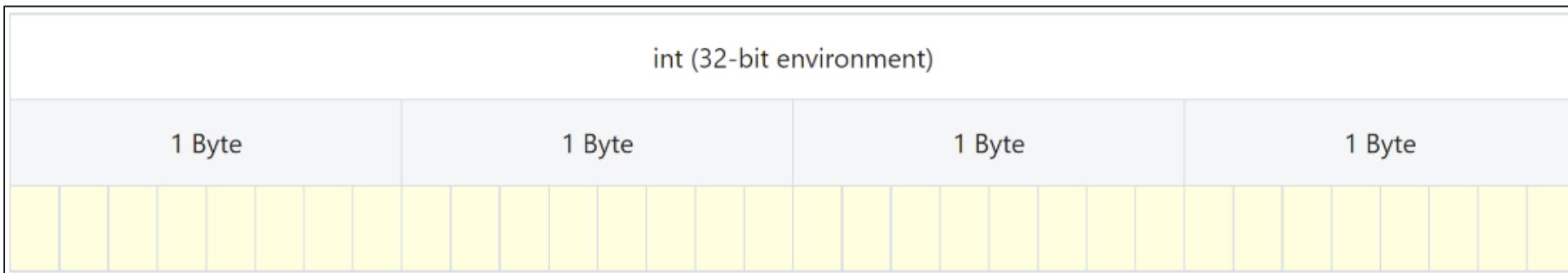
Arithmetic Types

- The four most common types of the C language for performing arithmetic calculations are: **char, int, float, double**

char: Occupies one byte and can store a small integer value, a single character or a single symbol:

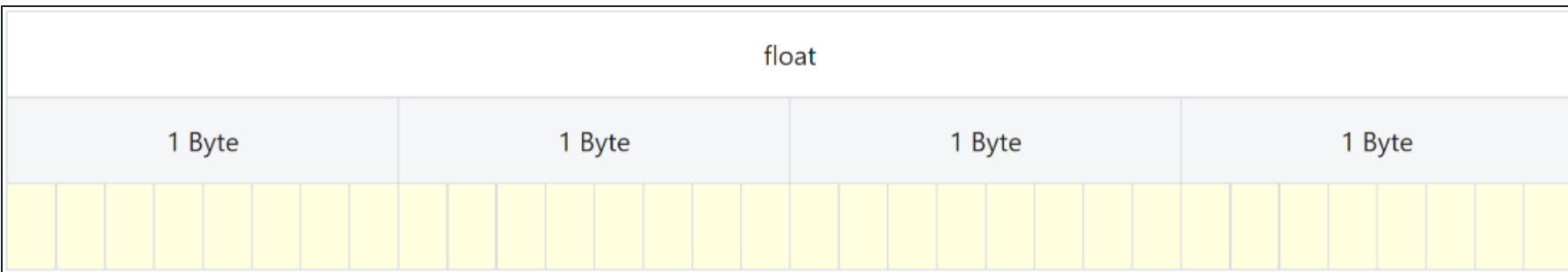


int: occupies one word and can store an integer value. In a 32-bit environment, an **int** occupies 4 bytes:

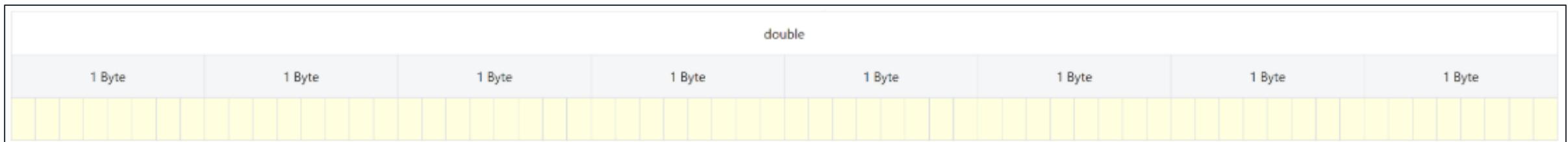


Arithmetic Types (cont.)

float: typically occupies 4 bytes and can store a single-precision, floating-point number:



double: typically occupies 8 bytes and can store a double-precision, floating-point number:

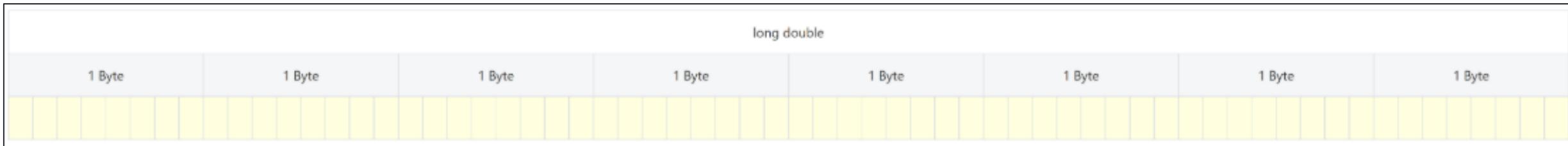


int Type Size Specifiers

- ◆ Specifying the size of an int ensures that the type contains a minimum number of bits. The three specifiers are:
 - **short**: at least 16 bits
 - **long**: at least 32 bits
 - **long long**: at least 64 bits
- ◆ *Standard C does not specify that a long double must occupy a minimum number of bits, only that it occupies no fewer bits than a double.*

double Type Size Specifier

- The size of a **long double** depends on the environment and is typically at least 64 bits:



- long double** type only ensures that it contains at least as many bits as a double

const Qualifier

- ◆ Any type can hold a constant value.
- ◆ A constant value cannot be changed.
- ◆ To qualify a type as holding a constant value we use the keyword **const**
- ◆ A type qualified as **const** is unmodifiable.
- ◆ If a program instruction attempts to modify a const qualified type, the compiler will report an error.

Representing Values

- ◆ Hardware manufacturers distinguish **integral** types from **floating-point** types and represent **integral** data and **floating-point** data differently.
 - Integral types: char, int
 - Floating-point types: float, double

Exercise 1:

- Convert the following decimal **integers** to **binary**:

63 0011 1111

219 1101 1011

- Convert the following **binary** notation to **decimal**:

0111 0101 117

0011 1011 59

Negative and Positive Values

- ◆ Computers store negative integers using encoding schemes:
 - Two's complement notation,
 - One's complement notation, and
 - Sign magnitude notation.
- ◆ All of these schemes represent non-negative integers identically.
- ◆ The most popular scheme is two's complement.
- ◆ To obtain the two's complement of an integer, we:
 - Flip the bits (1-complement)
 - Add one → 2-complement

Two's complement notation

- ◆ To obtain the two's complement of an integer, we
 - Flip the bits
 - Add one
- ◆ For example, we represent the integer **-92** by **10100100₂**

Bit #	7	6	5	4	3	2	1	0
92 =>	0	1	0	1	1	1	0	0
Flip Bits	1	0	1	0	0	0	1	1
Add 1	0	0	0	0	0	0	0	1
-92 =>	1	0	1	0	0	1	0	0

Exercise 2: Use signed 1-byte integral number

- ◆ What is the two's complement notation of

-63 1100 0001

-219 0010 0101

- ◆ Convert the following binary notation to decimal:

1111 0101 -11

1011 1011 -69

Unsigned Integers

- ◆ We can use all of the bits available to store the value of a variable.
- ◆ With unsigned variables, there is no need for a negative-value encoding scheme.

Type	Size	Min	Max - 32 bit	Max - 16 bit
<code>unsigned short</code>	≥ 16 bits	0	65,535	
<code>unsigned int</code>	1 word	0	4,294,967,295	65,535
<code>unsigned long</code>	≥ 32 bits	0	4,294,967,295	
<code>unsigned long long</code>	≥ 64 bits	0	18,446,744,073,709,551,615	

Cultural Symbols (characters)

- ◆ We store cultural symbols using an integral data type.
- ◆ We store a symbol by storing the integer associated with the symbol.
- ◆ Over 60 encoding sequences have already been defined.

Encoding Sequence	Full Name	# Bits	Defined In
UCS-4	Universal Multiple-Octet Coded Character Set	32	1993
BMP	Basic Multilingual Plane	16	1993
Unicode	Unicode	16	1991
ASCII	American Standard Code for Information Interchange	7	1963
EBCDIC	Extended Binary Coded Decimal Interchange Code	8	1963

We use the ASCII encoding sequence throughout this course

ASCII table for characters

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

Exercise 3:

- ◆ You are tasked to write a program to manage a small library system. Choose the most suitable **data type** for each variable based on the information provided.
- ◆ The library needs to store the following information:
 - *Total number of books in the library* (e.g., 10,000).
 - *Average price of books in dollars* (e.g., 15.75).
 - *A single character indicates the book's genre* ('F' for Fiction, 'N' for Non-fiction, etc.).
 - *The unique ID of a book* (e.g., 2345678901).
 - *Whether a book is available or not* (1 for available, 0 for not available).
 - *Number of pages in a book* (e.g., 500).
- ◆ **Tasks:**
 - Choose an appropriate data type for each variable and explain your choice.
 - Declare variables using the chosen data types and initialize them with sample values.
 - Print all the values along with their data types.

The range of values of data types

C Basis Data Types	32-bit CPU		64-bit CPU	
	size (bytes)	Range	size (bytes)	Range
char	1	-128 to 128	1	-128 to 128
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long long	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	3.4E +/-38	4	3.4E +/-38
double	8	1.7E +/-308	8	1.7E +/-308

Declaring (Creating) Variables

- ◆ To create a variable, specify the **data type** and assign it a **value**:

- ◆ **Syntax:** ***data_type identifier [= initial value];***

- ◆ For example:

```
char section  
int numberOfClasses;  
double cashFare = 2.25;
```

- ◆ Rules for naming variables are:

- Names can contain letters, digits and underscores.
- Names must begin with a letter or an underscore (_)
- Names are case-sensitive (**myVar** and **myvar** are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Must not be a C reserved word

Reserved words

The C reserved words are

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

For upward compatibility with C++, we also avoid using the C++ reserved words

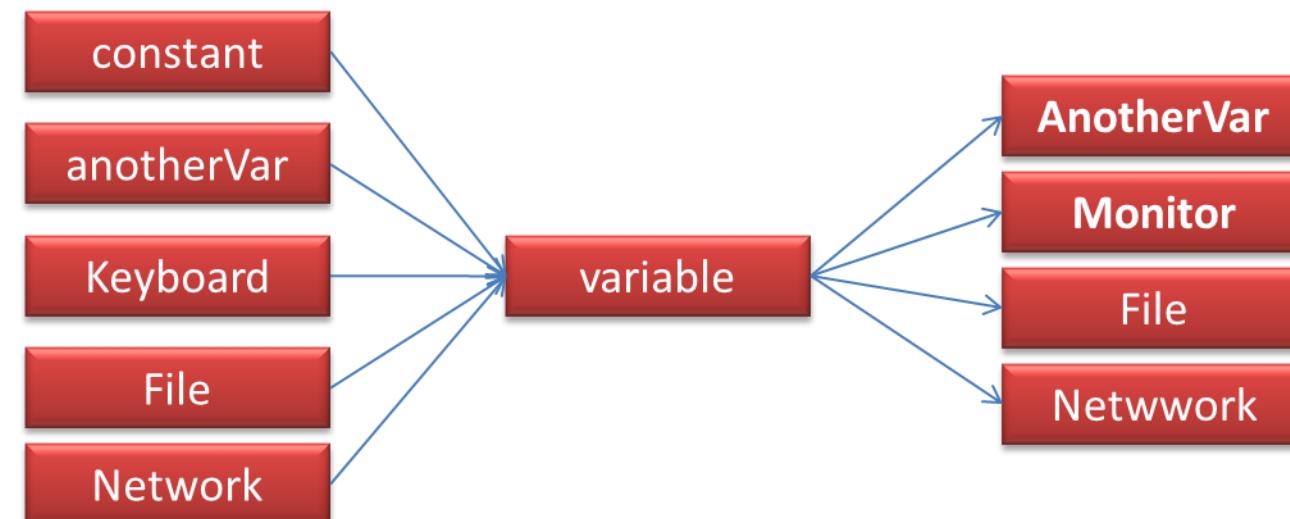
asm	export	private	throw
bool	false	protected	true
catch	friend	public	try
class	mutable	reinterpret_cast	typeid
const_cast	namespace	static_cast	typename
delete	new	template	using
dynamic_cast	operator	this	virtual
explicit			wchar_t

Exercise 4:

- ◆ You are tasked to create a C program that calculates the area and perimeter of a rectangle. As part of this exercise, ensure that all variable names:
 1. Follow the **rules for valid naming** in C.
 2. Are **descriptive and meaningful** to improve readability.
 3. Follow common **naming conventions** (e.g., camelCase or snake_case).
- ◆ **Tasks:**
 1. Identify the rules for valid variable names in C and apply them.
 2. Choose appropriate and meaningful variable names for the program.
 3. Write the program using the chosen names.

Some operations on variables:

- ◆ Assign a constant value to a variable,
- ◆ Assign the value of another variable to a variable,
- ◆ Output the value of a variable,
- ◆ Input a fresh value into a variable's memory location.



Example 1: Variable & Data Types

```
1 #include <stdio.h>
2
3 int main(){
4     // Create a variable called myNum of type int and assign the value 15 to it:
5     int myNum1 = 15;
6
7     // Or:
8     // Declare a variable
9     int myNum2;
10    // Assign a value to the variable
11    myNum2 = 15;
12
13    printf("myNum1 = %d\n", myNum1);
14    printf("myNum2 = %d", myNum2);
15
16    return 0;
17 }
```

Output:

```
myNum1 = 15
myNum2 = 15
-----
Process exited after 0.06531 seconds
```

Example 2: Variable & Data Types

```
variable_demo.c  x
1 #include <stdio.h>
2
3 int main(){
4     // Declare and initialize variables with data types: char, int, long, float, double
5     char charVar = 'A';
6     int integerVar = 10;
7     long longVar = 1000;
8     float floatVar = 13.5f;
9     double doubleVar = 100.15;
10
11    // Print table format
12    printf("%-15s %-15s %-15s %-15s\n", "Name", "Address", "Size (bytes)", "Value");
13    printf("-----\n");
14
15    // Print table rows includes: Name Address Size Value
16    printf("%-15s %-15u %-15d %-15c\n", "charVar", &charVar, sizeof(charVar), charVar);
17    printf("%-15s %-15u %-15d %-15d\n", "integerVar", &integerVar, sizeof(integerVar), integerVar);
18    printf("%-15s %-15u %-15d %-15ld\n", "longVar", &longVar, sizeof(longVar), longVar);
19    printf("%-15s %-15u %-15d %-15f\n", "floatVar", &floatVar, sizeof(floatVar), floatVar);
20    printf("%-15s %-15u %-15d %-15lf\n", "doubleVar", &doubleVar, sizeof(doubleVar), doubleVar);
21
22    return 0;
23}
```

Where are variables stored and how many bytes do they occupy?

Conversion Specifiers
& String format

Name	Address	Size (bytes)	Value
charVar	6684191	1	A
integerVar	6684184	4	10
longVar	6684180	4	1000
floatVar	6684176	4	13.500000
doubleVar	6684168	8	100.150000

Process exited after 0.06661 seconds with return value 0
Press any key to continue . . .

The operator **&** will get the address of a variable or code.

The operator **sizeof(var/type)** return the size (number of byte) occupied by a variable/type

Questions as Summary

- ◆ What is a variable?
- ◆ What is a data type?
- ◆ Characteristics of a data type are and ...
- ◆ The size of the int data type is Bytes.
- ◆ Choose the wrong declarations:

int n = 10;

char c1, c2 = 'A';

int m = 19; k = 2;

char c3; int t;

Float f1; f2 = 5.1;

- ◆ Explain little-endian ordering and big-endian ordering.

Basic Memory Operations

1. Literals

```
2 #include <stdio.h>
3 int main()
4 {   char c1= 'A'; ←
5     printf("Hello everyone");
6     int n= 258; ←
7     getchar();
8     return 0;
9 }
```

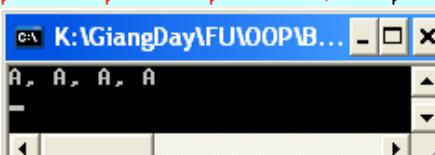
- ◆ Constant values are specified directly in the source code.
- ◆ They can be
 - Character literals (constant characters)
 - String literals(constant strings)
 - Number literals (constant numbers)

Literals: Characters, Strings

- ◆ 4 ways for representing a character literal:

- Enclose the character single quotes → 'A'
- Decimal ASCII code of the character → 65 for 'A'
- Octal ASCII code of the character → 0101 for 'A',
- Hexadecimal ASCII code of the character → 0x41 for 'A',

```
2 #include <stdio.h>
3 int main()
4 {    char c1= 'A';
5     char c2= 65;
6     char c3= 0x41;
7     char c4= 0101;
8     printf("%c, %c, %c, %c\n", c1, c2, c3, c4);
9     getchar();
10    return 0;
11 }
```



Assign value to a variable:
The operator =

Literals: Escape Sequences

- ◆ Pre-defined literals for special actions:

Character	Sequence	ASCII	EBCDIC
alarm	\a	7	47
backspace	\b	8	22
form feed	\f	12	12
newline	\n	10	37
carriage return	\r	13	13
horizontal tab	\t	9	5
vertical tab	\v	11	11
backslash	\\\	92	*
single quote	\'	39	125
double quote	\"	34	127
question mark	\?	63	111

Literals: Escape Sequences (cont.)

```
1 /* Test ESCAPE sequences */
2 #include <stdio.h>
3 int main()
4 {   printf("\a");
5   printf("He said that "I love you"\n");
6   printf("She says 'A'\n");
7   printf("My file: C:\t1\t111\new_year.txt");
8   getchar();
9   return 0;
10 }
```

Error! Why?

```
1 /* Test ESCAPE sequences */
2 #include <stdio.h>
3 int main()
4 {   printf("\a");
5   printf("He said that \"I love you\"\n");
6   printf("She says 'A'\n");
7   printf("My file: C:\t1\t111\new_year.txt");
8   getchar();
9   return 0;
10 }
```

Modify
then run it

Change \ to \\ then run it

Literals: Numbers

- ◆ *The compiler will convert directly numeric literals (constants) to binary numbers and put them in the executable file.* → How long of binary constants? → They depend on their data types specified by programmers.
- ◆ Default: Integral value → **int**, real number → **double**
- ◆ Specifying data type of constants: **Suffixes after numbers.**

Type	Size	Suffix	Example
int	1 word	none	1456234
long	32 bits	L or l (ell)	75456234L
long long	64 bits	LL or ll (ell ell)	75456234678LL
unsigned		u or U	75456234U
double	2 words	none	1.234
float	32 bits	F or f	1.234F

2. Named Constants

- ◆ Use **const** keyword

Syntax:

const data_type constant_name = value;

- ◆ For example:

```
const_keyword.c  × |  
1 #include <stdio.h>  
2  
3 int main() {  
4     const float pi = 3.1415926; // Declaring a constant for Pi  
5     float radius, area;  
6  
7     printf("Enter the radius of the circle: ");  
8     scanf("%f", &radius);  
9  
10    area = pi * radius * radius; // Using the constant 'pi'  
11  
12    printf("The area of the circle is %f\n", area);  
13  
14    return 0;  
15 }
```

Output:

```
D:\MonHoc\PRF192\ThucHanh\const_keyword.exe  
Enter the radius of the circle: 26.02  
The area of the circle is 2126.985107  
-----  
Process exited after 21.55 seconds with return value 0  
Press any key to continue . . .
```

2. Named Constants (cont.)

- ◆ Use the pre-processor (pre-compiled directive) **#define**

Syntax:

#define constant_name value

- ◆ For example:

```
define_directive.c  × |  
1 #include <stdio.h>  
2  
3 #define PI 3.1415926 // Defining a constant for Pi  
4  
5 int main() {  
6     float radius, area;  
7  
8     printf("Enter the radius of the circle: ");  
9     scanf("%f", &radius);  
10  
11    area = PI * radius * radius; // Using the constant 'PI'  
12  
13    printf("The area of the circle is %f\n", area);  
14  
15    return 0;  
16 }
```

Output:

```
D:\MonHoc\PRF192\ThucHanh\define_directive.exe  
Enter the radius of the circle: 26.02  
The area of the circle is 2126.985107  
-----  
Process exited after 16.56 seconds with return value 0  
Press any key to continue . . .
```

2. Named Constants (cont.)

- Attention when the directive **#define** is used:
 - The compiler will not allocate memory block for values but all pre-defined names in the source code will be replaced by their values before the translation performs (The MACRO REPLACEMENT)
 - A name is call as a MACRO.

The screenshot shows a code editor with the following C code:

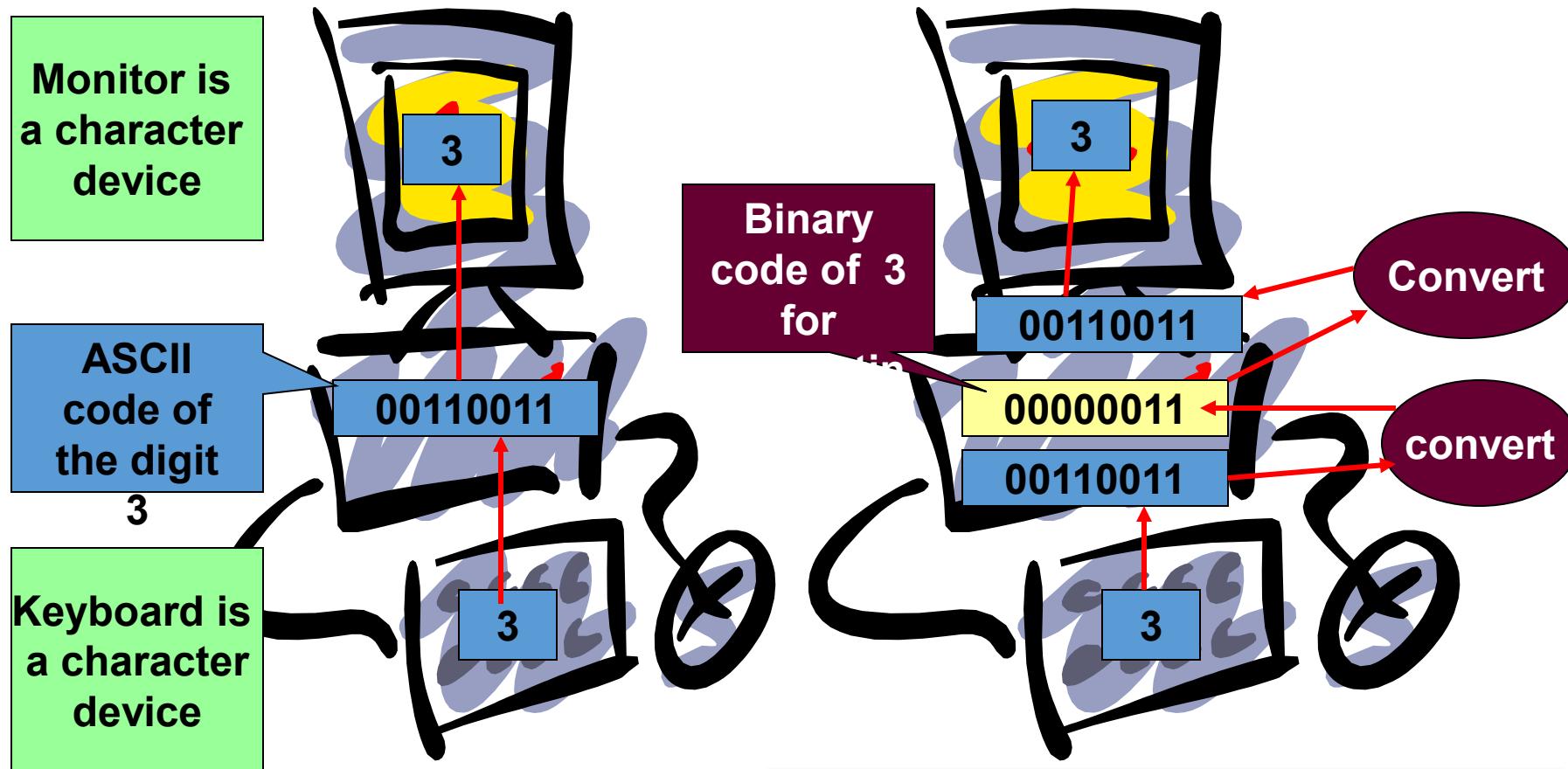
```
1 #include <stdio.h>
2
3 #define PI = 3.14; // Defining a constant for Pi
4
5 int main() {
6     printf("%lf", PI*3*3);
7     getchar();
8     return 0;
9 }
```

A green box highlights the line `#define PI = 3.14;` with the word "replace" written above it in red. A blue arrow points from this box to the same line in the code editor. The code editor has tabs for Compiler (3), Resources, Compile Log, Debug, Find Results, Console, and Close. Below the tabs is a message window:

Line	Col	File	Message
D:\MonHoc\PRF192\ThucHanh\define_di...	In function 'main':		
3	12	D:\MonHoc\PRF192\ThucHanh\define_di...	[Error] expected expression before '=' token
6	19	D:\MonHoc\PRF192\ThucHanh\define_di...	[Note] in expansion of macro 'PI'

At the bottom of the message window, there are status fields: Line: 3 Col: 12 Sel: 0 Lines: 11 Length: 152 Insert.

3. Input/ Output variables



Conversion rules are pre-defined in C

Conversion Specifiers

Specifier	Output As A	Use With Data Type
%c	character	char
%d	decimal	char, int
%u	decimal	unsigned int
%o	octal	unsigned char, int, short, long
%x	hexadecimal	unsigned char, int, short, long
%hd	short decimal	short
%ld	long decimal	long
%lld	very long decimal	long long
%f	floating-point	float
%lf	floating-point	double
%le	exponential	double

Example 1:

```

1 /*Study_in_output.c */
2 #include <stdio.h>
3 int n;
4 int main()
5 { int m;
6   printf("Var. n, add:%u\n", &n);
7   printf("Var. m, add:%u\n", &m);
8   printf("main code, add:%u\n", &main);
9   printf("Enter 2 integers:");
10  scanf("%d%d", &n, &m);
11  printf("Values entered: n=%d, m=%d\n", n,m);
12  getchar();
13  getchar();
14  return 0;
15 }
  
```

Format
string

4210784

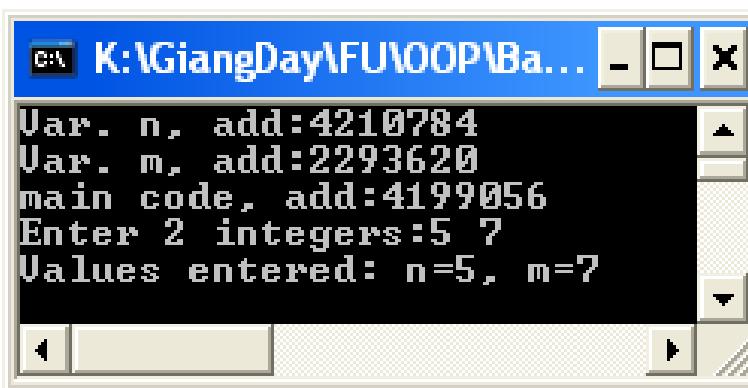
n

4199056

main

2293620

m



scanf("%d%d", &n, &m) →

scanf("%d%d", 4210784, 2293620) means that get keys pressed then change them to decimal integers and store them to memory locations 4210784, 2293620.

Example 2:

```
2 #include <stdio.h>
3 int main()
4 {    int n;
5     double x, y;
6     printf("Enter an integer:");
7     scanf("%d", &n);
8     printf("Enter 2 real numbers:");
9     scanf("%lf%lf", &x, &y);
10    double num1, num2;
11    char op;
12    printf("Enter an expression + - * / :");
13    scanf("%lf%c%lf", &num1, &op, &y);
14    printf("Expression inputted is: %lf%c%lf\n", num1, op, num2);
15    getchar();
16    getchar();
17    return 0;
18 }
```

Format string

Input a value to a variable:

scanf ("input format", &var1, &var2,...)

Output the value of a variable:

printf ("output format", var1, var2,...)

```
Enter an integer:12
Enter 2 real numbers:1.23 7.809
Enter an expression + - * / :6.5-12.9
Expression inputted is: 6.500000-0.000000
```

```
Enter an integer:12
Enter 2 real numbers:1.23
7.809
Enter an expression + - * / :6.5-12.9
Expression inputted is: 6.500000-0.000000
```

The function **scanf** receive the BLANK or
ENTER KEYS as separators.



Data holders

Question

- ◆ Explain means of parameters of the **scanf(...)** and the **printf(...)** functions.
- ◆ Use words “left” and “right”. The assignment `x=y;` will copy the value in the side to the side.

Exercise 5:

- ◆ Develop a C program in which 2 integers, 2 float numbers and 2 double numbers are declared. Ask user for values of them then print out values and addresses of them. Write down the memory map of the program.
- ◆ Run the following program:

```
2 #include <stdio.h>
3 int main()
4 {    int n;
5     char c;
6     printf("Enter an integer:");
7     scanf("%d",&n);
8     printf("Enter a character:");
9     scanf("%c",&c);
10    getchar(); ←
11    return 0;
12 }
```

Why user do not have a chance to stroke the ENTER key before the program terminate?

Modify and re-run:
getchar();

Expressions

Expressions

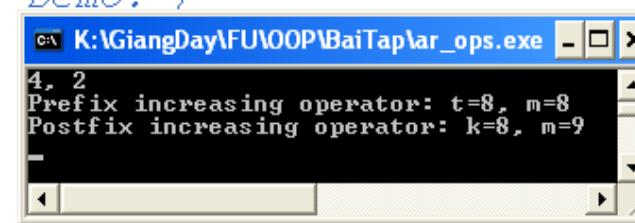
- ◆ **Expression is a valid association of constants, variables, operators and functions and returns an only result.**
- ◆ Examples:
 - $32-x+y/6$ $16.5 + 4/\sqrt{15} * 17 - 8$
 - $45 > 5*x$ $y = 17 + 6*5/9 - z*z$
- ◆ Hardware for calculating expressions: ALU
- ◆ Operations that can be supported by ALU: Arithmetic, relational and logic operations.

1. Arithmetic Operators

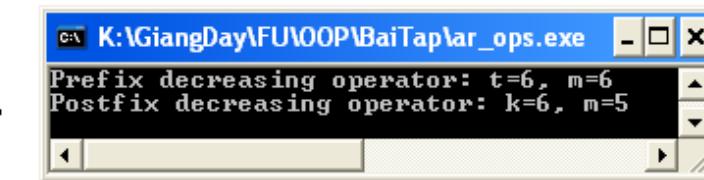
Op.	Syntax	Description	Example
+	+x	leaves the variable, constant or expression unchanged	y = +x ; \leftrightarrow y = x;
-	-x	reverses the sign of the variable	y= -x;
+ -	x+y x-y	Add/subtract values of two operands	z= x+y; t = x-y;
* /	x*y x/y	Multiplies values of two operands Get the quotient of a division	z= x-y; z = 10/3; \rightarrow 3 z = 10.0/3; \rightarrow 3.3333333
%	x%y	Get remainder of a integral division	17%3 \rightarrow 2 15.0 % 3 \rightarrow ERROR
++ --	++x --x x++ x--	Increase/decrease the value of a variable (prefix/postfix operators)	Demo in the next slide.

1. Arithmetic Operators - Example 1

```
1 /*ar_ops.c Arithmetic operators Demo.*/
2 #include <stdio.h>
3 int main()
4 { int n=30, m= 7;
5   printf("%d, %d\n", n/m, n%m);
6   int t= ++m;
7   printf("Prefix increasing operator: t=%d, m=%d\n", t, m);
8   int k= m++;
9   printf("Postfix increasing operator: k=%d, m=%d\n", k, m);
10  getchar();
11  return 0;
12 }
```



```
1 /*ar_ops.c Arithmetic operators Demo.*/
2 #include <stdio.h>
3 int main()
4 { int n=30, m= 7, t, k;
5   t= --m;
6   printf("Prefix decreasing operator: t=%d, m=%d\n", t, m);
7   k= m--;
8   printf("Postfix decreasing operator: k=%d, m=%d\n", k, m);
9   getchar();
10  return 0;
11 }
```



1. Arithmetic Operators - Example 2

```
2 #include <stdio.h>
3 int main()
4 {    int n=30;
5     double x= 5.1;
6     printf("%lf\n", n*x);
7     getchar();
8     return 0;
9 }
```



Explain yourself the output

Line	File	Message
	K:\GiangDay\FU\OOP\BaiTap\ar_o...	In function `main':
6	K:\GiangDay\FU\OOP\BaiTap\ar_o...	invalid operands to binary %

Statistic:

- Multiply > Division
- Integral operations > floating-point ones.

2. Relational Operators

- ◆ For comparisional operators.
- ◆ < <= == >= > !=
- ◆ Return 1: true/ 0: false

```
2 #include <stdio.h>
3 int main()
4 {    int n=30;
5     int x= 5;
6     printf("%d, %d, %d\n", n<x, n==x, n!=x);
7     getchar();
8     return 0;
9 }
```



The terminal window displays the following output:

```
0, 0, 1
```

3. Logical Operators

- ◆ Operator for association of conditions
- ◆ && (and), || (or) , ! (not)
- ◆ Return 1: true, 0: false

```
2 #include <stdio.h>
3 int main()
4 {    int n=30, m= 5;
5     float x= 3.5F, y=8.12F;
6     printf("%d, %d\n", (n<m || x>=y), !(x>y));
7     getchar();
8     return 0;
9 }
```



4. Bitwise Operators

- ◆ **&** (and), **|** (or) , **^** (xor): Will act on a pair of bits at the same position in 2 operands.
- ◆ **<<** Left shift bits of the operand (operands unchanged)
- ◆ **>>** Right shift bits of the operand (operands unchanged, the sign is preserved.)
- ◆ **~** : Inverse bits of the operand.

```

2 #include <stdio.h>
3 int main()
4 { short n=12, m= 8, t=2, k=-1;
5   printf("%d, %d, %d\n", n&m, n|m, n^m);
6   printf("%d, %d\n", n<<1, n<<t);
7   printf("n=%d\n", n);
8   printf("%d, %d\n", n>>1, k>>t);
9   printf("%d\n", ~t);
10  getchar();
11  return 0;
12 }
```

n=12: 0000 0000 0000 1100
m= 8: 0000 0000 0000 1000



n&m

$$\begin{array}{r} 0000\ 0000\ 0000\ 1100 \\ 0000\ 0000\ 0000\ 1000 \\ \hline 0000\ 0000\ 0000\ 1000 \end{array} \rightarrow 8$$

n|m

$$\begin{array}{r} 0000\ 0000\ 0000\ 1100 \\ 0000\ 0000\ 0000\ 1000 \\ \hline 0000\ 0000\ 0000\ 1100 \end{array} \rightarrow 12$$

n^m

$$\begin{array}{r} 0000\ 0000\ 0000\ 1100 \\ 0000\ 0000\ 0000\ 1000 \\ \hline 0000\ 0000\ 0000\ 0100 \end{array} \rightarrow 4$$

4. Bitwise Operators (cont.)

n=12: 0000 0000 0000
1100
n<<1:
0000 0000 0000 1100
0000 0000 0001 100 ← 0
0000 0000 0001 1000 (24_{10})
Left shift 1 bit: multiply by 2

n=12: 0000 0000 0000
1100
n>>1:
Sign: **0**
0000 0000 0000 1100
0000 0000 0000 110
Add the sign to the left"
0000 0000 0000 110 → 6

k=-1:
1: 0000 0000 0000 0001
-1: 1111 1111 1111 1111 (2-complement)
Sign: **1**
1111 1111 1111 1111
111 1111 1 111 1111
Add the sign to the left:
1111 1111 1111 1111 → (-1 $_{10}$)

5. Assignments Operators

- ◆ Variable = expression
- ◆ Shorthand assignments:

Operator	Shorthand	Longhand	Meaning
<code>+=</code>	<code>age += 4</code>	<code>age = age + 4</code>	add 4 to age
<code>-=</code>	<code>age -= 4</code>	<code>age = age - 4</code>	subtract 4 from age
<code>*=</code>	<code>age *= 4</code>	<code>age = age * 4</code>	multiply age by 4
<code>/=</code>	<code>age /= 4</code>	<code>age = age / 4</code>	divide age by 4
<code>%=</code>	<code>age %= 4</code>	<code>age = age % 4</code>	remainder after age/4

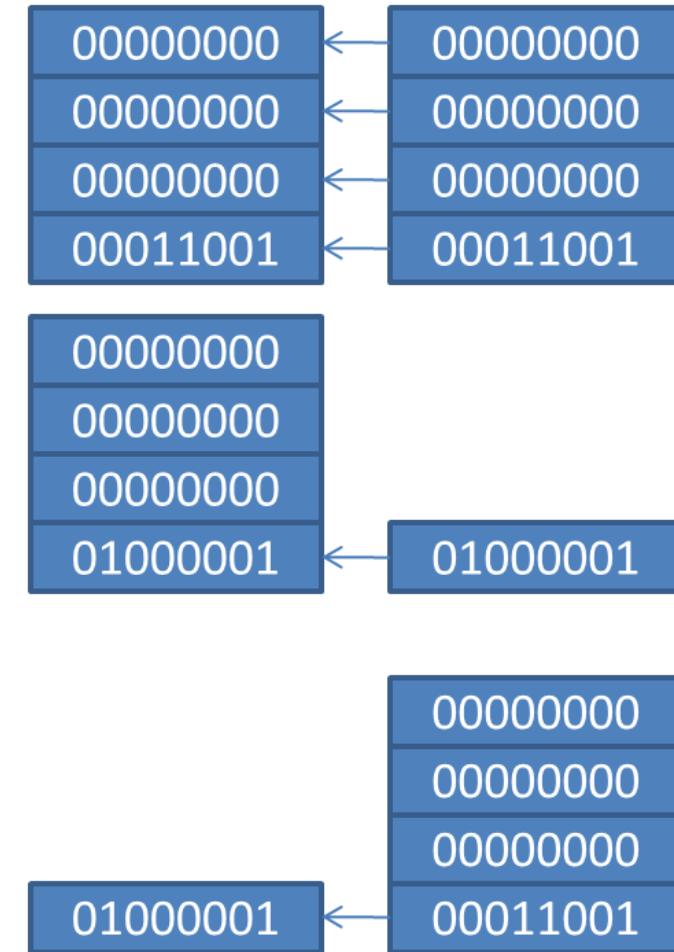
6. Mixing Data Types

- ◆ Although the ALU does not perform operations on operands of differing data type directly, C compilers can interpret expressions that contain operands of differing data type.
- ◆ If a binary expression contains operands of differing type, a C compiler changes the data type of one of the operands to match the other.
- ◆ Data type hierarchy: **double, float, long, int , char** .

6. Mixing Data Types (cont.)

- ◆ Casting Data Type: **x = y;**

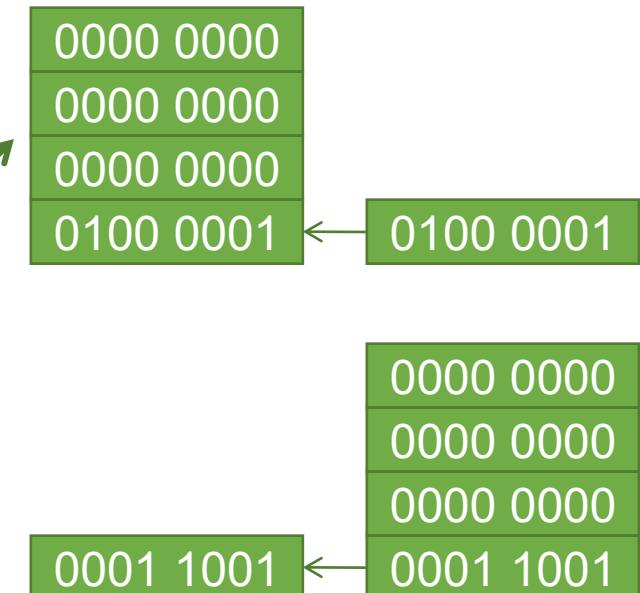
```
#include <stdio.h>
int main()
{   char c1=65, c2;
    int i1=25 , i2;
    printf("Size of int:%d, char:%d byte(s)\n",
           sizeof(int), sizeof(char));
    i2 = i1;
    printf("i2=%d\n", i2);
    i2= c1;
    printf("i2=%d\n", i2);
    c2 = i1;
    printf("c2=%d\n", c2);
    getchar();
    return 0;
}
```



Direction for copying: From the lowest byte to higher bytes

6. Mixing Data Types (cont.)

- ♦ **Implicit Casting for the assignment**
- ♦ If the data type of the variable on the left side of an assignment operator differs from the data type of the right side operand, the compiler:
 - Promotes the right operand to the data type of the left operand **if the left operand is of a higher data type than the right operand**,
 - Truncates the right operand to the data type of the left operand **if the left operand is of a lower data type than the right operand**.



6. Mixing Data Types (cont.)

- Implicit Casting for arithmetic and relational expressions

- If the operands in an arithmetic or relational expression differ in data type, the compiler promotes **the value of lower data type to a value of higher data type** before implementing the operation.

left operand	right operand				
	double	float	int	char	long
double	double	double	double	double	double
float	double	float	float	float	float
int	double	float	int	int	long
char	double	float	int	int	long
long	double	float	long	long	long

Data Type of Promoted Operand

```
int n=3; long t=123; double x=5.3;
3*n      +      620*t      - 3*x
(int*int) + (int*long) - (int*double)
    int + long      - double
    long      - double
                    double
```

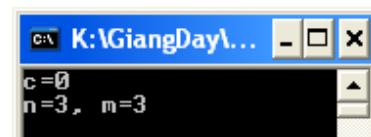
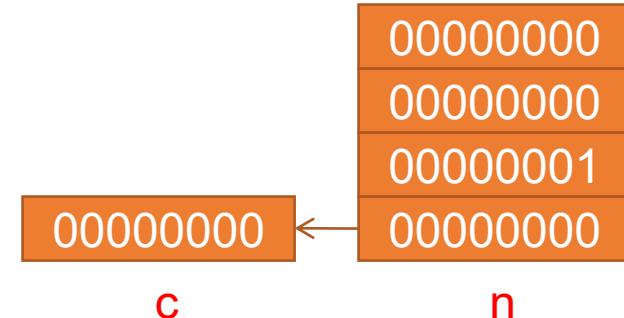
6. Mixing Data Types (cont.)

Explicit Casting

- We may temporarily change the data type of any operand in any expression to obtain a result of a certain data type.

Cast Expression	Meaning
(double)	variable or constant
(float)	variable or constant
(int)	variable or constant
(char)	variable or constant
(long)	variable or constant

```
#include <stdio.h>
int main()
{   int n= 256, m;
    char c;
    c = (char)n;
    printf("c=%d\n", c);
    double x= 3.251;
    n = x;
    m = (int)x;
    printf ("n=%d, m=%d\n", n,m);
    getchar();
    return 0;
}
```



7. Operator Precedence

- In a expression containing some more than one operator. Which operator will perform first? → Pre-defined Precedence.
- We can use () to instruct the compiler to evaluate the expression within the parentheses first

Operator	Evaluate From
<code>++ -- (post)</code>	left to right
<code>++ -- (pre) + - & ! (all unary)</code>	right to left
<code>(data type)</code>	right to left
<code>* / %</code>	left to right
<code>+ -</code>	left to right
<code>< <= > >=</code>	left to right
<code>== !=</code>	left to right
<code>&&</code>	left to right
<code> </code>	left to right
<code>= += -= *= /= %=</code>	right to left

`int m=3, k=2, n=4;`
What is the results?

m<n
k<m<n
k>m>n
m<n>k
m&& k<n

Summary

- ◆ **Variable is**
- ◆ **Basic memory operations are.....**
- ◆ **Expression is**
 - Which of the following operators will change value of a variable? + - * / % ++
 - Which of the following operators can accept only one operand? + - * / % --
 - $13 \& 7 = ?$
 - $62 | 53 = ?$
 - $17 ^ 21 = ?$
 - $12 >> 2 = ?$
 - $65 << 3 = ?$

Summary (cont.)

◆ Expressions

- Arithmetic operators
- Relational operators
- Logical operators
- Bit operators
- Shorthand Assignment Operators
- Casting
- Precedence