Slots 02-03
Basic Computations

Variables

Basic Memory Operations

Expressions

Objectives

A problem needs to be represented by data. After studying this chapter, 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
  - In-Class Problem
  - Statistics: Which operators should be used?
  - Relational
  - Logical
  - Bit operators
  - Shorthand Assignment Operators
  - Mixing Data Types
  - Casting
  - Precedence

Review

- ***Computer program***: A set of instructions that computer hardware will execute.
- ***Issues for a program/software***: Usability, Correctness, Maintainability, Portability
- ***Computer software***: A set of related programs
- ***Steps to develop a software***: Requirement collecting, Analysis, Design, Implementing, Testing, Deploying, Maintaining
- ***Data***: Specific values that describe something
- ***Information***: Mean of data
- ***Fundamental Data Units***: Bit, Nibble, Byte, KB, MB, GB, TB
- ***Data Representation***: Number systems: 2, 10, 8, 16
- ***Program Instructions***: <opcode, operand1, operand 2>
- ***Programming Languages***: Machine language, Assembly, High-level languages

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.

2.1- Variables and Data Types

A variable is a name referencing to a memory location (address)

→ Holds binary data

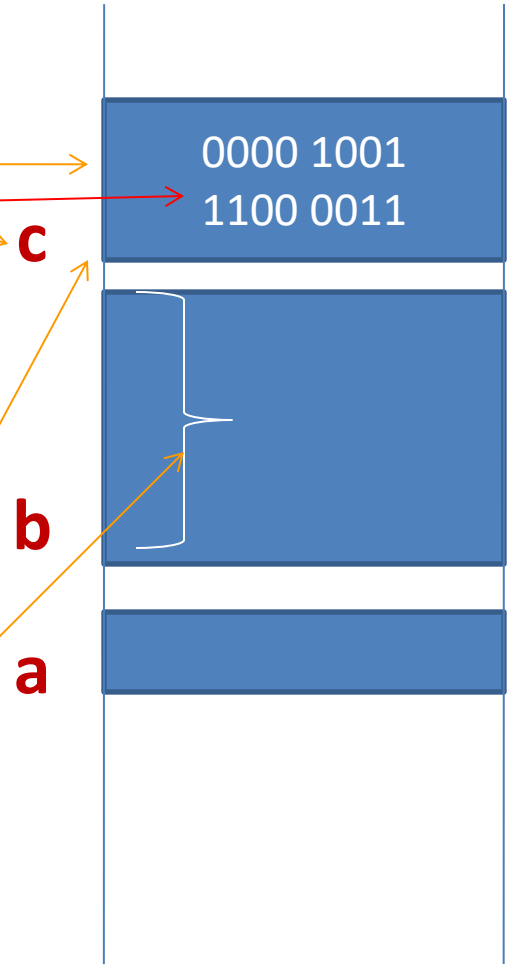
→ 2 basic operations: set value, get value.

→When the program is compiled, the compiler will determine the position where the variable is allocated.

Questions:

(1) Where is it? → It's Address

(2) How many bytes does it occupy?
→ Data type



2.1- Variables and Data Types…

Data Types:

- Typed languages, such as C, subdivide the universe of data values into sets of distinct type.
- 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.

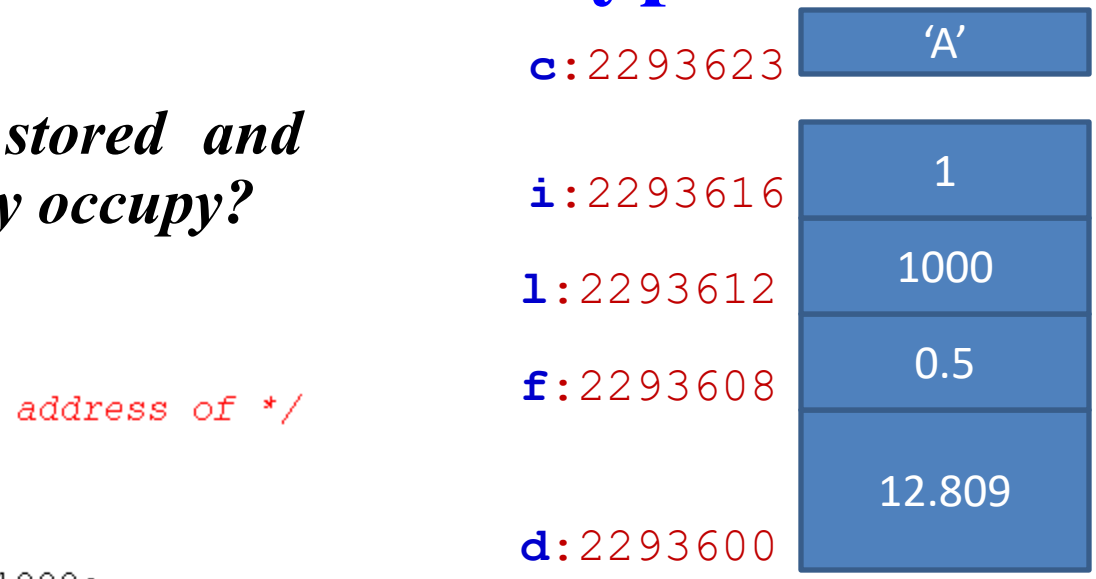
2.1- Variables and Data Types…

C has 4 primitive data types:

Type	Length	Range
int	Word (length of CPU register)	-32,768 to 32,767 (16 bit) -2,147,483,648 to 2,147,483,647 (32 bit)
char	byte	-128 to 127
float	4 bytes	$3,4 * 10^{-38}$ to $3,4 * 10^{38}$
double	8 bytes	$1,7 * 10^{-308}$ to $1,7 * 10^{308}$

2.1- Variables and Data Types…

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



Vars_demo.c

```
/* Variables Demo - Operator &: address of */
#include <stdio.h>
#include <conio.h>
int main() {
   char c='A'; int i=1; long l=1000;
   float f=0.5f; double d=12.809 ;
   printf("Variable c: at addr: %u, value: %c, size: %d\n", &c, c, sizeof(c));
   printf("Variable i: at addr: %u, value: %d, size: %d\n", &i, i, sizeof(i));
   printf("Variable l: at addr: %u, value: %ld, size: %d\n", &l, l, sizeof(l));
   printf("Variable f: at addr: %u, value: %f, size: %d\n", &f, f, sizeof(f));
   printf("Variable d: at addr: %u, value: %lf, size: %d\n", &d, d, sizeof(d));
   getch();
}
```

```
G:\GiangDay\FU\PFC\PFC_Lab\Vars_demo.exe
Variable c: at addr: 2293623, value: A, size: 1
Variable i: at addr: 2293616, value: 1, size: 4
Variable l: at addr: 2293612, value: 1000, size: 4
Variable f: at addr: 2293608, value: 0.500000, size: 4
Variable d: at addr: 2293600, value: 12.809000, size: 8
```

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

2.1- Variables and Data Types…

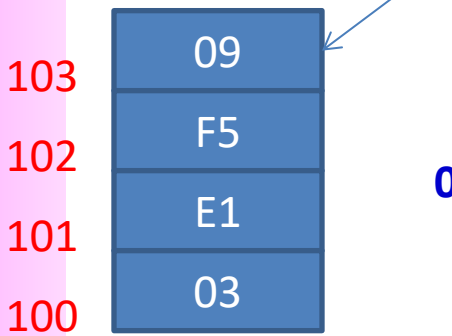
Qualifiers:

- We can qualify the int data type so that it contains a minimum number of bits.
- Qualifiers:
  - **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 less bits than a double.*

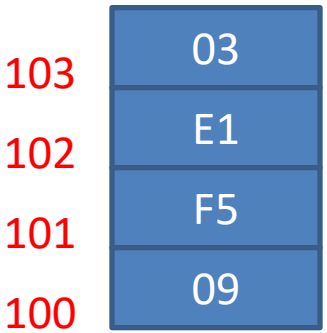
2.1- Variables and Data Types…

Representation of Integral Values:

- C stores integral values in equivalent binary form.
- Non-Negative Values:
  - **Intel** uses this little-endian ordering.
  - **Motorola** uses big-endian ordering.



Value is stored: 09F5E103



Least significant is stored in lowest byte

Most significant is stored in lowest byte

Each CPU family has it's own way to store data . So, compilers must have a suitable way for copying data from this variable to other. And due to this reason also, each compiler can run well only on a specific family of CPU only. A supplier may supply some versions of their compiler for some CPU families.

2.1- Variables and Data Types…

Exercises:

- Convert the following decimal integers to binary:
  63 _____
  219 _____
- Convert the following binary notation to decimal:
  - 0111 0101 _____
  - 0011 1011 _____
  - 01011011 _____

2.1- Variables and Data Types…

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
- For example:

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

2.1- Variables and Data Types…

***Exercises** (Use signed 1-byte integral number):*

- What is the two's complement notation of

-63_____

-219_____

- Convert the following binary notation to decimal:

1111 0101_____

1011 1011_____

2.1- Variables and Data Types…

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
unsigned short	>=16 bits	0	65,535	
unsigned int	1 word	0	4,294,967,295	65,535
unsigned long	>=32 bits	0	4,294,967,295	
unsigned long long	>=64 bits	0	18,446,744,073,709,551,615	

2.1- Variables and Data Types…

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

2.1- Variables and Data Types…

The ASCII table for characters

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂*

2.1- Variables and Data Types…

Exercises:

- What is the ASCII encoding for
  '0' _____
  'a' _____
  'A' _____
- What is the EBCDIC encoding for
  '0' _____
  'a' _____
  'A' _____
- Convert the following binary notation to an ASCII character:
  0110 1101 _____
  0100 1101 _____
- Convert the following decimal notation to an EBCDIC character:
  199 _____
  35 _____

2.1- Variables and Data Types…

Representation of Floating-Point Data :

- Computers store floating-point data using two separate components:
  - an exponent and
  - a mantissa (phần định trị)

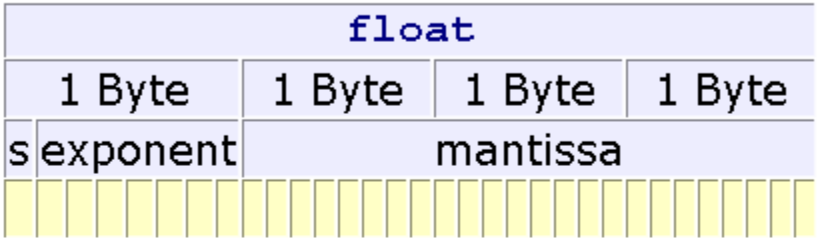
$1.2345 = 1.2345 * 10^{0}$
$123.45 = 1.2345 * 10^{2}$
$1234.5 = 1.2345 * 10^{3}$
$0.0012345 = 1.2345 * 10^{-3}$

Give your comment about position of the point mark and it's mantissa

2.1- Variables and Data Types…

IEEE 754, 32 bits float:

Under IEEE 754, the model for a `float` occupies 32 bits, has one sign bit, a 23-bit mantissa and a 8-bit exponent:



We calculate the value using the formula

$$x = sign * 2^{exponent} * \{ 1 + f_1 2^{-1} + f_2 2^{-2} + ... + f_{23} 2^{-23} \}$$

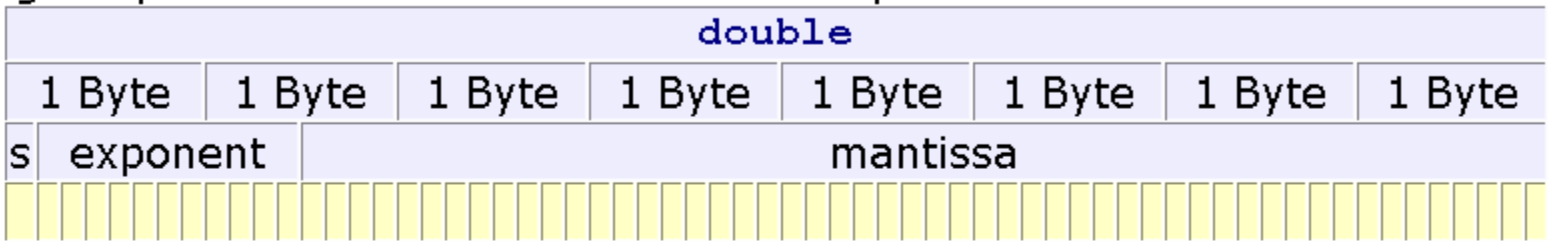
where f_i is the value of bit i and

$$-126 <= exponent <= 127$$

2.1- Variables and Data Types…

IEEE 754, 64 bits double :

Under the IEEE standard, the model for a `double` occupies 64 bits, has one sign bit, a 52-bit mantissa and a 11-bit exponent:



We calculate the value using the formula

$$x = sign * 2^{exponent} * \{ 1 + f_1 2^{-1} + f_2 2^{-2} + ... + f_{52} 2^{-52} \}$$

where f_i is the value of bit i and

$$-1022 <= exponent <= 1023$$

2.1- Variables and Data Types…

Limits on float and double data type in the IEEE standards:

The limits on `float` and `double` under the IEEE standard are:

Type	Size	Significant	Min Exponent	Max Exponent
`float`	4 bytes	6	-37	38
`double`	8 bytes	15	-307	308

2.1- Variables and Data Types…

Variable Declarations in C:

data_type identifier [= initial value];

- For example:

  char section;

  int numberOfClasses;

  double cashFare = 2.25;

Naming Conventions: Name is one word only

- must not be a C reserved word
- Some compilers allow more than 31 characters, while others do not. To be safe, we avoid using more than 31 characters.

2.1- Variables and Data Types…

Exercises:

Which of the following is an invalid identifier?

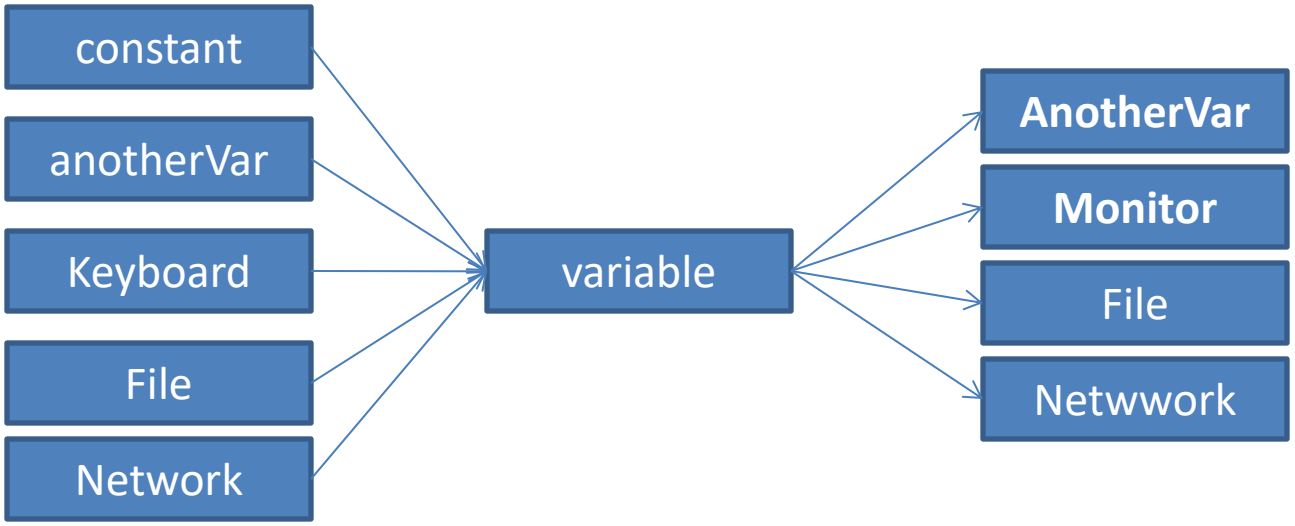
whale giraffe's *camel_back* 4me2
_how_do_you_do senecac.on.ca *digt3* register

- **Select a descriptive identifier for and write a complete declaration for:**
  - A shelf of books_____
  - A cash register_____
  - A part_time student_____
  - A group of programs_____

2.1- Variables and Data Types…

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.



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.

2.2- 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)

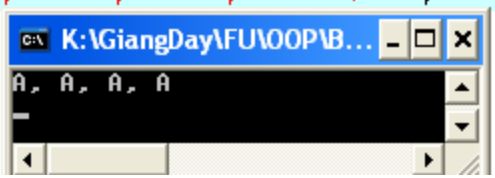
2.2- Literals: Characters, Strings

4 ways for representing a character literal:

- 04 ways:
  - Enclose the character single quotes - for example '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 =



2.2- 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

2.2- Literals: Escape Sequences…

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

Error! Why?

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

Modify then run it

Change \ to \\ then run it

2.2- 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

2.3- Named Constants

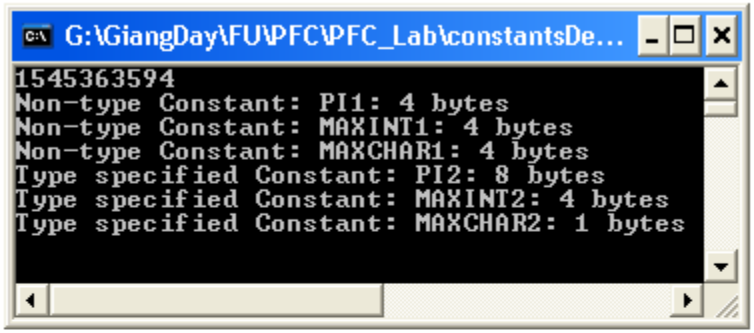
- Use the pre-processor (pre-compiled directive) **#define** or the keyword **const**

Compiler will allocate memory location for constants that are declared using the keyword **const**

constantsDemo.c

```
/* Constants demonstration */
#include <stdio.h>
#include <conio.h>
#define PI 3.141592
const PI1 = 3.141593;
const double PI2 = 3.141593;
const MAXINT1 = 12;
const int MAXINT2 = 10;
const MAXCHAR1 = 'Z';
const char MAXCHAR2 = 'A';

int main ()
{  printf("%d\n", PI*3*3);
   printf("Non-type Constant: PI1: %d bytes\n", sizeof(PI1));
   printf("Non-type Constant: MAXINT1: %d bytes\n", sizeof(MAXINT1));
   printf("Non-type Constant: MAXCHAR1: %d bytes\n", sizeof(MAXCHAR1));
   printf("Type specified Constant: PI2: %d bytes\n", sizeof(PI2));
   printf("Type specified Constant: MAXINT2: %d bytes\n", sizeof(MAXINT2));
   printf("Type specified Constant: MAXCHAR2: %d bytes\n", sizeof(MAXCHAR2));
   getch();
}
```



2.3- Named Constants

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.

```
2 #include <stdio.h>
3 #define PI =3.14;
4 int main()
5 {    printf("%lf\n",PI*3*3);
6      getchar();
7      return 0;
8 }
```

```
#define PI =3.14;
```

(... , PI*3*3)

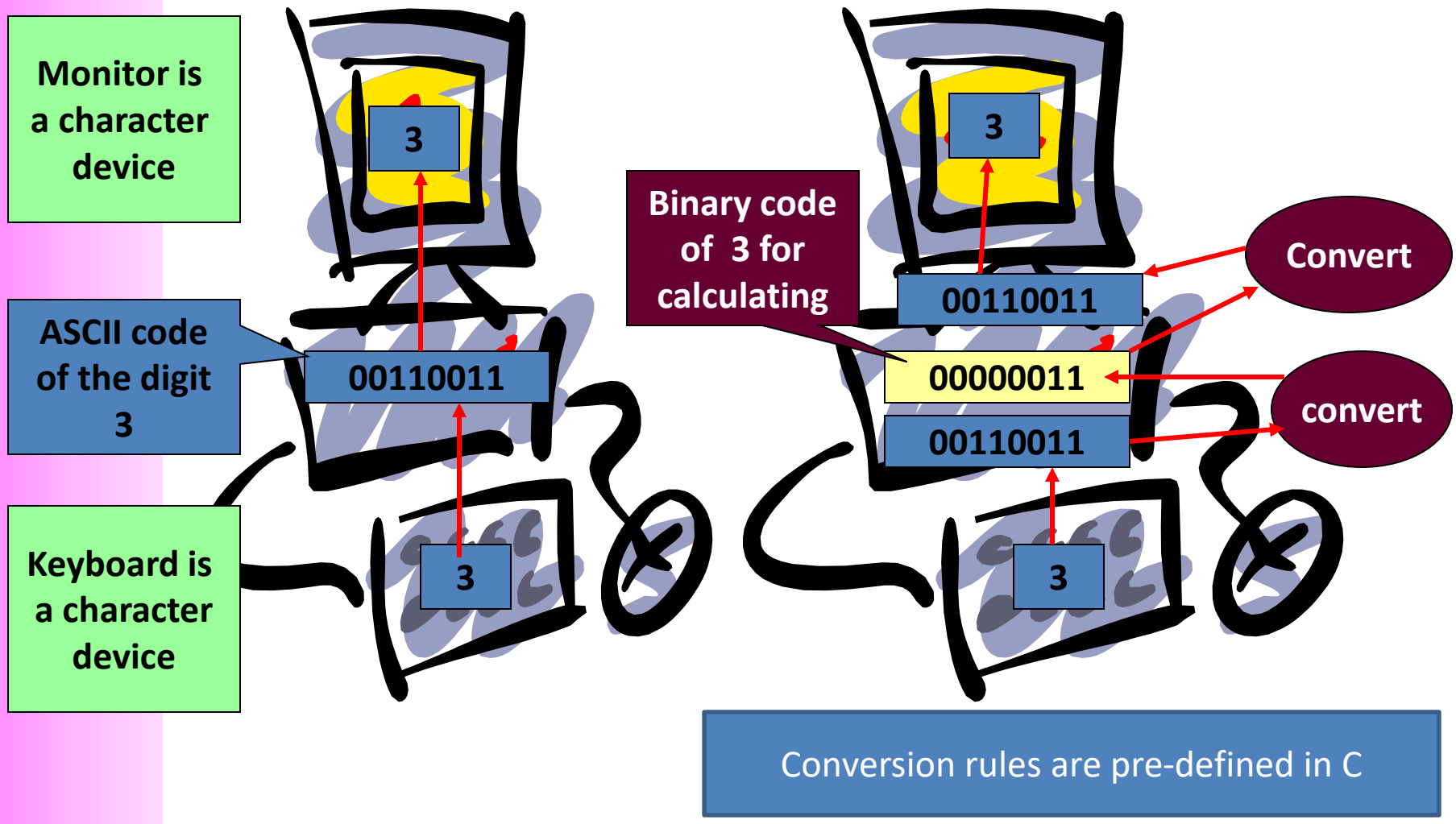
(... , =3.14;*3*3)

Line	File	Message
	K:\GiangDay\FU\OOP\BaiTap\testE...	In function `main':
5	K:\GiangDay\FU\OOP\BaiTap\testE...	syntax error before '=' token
5	K:\GiangDay\FU\OOP\BaiTap\testE...	invalid type argument of `unary *'
5	K:\GiangDay\FU\OOP\BaiTap\testE...	syntax error before ')' token

Fill the blank

- 3 ways to specify a constant in a source program: use a literal, use the keyword…., and specify a macro using the directive …..

2.4- Input/Output Variables



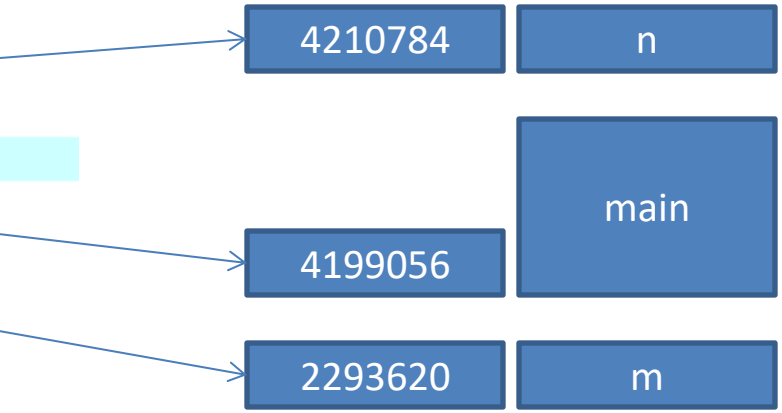
2.4- Input/Output Variables…

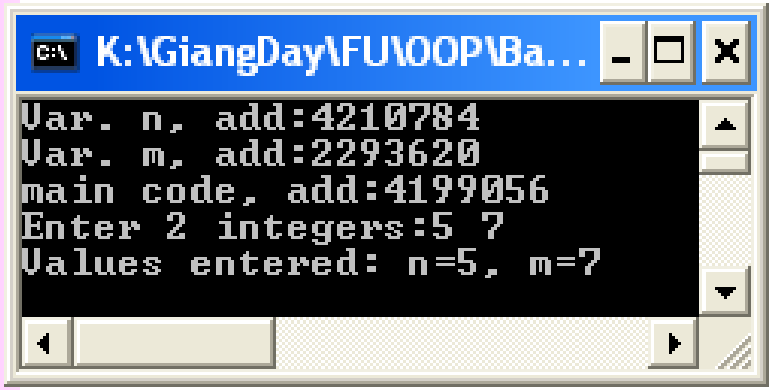
Conversion Specifiers

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

2.4- Input/Output Variables…

```
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 }
```





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.

2.4- Input/Output Variables…

Input a value to a variable:

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

Output the value of a variable:

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

```
#include <stdio.h>
int main()
{    int n;
     double x, y;
     printf("Enter an integer:");
     scanf("%d",&n);
     printf("Enter 2 real numbers:");
     scanf("%lf%lf",&x, &y);
     double num1, num2;
     char op;
     printf("Enter an expression + - * / :");
     scanf("%lf%c%lf",&num1,&op, &y);
     printf("Expression inputted is: %lf%c%lf\n",num1, op, num2);
     getchar();
     getchar();
     return 0;
}
```

```
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.

Format string

Data holders

Questions

- **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.**

Exercises

1- 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.

2- 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();
getchar();

2.5- Expressions

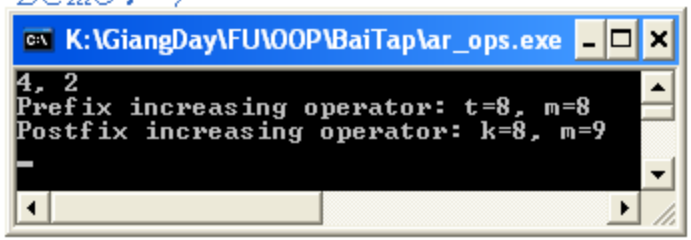
- Expression is a valid association of constants, variables, operators and functions and returns an only result.
- Examples:
  - 32-x+y/6
  - 45 > 5*x
  - 16.5 + 4/sqrt(15) * 17 – 8
  - y = 17 + 6*5/9 –z*z
- Hardware for calculating expressions: ALU
- Operations that can be supported by ALU: Arithmetic,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

2.5- Expressions: Arithmetic Operators in C

Op.	Syntax	Description	Example
+	+x	leaves the variable, constant or expression unchanged	y = +x ; ←→ y = x;
-	-x	reverses the sign of the variable	y= -x;
+ -	x+y x-y	Add/substract 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; → 3 z = 10.0/3; → 3.3333333
%	x%y	Get remainder of a integral division	17%3 → 2 15.0 % 3 → ERROR
++ --	++x --x x++ x--	Increase/decrease the value of a variable (prefix/postfix operators)	Demo in the next slide.

Expressions: Math. Operators…

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




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

Explain yourself the output

Expressions: Math. Operators…

```
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.

Expressions: 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 }
```

Expressions: Logical Operators

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

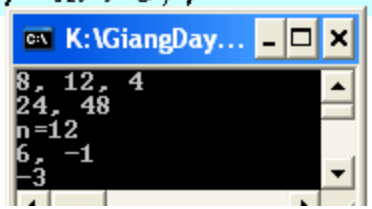
```
2 #include <stdio.h>
3 int main()
4 {   int n=30, m= 5;
5     float x= 3.5F, y=8.12F;
6     printf("%d, %d\n", (n<m || x>=y), !(x>y));
7     getchar();
8     return 0;
9 }
```

Expressions: 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
0000 0000 0000 1100
0000 0000 0000 1000
0000 0000 0000 1000 → 8

n|m
0000 0000 0000 1100
0000 0000 0000 1000
0000 0000 0000 1100 → 12

n^m
0000 0000 0000 1100
0000 0000 0000 1000
0000 0000 0000 0100 → 4

Expressions: Bitwise Operators

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})

Expressions: Assignments Operators

- Variable = expression
- Shorthand assignments:

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

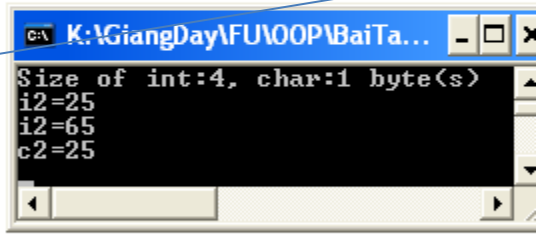
Expressions: Mixing Data Types

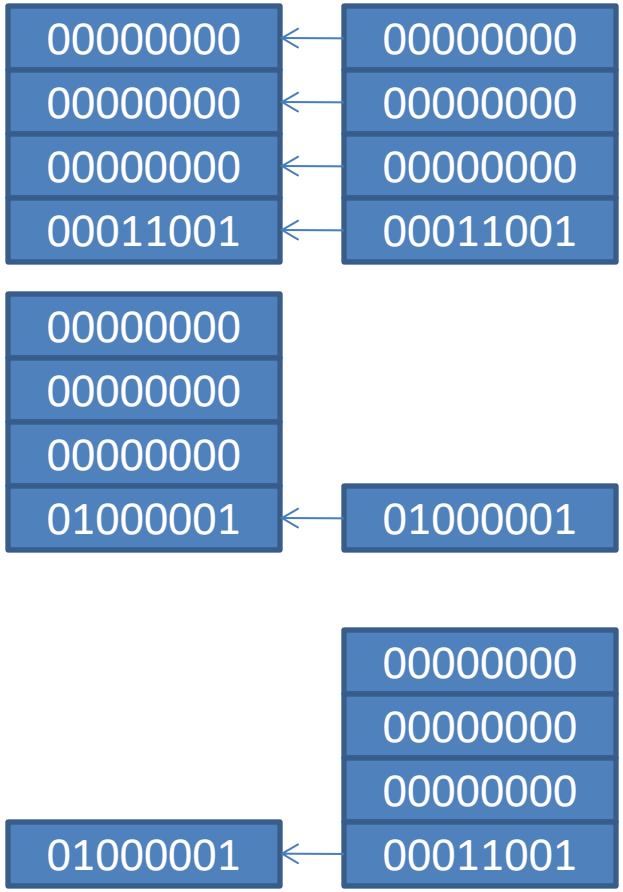
- Although the ALU does not perform operations on operands of differing data type directly, C compilers can interpret expressions that contain operands of differing data type.
- If a binary expression contains operands of differing type, a C compiler changes the data type of one of the operands to match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

Expressions: Mixing Data Types

- 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;
}
```

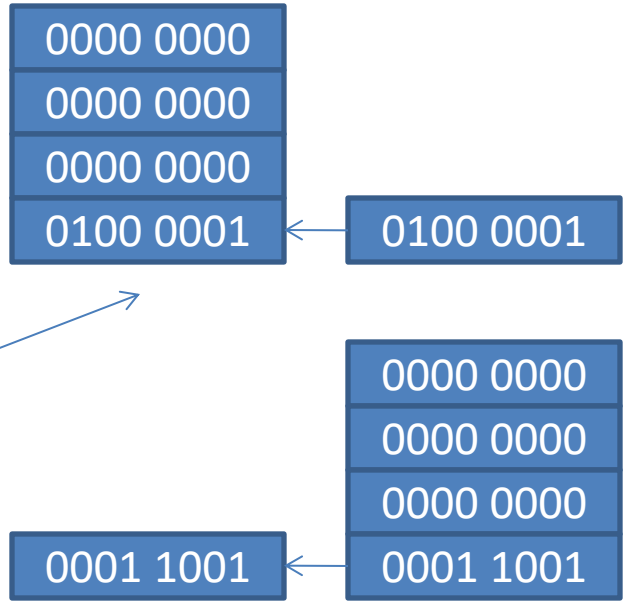




Direction for copying: From the lowest byte to higher bytes

Expressions: Mixing Data Types

- **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.



Expressions: Mixing Data Types

- **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.

	right operand				
left 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

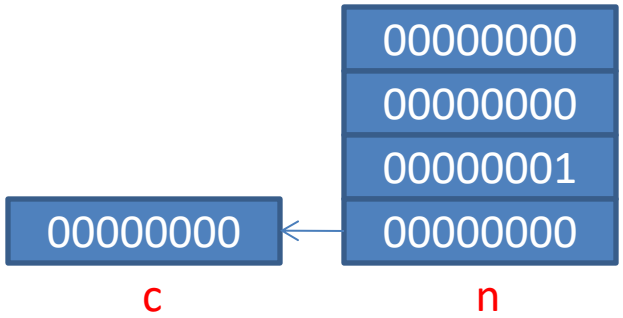
Expressions: Mixing Data Types

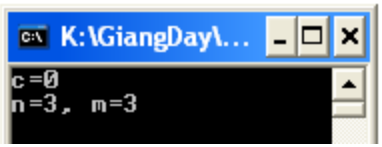
- **Explicit Casting**

  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*	double version of
(float)	*variable or constant*	float version of
(int)	*variable or constant*	int version of
(char)	*variable or constant*	char version of
(long)	*variable or constant*	long version of

```
#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;
}
```





Expressions: 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
++ -- (post)	left to right
++ -- (pre) + - & ! (all unary)	right to left
(data type)	right to left
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
= += -= *= /= %=	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

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

Extra Exercise

```
#include <stdio.h>
int n;
double x;
char c1;
int main()
{   int m;
    short s;
    long L;
    float y;
    printf("Code of main:%u\n", &main));
    printf("Variable n, add:%u, memory size:%d\n", &n, sizeof(n));
    /* Your code to view address and memory size of other variables*/
    /* Complete the program, compile and run it */
    /* Draw the memory of the program*/
    getchar();
    return 0;
}
```

Thank You