Lecture 3-6
Getting Started with C

Muhammad Ali Nayeem

http://teacher.buet.ac.bd/ali_nayeem/CSE109_Feb2015/

Bits and Bytes

- Bit
  - Bit - stores just a 0 or 1
  - Can store/communicate 2 states
  - Transistors on a chip can make a bit
  - Too small to be much use on its own .. form into a byte
- Byte
  - Byte - the **most important unit** of storage
  - One byte is made of 8 bits
  - We can access each byte of RAM

Byte

- How much can one byte hold?
  - 1 bit -- 0/1 -- 2 patterns
  - 2 bits -- 00/01/10/11 -- 4 patterns
  - 3 bits -- 000/001/010/011/100/101/110/111 -- 8 patterns
  - 3 bit pattern has twice as many patterns vs. the 2 bit pattern
  - So n bits has twice as many patterns as (n-1) bits
  - What is the general formula for # patterns for n-bits?
  - 2^n (2 to the nth power)

Kilobyte, Megabyte, …

- Kilobyte
  - Kilobyte KB – 1024 (2^{10}) bytes
  - About a thousand bytes
- Megabyte
  - Megabyte (MB) - 1024 kilobytes
  - About a million bytes
- Gigabyte (GB), Terabyte (TB), …

Number system

Most numbering system use positional notation :

$N = a_n r^n + a_{n-1} r^{n-1} + \ldots + a_1 r^1 + a_0 r^0$

Where:

N: an integer with n+1 digits

r: base

$a_i \in \{0, 1, 2, \ldots, r-1\}$

Example:

a) N = 278

r = 10 (base 10) => decimal numbers

symbol: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (10 different symbols)

N = 278 => $n = 2$; $a_2 = 2$; $a_1 = 7$; $a_0 = 8$

$278 = (2 \text{ x } 10^2) + (7 \text{ x } 10^1) + (8 \text{ x } 10^0)$

Decimal, Binary, Hexadecimal

- Base: 10,2,16
- Decimal <--> Binary
- Binary <--> Hexadecimal

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Example:

1. Convert the following binary numbers into hexadecimal numbers:

(a) 00101111_2

Refer to the binary-hexadecimal conversion table above

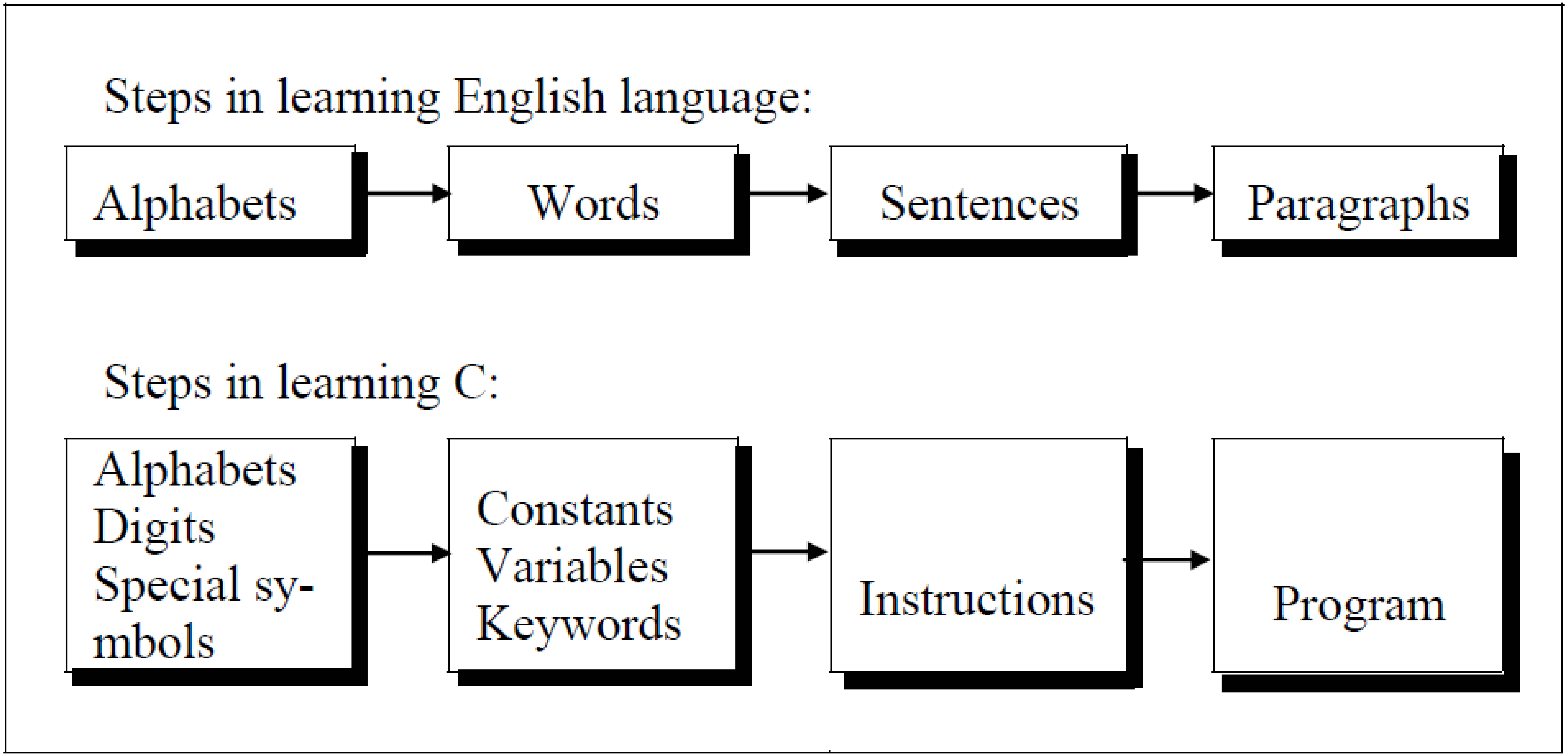
$$\underbrace{0010}_{2} \mid \underbrace{1111_2}_{F} = 2F_{16}$$

A Mental Picture of Memory/RAM

- You can access each byte of RAM from the program
- Every byte has
  - Location/address: where is the byte in the RAM?
  - Content/value: what the byte contains?

Address	Value
0	0101 1111
1	1101 1111
2	0101 0000
3	0000 1100
4	1101 0101
5	1010 0101
…	…

Learning a Language



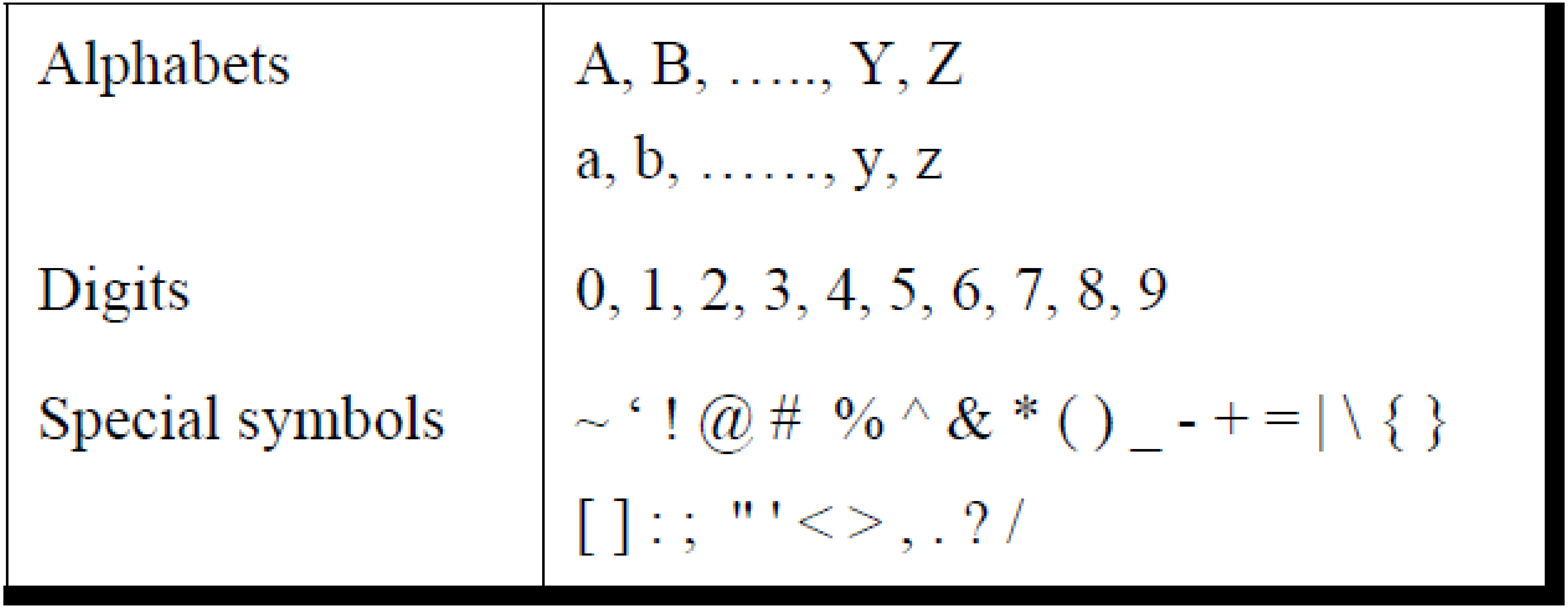
Steps in learning English language:

Alphabets → Words → Sentences → Paragraphs

Steps in learning C:

Alphabets
Digits
Special symbols
→
Constants
Variables
Keywords
→ Instructions → Program

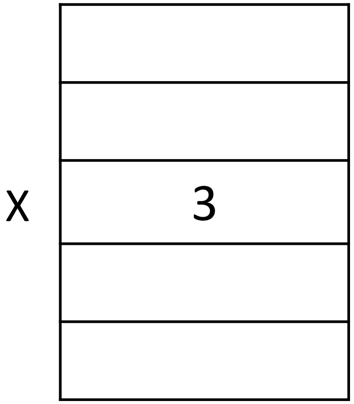
Alphabet/Character Set for C



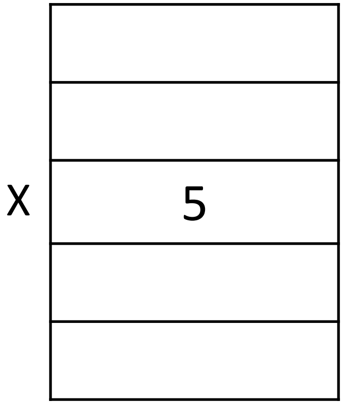
Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Constant Vs. Variable

- Variable
  - a named memory location
  - Can hold different values at different times
  - Can hold only one value at a time
- Constant
  - Just a value
  - Doesn’t change
  - Doesn’t have any memory location

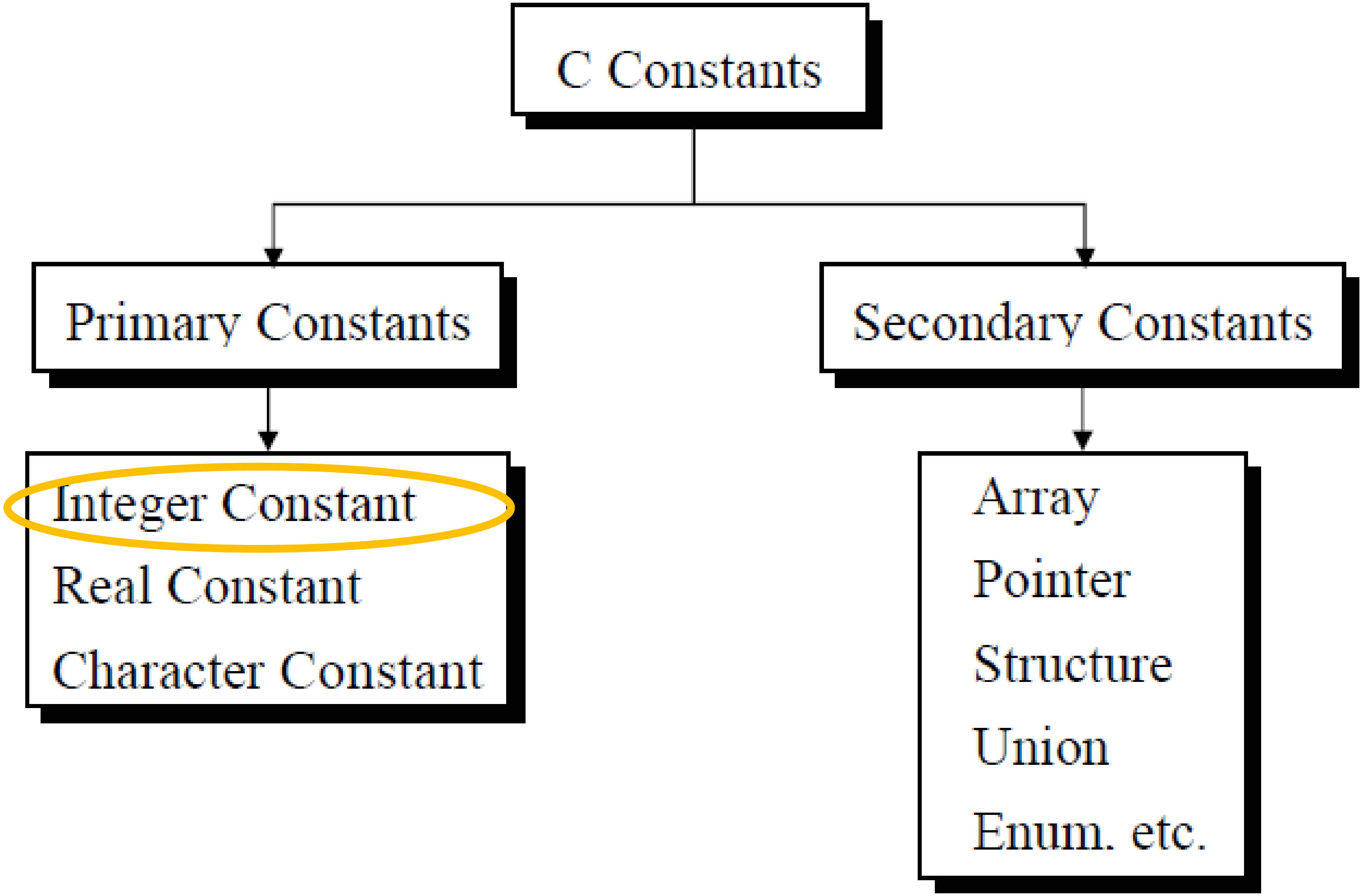


X = 3



X = 5

C Constants



C Constants

Primary Constants

Secondary Constants

Integer Constant

Real Constant

Character Constant

Array

Pointer

Structure

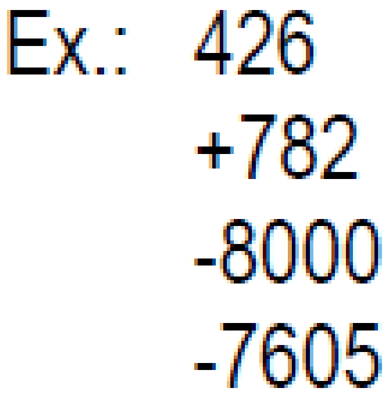
Union

Enum. etc.

Let's focus on Integer first

Rules for Constructing Integer Constants in C

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can be either positive or negative.
- If no sign precedes an integer constant it is assumed to be positive.
- No commas or blanks are allowed within an integer constant.
- Has a fixed size
  - Usually 32 bits (4 bytes)



Ex.: 426
+782
-8000
-7605

Keywords

- C has some words that has a **special meaning** for the **compiler**

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

Types of C Variables

- depends on the type of constant (integer/real/character…) it can hold
- A particular type of variable can hold only the same type of constant.
- For example, an integer variable can hold only an integer constant

`int` refers to integer type in C

Rules for Constructing Variables Names

- Case sensitive
  - Count, count & COUNT are different
- Can be of any length, but only first **31** characters are important
- Can contain letters, digits and the '_'
- But first character must be a letter or '_'
- Variable name cannot be same as a keyword
- For example –

  correct: abcd, abcd2, abcd_3, Abcd

  incorrect: ab cd, 2abcd, abcd…3, ab!cd

Variable Name

- Should be clear and meaningful
- If two or more words are needed then
  - either separate them using a ‘_’
  - or keep them together, but start each word except the first one with a capital
- For Example

student_no   average_age

dateOfBirth   averageAge

Arithmetic Operators

- Acts on variables and constants
  - +
  - -
  - * (Multiplication)
  - / (Division)
  - % (Modulus)
    - Applicable only for integers

Statements / C instruction

- Combination of variables, constant, keywords, operators etc.
- Specifies an action to be performed by the program
- **A C program is a series of statements**
- The **statements** in a program must appear in the same order in which we wish them to be executed
- Every C **statement** must end with a semicolon(**;)**
- Spaces may be inserted between two **words** to improve the readability of the statement
  - However, no blank spaces are allowed **within** a variable, constant or keyword.

Variable declaration

- Each variable must be declared before use
- This declaration is done at the beginning of the program
- A variable is declared using a statement of the form:

```
type name;
```

  - `type`: type of the variable
  - `name`: name of the variable

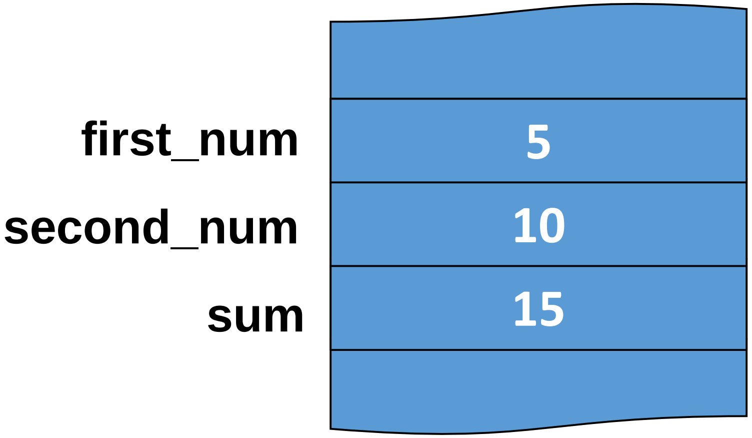
  - Example: `int first_num;`
  - `In C int is the type for integer variable`
- Multiple variables of the same type can be declared together separated by comma (,)
  - Example:

```
int first_num, second_num, sum;
```

Adding 2 integers

```
main()
{
        int first_num, second_num, sum;
        first_num = 5;
        second_num = 10;
        sum = first_num + second_num ;
}
```

Can you see the constants & variables??



A Portion of RAM

More on Variable Declaration

- While declaring the type of variable we can also **initialize** it, see below

```
int first_num=5, second_num=10, sum;
```

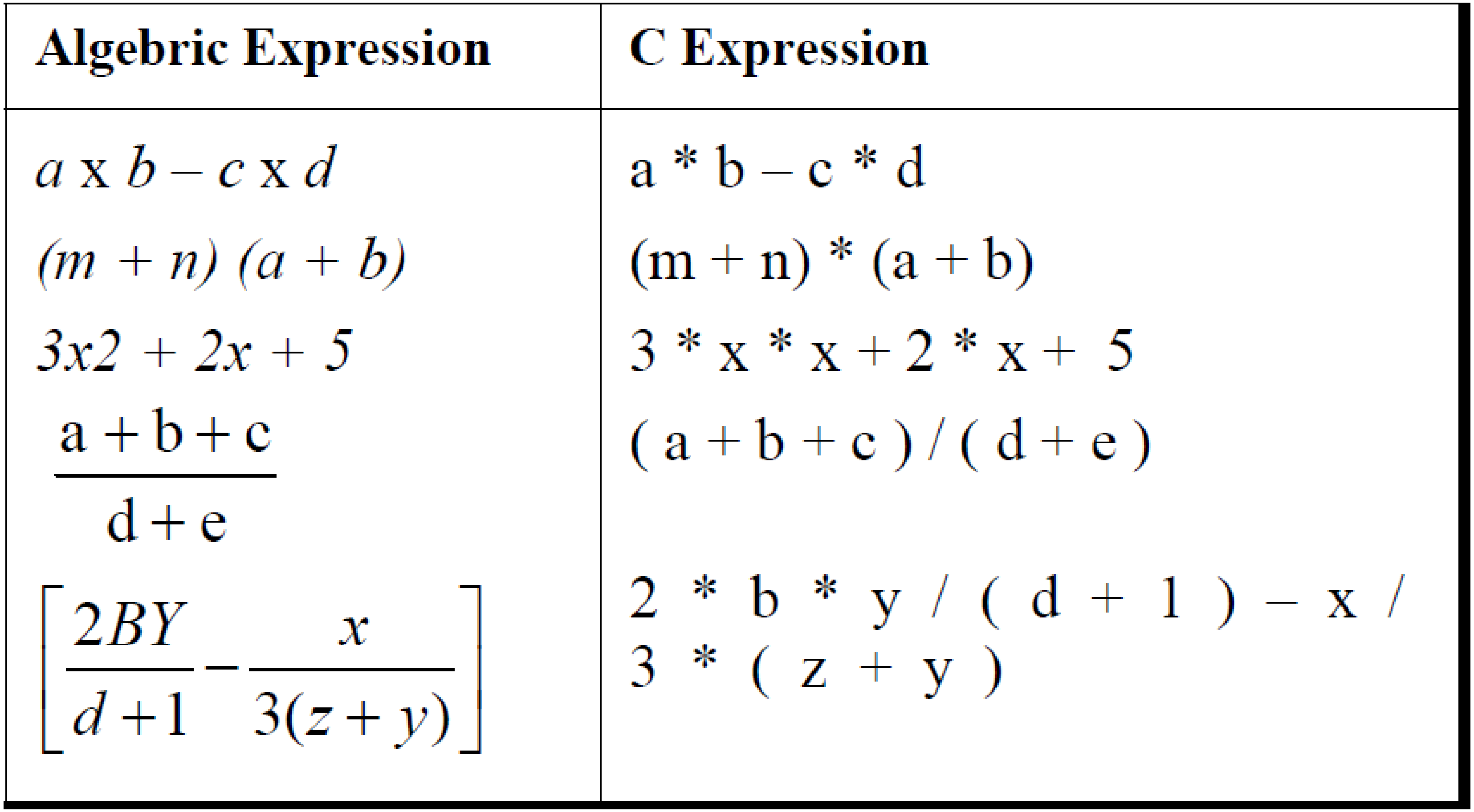
- Without initializing or before assigning values, a variable contains unknown value
  - Known as Garbage value

Arithmetic Expressions

Any Combination of arithmetic operators and operands like

- 2 * var1 + 6
- 28
- (var1 - var2) * 2
- var2

Arithmetic Expressions



Algebric Expression	C Expression
$a \text{ x } b - c \text{ x } d$	a * b – c * d
$(m + n)\ (a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\dfrac{a+b+c}{d+e}$	(a + b + c) / (d + e)
$\left[\dfrac{2BY}{d+1} - \dfrac{x}{3(z+y)}\right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

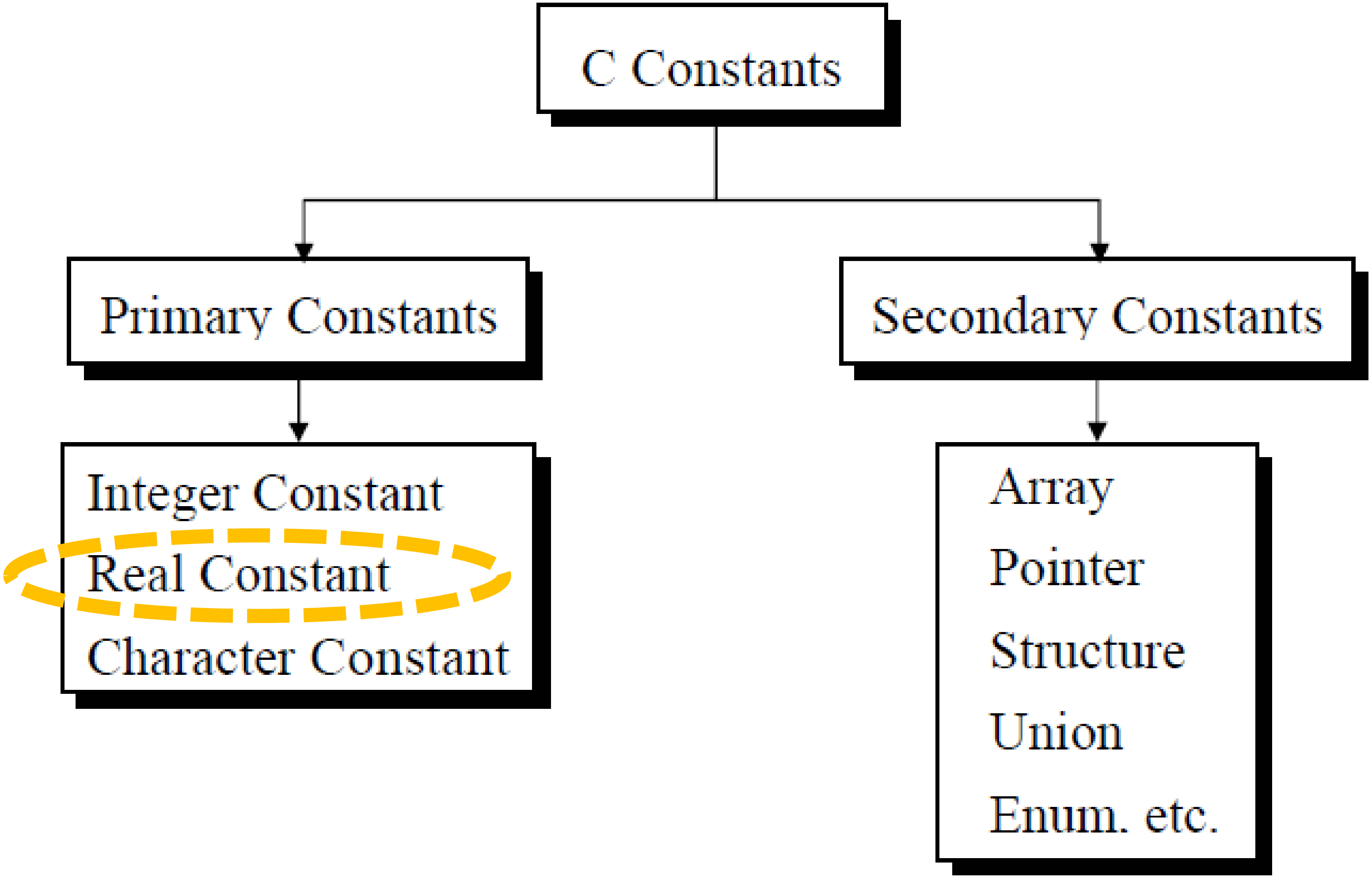
Assignment statement

- Assigns the result of an Arithmetic Expression to a variable

  sum = first_num + second_num ;

  - *= assignment operator*
    - Expression at the right
    - Variable at the left
  - At first the expression at the right is evaluated
  - Then the result is stored at the memory location represented by the variable at the left
  - C allows only one variable on left-hand side of =.
  - **Remember Assignment statement is not an equation**

C Constants



C Constants

Primary Constants

Secondary Constants

Integer Constant
Real Constant
Character Constant

Array
Pointer
Structure
Union
Enum. etc.

Now focus on Real

Real Constants

- Often called Floating Point constants
- Size 32 bits
- Written in two forms
  - Fractional form
    - +325.34
    - 426.0
    - -32.76
    - -48.5792
  - Exponential form
    - usually used if the value is too small or too large
    - +3.2e-5
    - 4.1e8
    - -0.2e+3
    - -3.2e-5
    - part appearing before ‘e’ is called mantissa
    - the part following ‘e’ is called exponent

Rules for Fractional form

- A real constant must have at least one digit
- It must have a decimal point
- It could be either positive or negative
- Default sign is positive
- No commas or blanks are allowed within a real constant

Rules for Exponential form

- The mantissa part and the exponential part should be separated by a letter e or E
- The mantissa part may have a positive or negative sign
- Default sign of mantissa part is positive
- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive
- Range of real constants expressed in exponential form is -3.4e38 to 3.4e38

Real Variable

- Store real constants in memory
- C keyword for real type **`float`**

```
float radius = 5.66;
```

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

Arithmetic Operators for Real values

  - +
  - -
  - *
  - /
- What is missing here?

Displaying output

```
#include<stdio.h> /*Header file*/
main() /* The main function */
{
        int first_num, second_num, sum; /*Variable Declaration*/
        first_num = 5;
        second_num = 10;
        sum = first_num + second_num ;
        printf(“%d”, sum ); /*Can you remember this?*/
}
```

printf() – Readymade library function

- printf ("<format string>", <list of variables>) ;
- <format string>
  - Fixed portion
  - Variable portion
    - Format specifier (start with %)
    - %f for real values
    - %d for integer values

```
int a=4,b=5,c=8;
float x= 3.5;
printf("%f %d %d %d",x,3+2,c,a+b*c-d);
```

Running a program for different values

- Make the relevant change in the program
- Again compile and execute it
- What are the problems?
- Need to make the program general

Receive input from user

- Again readymade library function
- scanf()
- Counterpart of printf()

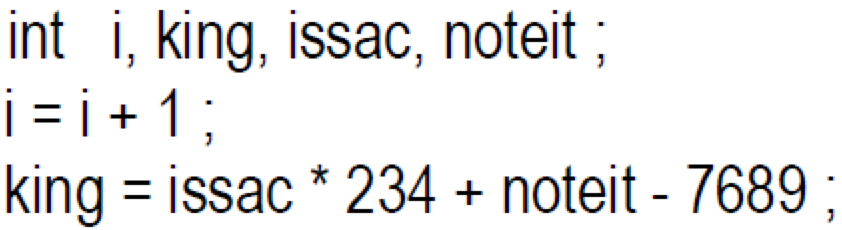
Receive inputs from user

```
#include<stdio.h>
main()
{
        int first_num, second_num, sum;
        printf(“Enter two numbers:”);
        scanf(“%d%d”,&first_num,&second_num); /*Wait for input*/
        sum = first_num + second_num ;
        printf(“The sum is:%d”, sum );
}
```

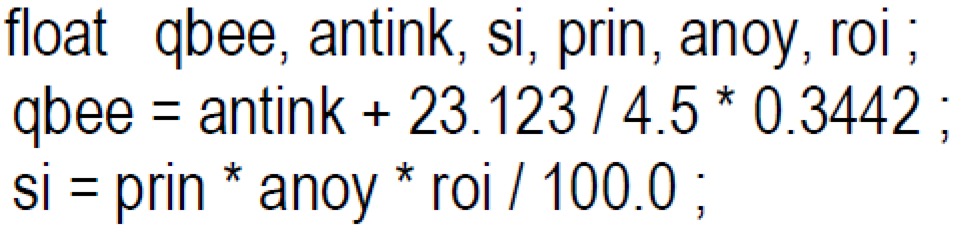
Cautions while using scanf()

- Don’t miss the & before variable name
- Don’t put any characters between “” other than format specifiers

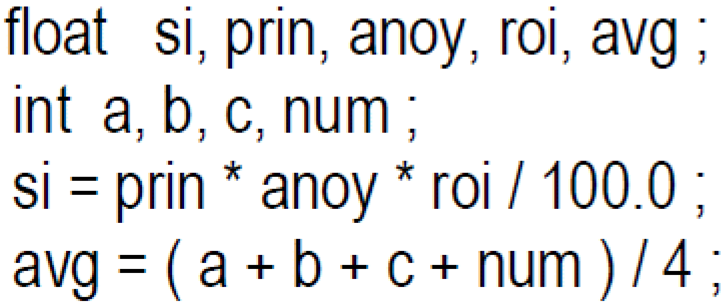
Arithmetic expressions



```
int   i, king, issac, noteit ;
i = i + 1 ;
king = issac * 234 + noteit - 7689 ;
```



```
float   qbee, antink, si, prin, anoy, roi ;
 qbee = antink + 23.123 / 4.5 * 0.3442 ;
 si = prin * anoy * roi / 100.0 ;
```

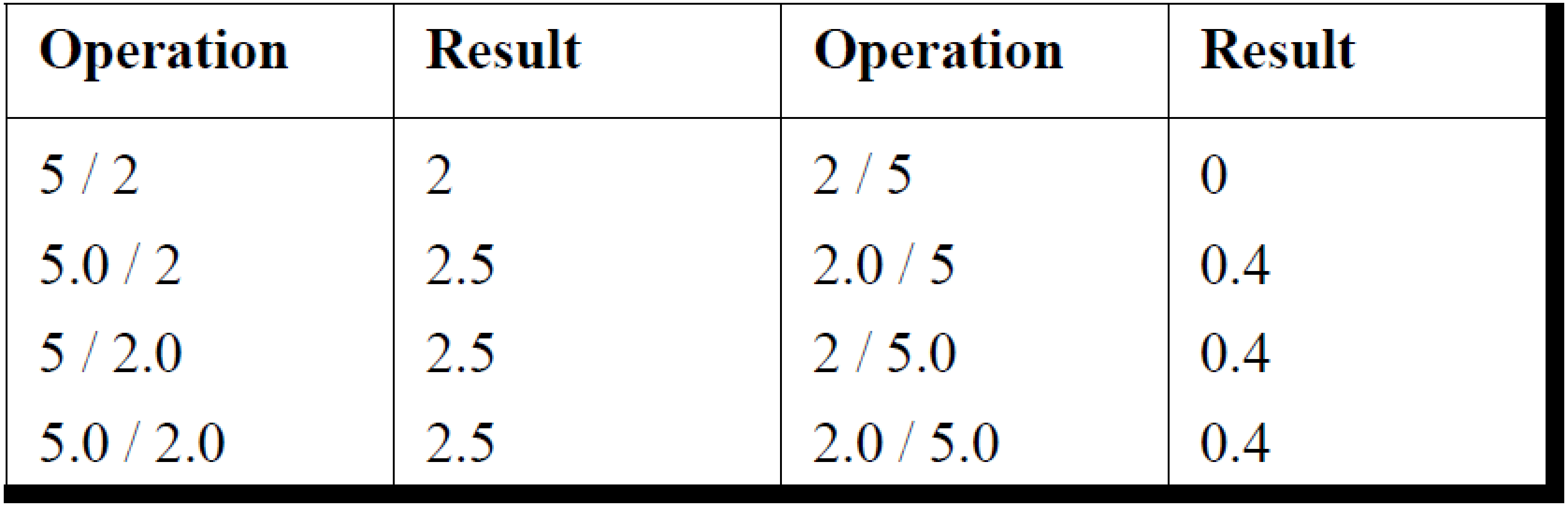


```
float   si, prin, anoy, roi, avg ;
 int  a, b, c, num ;
 si = prin * anoy * roi / 100.0 ;
 avg = ( a + b + c + num ) / 4 ;
```

Integer and Float Conversions

- An arithmetic operation between an integer and integer always yields an integer result
- An operation between a real and real always yields a real result
- An operation between an integer and real always yields a real result.
  - In this case the integer is first converted to a real
  - then the operation is performed.
  - Hence the result is real.

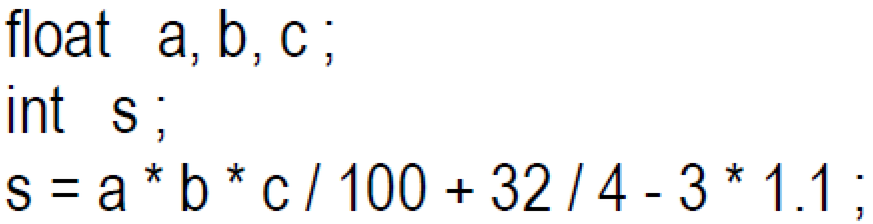
Integer and Float Conversions



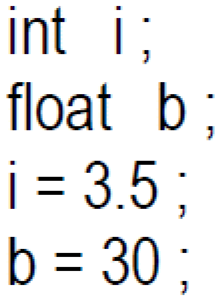
Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments

- What value is stored in the variable?



```
float   a, b, c ;
int   s ;
s = a * b * c / 100 + 32 / 4 - 3 * 1.1 ;
```

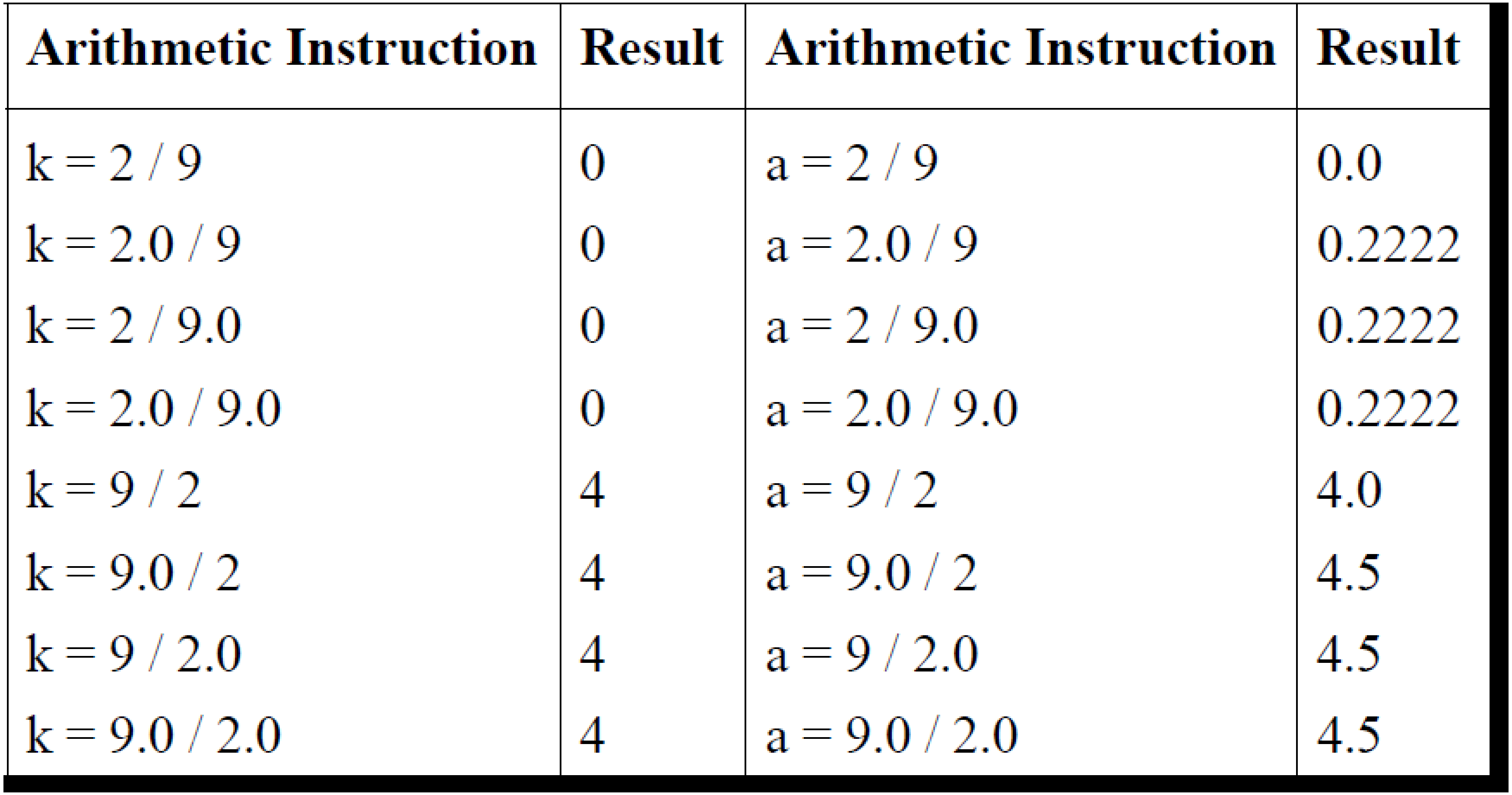


```
int   i ;
float   b ;
i = 3.5 ;
b = 30 ;
```

- during evaluation of the expression
  - the ints would be converted to floats
  - the result of the expression would be a float
- But when this float value is assigned to s
  - it is again demoted to an int and then stored in s

Type Conversion in Assignments

- Let’s assume that **k** is an integer variable and **a** is a real variable



Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

Type cast

- Cause Temporary type change

```
(type) value
```

- When needed?

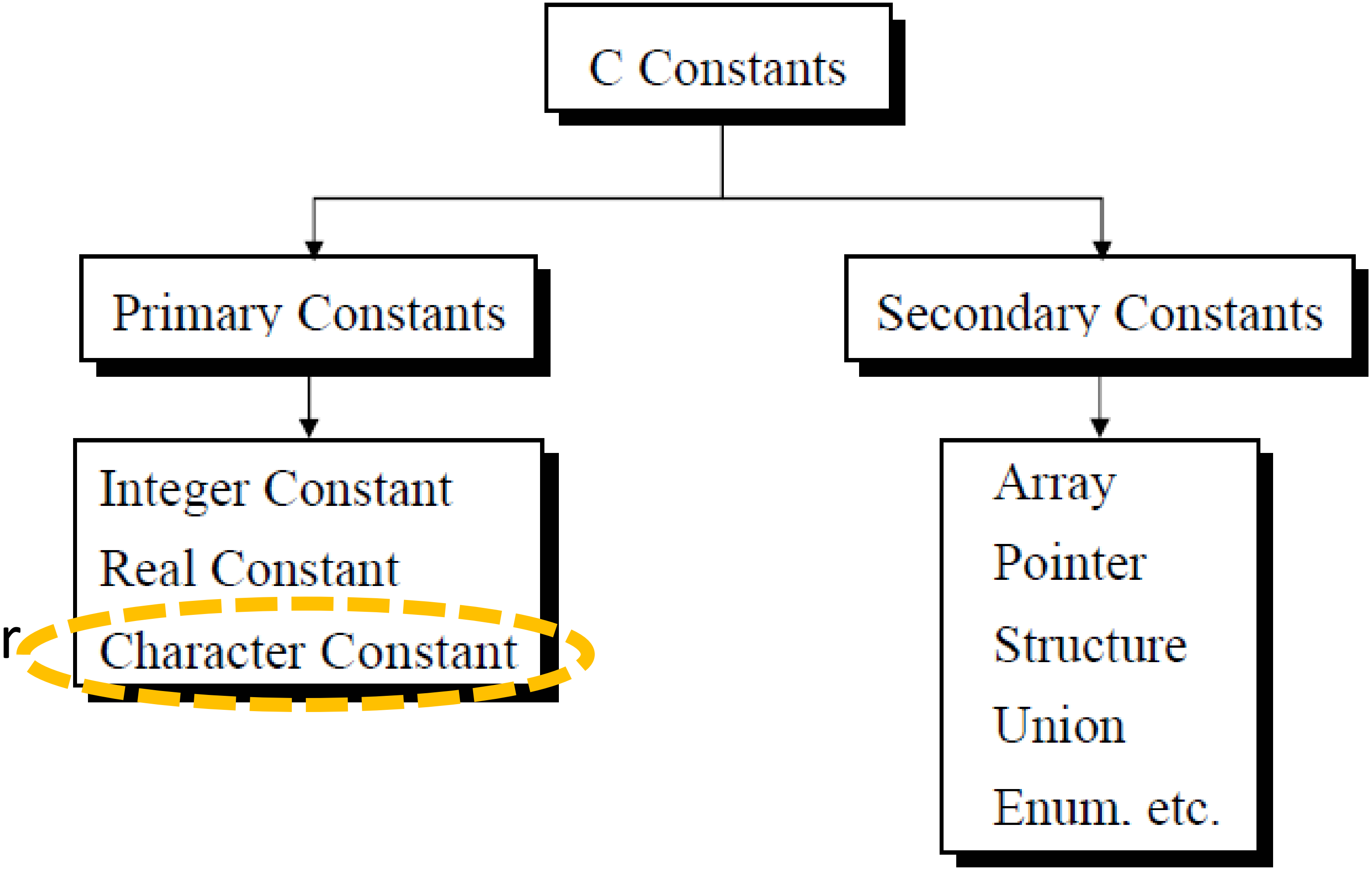
```
int x;
float y;
x = 3;
y = (float) x; /* Explicit casting
*/
y = x; /* Implicit casting */
```

How to represent text in computer?

- How to store A, B, C, …, $,@,… in RAM?

C Constants

Now focus on Character



C Constants

Primary Constants

Secondary Constants

Integer Constant

Real Constant

Character Constant

Array

Pointer

Structure

Union

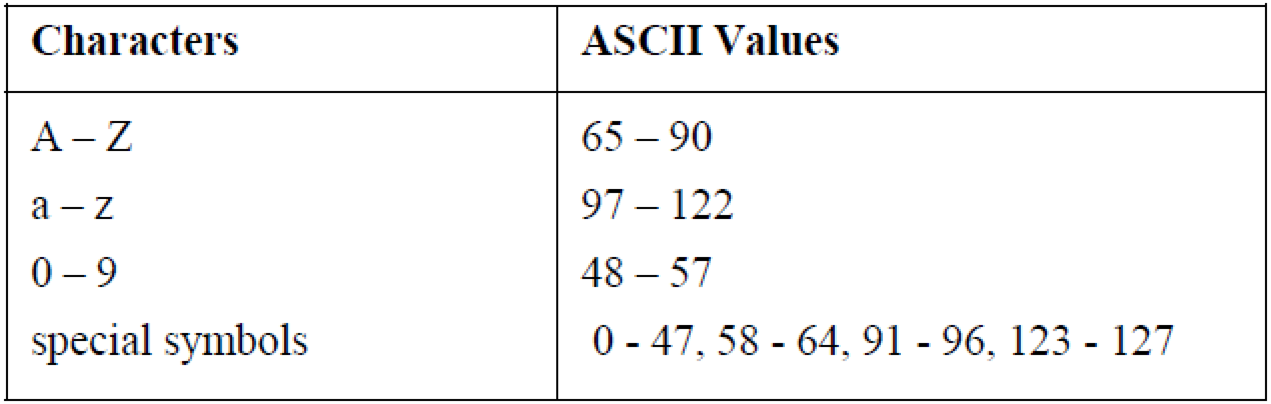
Enum. etc.

Character Constants

- Size 8 bits
- Like small integer (0-255)
- Rules for constructing character constant
  - a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
  - The maximum length of a character constant can be 1 character.
- Example
  - 'A'
  - 'I'
  - '5'
  - '='

How character is stored in memory

- Needs represent character by integer
- Needs a standard
  - American Standard Code for Information Interchange (ASCII)



Characters	ASCII Values
A – Z	65 – 90
a – z	97 – 122
0 – 9	48 – 57
special symbols	0 - 47, 58 - 64, 91 - 96, 123 - 127

ASCII

Value	Char	Value	Char	Value	Char	Value	Char	Value	Char	Value	Char
0		22	▬	44	,	66	B	88	X	110	n
1	☺	23	↨	45	-	67	C	89	Y	111	o
2	☻	24	↑	46	.	68	D	90	Z	112	p
3	♥	25	↓	47	/	69	E	91	[	113	q
4	♦	26	→	48	0	70	F	92	\	114	r
5	♣	27	←	49	1	71	G	93	]	115	s
6	♠	28	⌐	50	2	72	H	94	^	116	t
7	●	29	↔	51	3	73	I	95	_	117	u
8	◘	30	▲	52	4	74	J	96	`	118	v
9	○	31	▼	53	5	75	K	97	a	119	w
10	◙	32		54	6	76	L	98	b	120	x
11	♂	33	!	55	7	77	M	99	c	121	y
12	♀	34	"	56	8	78	N	100	d	122	z
13	♪	35	#	57	9	79	O	101	e	123	{
14	♫	36	$	58	:	80	P	102	f	124	|
15	☼	37	%	59	;	81	Q	103	g	125	}
16	►	38	&	60	<	82	R	104	h	126	~
17	◄	39	’	61	=	83	S	105	i	127	$^{\mathrm{M}}$H
18	↨	40	(	62	>	84	T	106	j	128	Ç
19	‼	41	)	63	?	85	U	107	k	129	ü
20	¶	42	*	64	@	86	V	108	l	130	é
21	§	43	+	65	A	87	W	109	m	131	â

ASCII

Value	Char	Value	Char	Value	Char	Value	Char	Value	Char	Value	Char
132	ä	154	Ü	176	░	198	╞	220	▄	242	≥
133	à	155	¢	177	▒	199	╟	221	▌	243	≤
134	å	156	£	178	▓	200	╚	222	▐	244	⌠
135	ç	157	¥	179	│	201	╔	223	▀	245	⌡
136	ê	158	Pts	180	┤	202	╩	224	α	246	÷
137	ë	159	ƒ	181	╡	203	╦	225	ß	247	≈
138	è	160	á	182	╢	204	╠	226	Γ	248	°
139	ï	161	í	183	╖	205	═	227	π	249	•
140	î	162	ó	184	╕	206	╬	228	Σ	250	·
141	ì	163	ú	185	╣	207	╧	229	σ	251	√
142	Ä	164	ñ	186	║	208	╨	230	µ	252	η
143	Å	165	Ñ	187	╗	209	╤	231	τ	253	²
144	É	166	ª	188	╝	210	╥	232	Φ	254	■
145	æ	167	º	189	╜	211	╙	233	θ	255	
146	Æ	168	¿	190	╛	212	╘	234	Ω		
147	ô	169	⌐	191	┐	213	╒	235	δ		
148	ö	170	¬	192	└	214	╓	236	∞		
149	ò	171	½	193	┴	215	╫	237	ø		
150	û	172	¼	194	┬	216	╪	238	ε		
151	ù	173	¡	195	├	217	┘	239	∩		
152	ÿ	174	«	196	─	218	┌	240	≡		
153	Ö	175	»	197	┼	219	█	241	±		

Character Variable

- Type char
- Format specifier %c

```
char a, b, d ;
a = 'F' ;
b = 'G' ;
d = '+' ;
```

- ASCII values of the characters are stored in the variables.

Arithmetic Operators for Real values

- +
- -
- *
- /
- % ??

Precedence of Operator

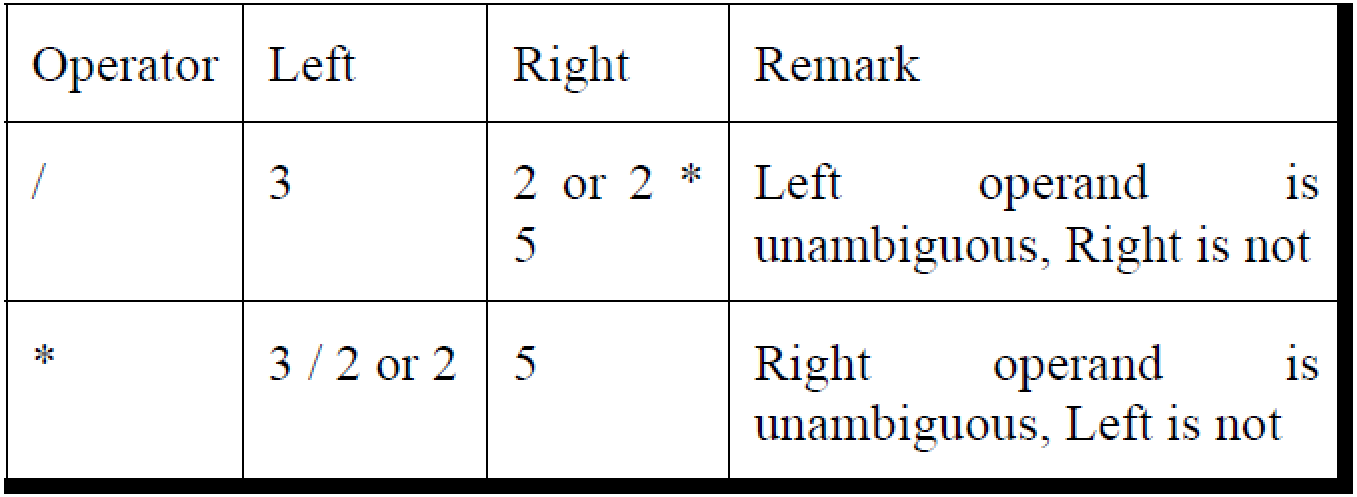
- When in a expression 2 or more operator
  - how exactly does it get executed?
  - Unfortunately, no simple rules such as “BODMAS”
- 2 * x - 3 * y
  - (2x)-(3y) ?
  - 2(x-3y)?
- Precedence/Priority: Which operator is applied when?
- *, / and % are higher in precedence that + and -
- Precedence can be altered by using parentheses
  - Innermost parentheses evaluated first
- For example-
  - 6+4/2 is 8
  - because ‘/’ has precedence over ‘+’
  - if we want the ‘+’ to work first, we should write- (6+4)/2

Associativity of Operators

- When an expression contains two operators of equal priority
  - the tie between them is settled using the associativity of the operators
- Two types—Left to Right or Right to Left
- Left to Right associativity means that the left operand must be unambiguous/ clear
  - must not be involved in evaluation of any other sub-expression

Associativity of Operators

- Consider `a = 3 / 2 * 5 ;`
  - Tie between between / and *
  - settled using the associativity of / and *
  - Both /,* have L to R associativity

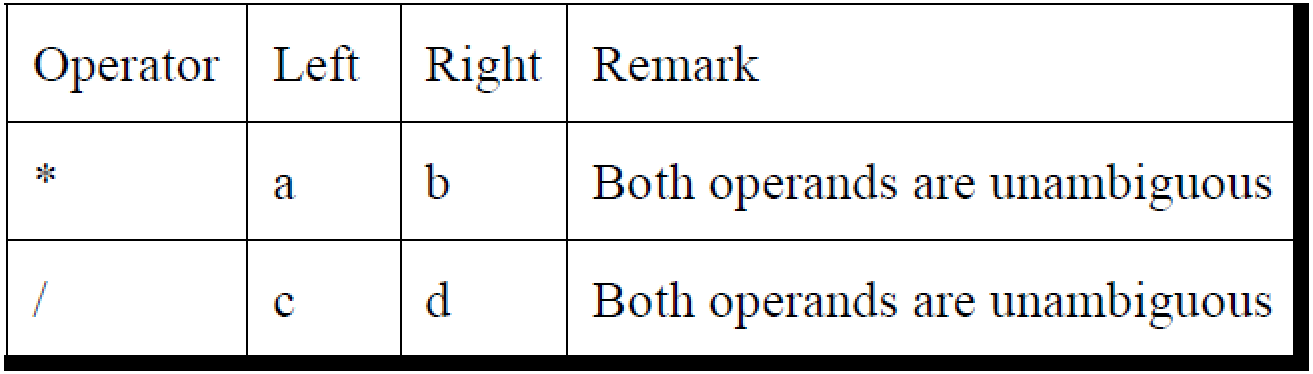


Operator	Left	Right	Remark
/	3	2 or 2 * 5	Left operand is unambiguous, Right is not
*	3 / 2 or 2	5	Right operand is unambiguous, Left is not

  - only / has unambiguous left operand
  - Result??

Associativity of Operators

- Consider `z = a * b + c / d ;`



Operator	Left	Right	Remark
*	a	b	Both operands are unambiguous
/	c	d	Both operands are unambiguous

- left operands for both operators are unambiguous
- Compiler is free to perform ***** or **/** operation as per its convenience
- no matter which is performed earlier the result would be same

By the grace of Allah we’ve finished

- Chapter 1
- Let Us C