CS F111: Computer Programming

(Second Semester 2020-21)

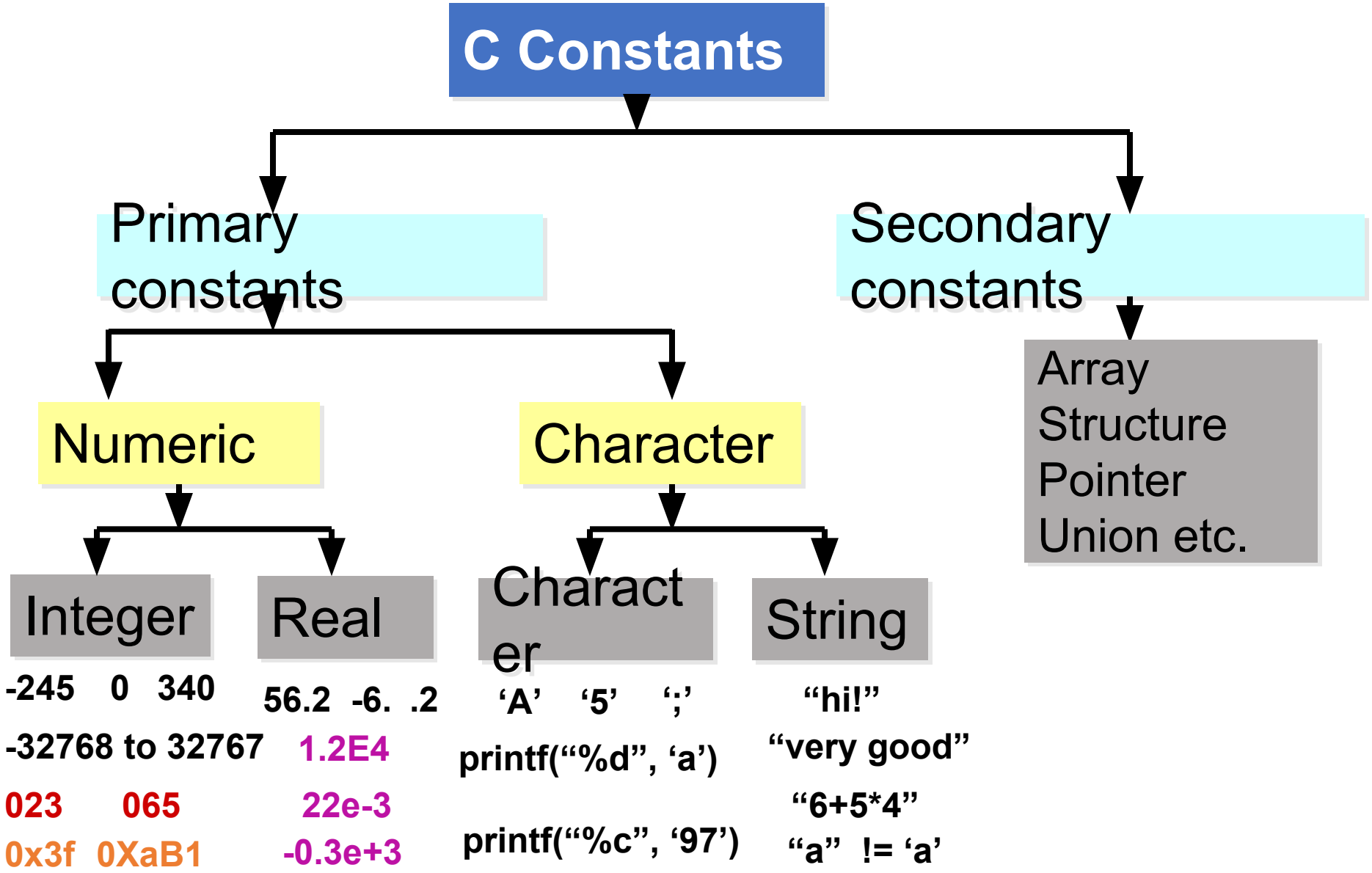
Lect 8: Constants and Operators

Dr. Nikumani Choudhury
Asst. Prof., Dept. of Computer Sc. & Information Systems
nikumani@hyderabad.bits-pilani.ac.in

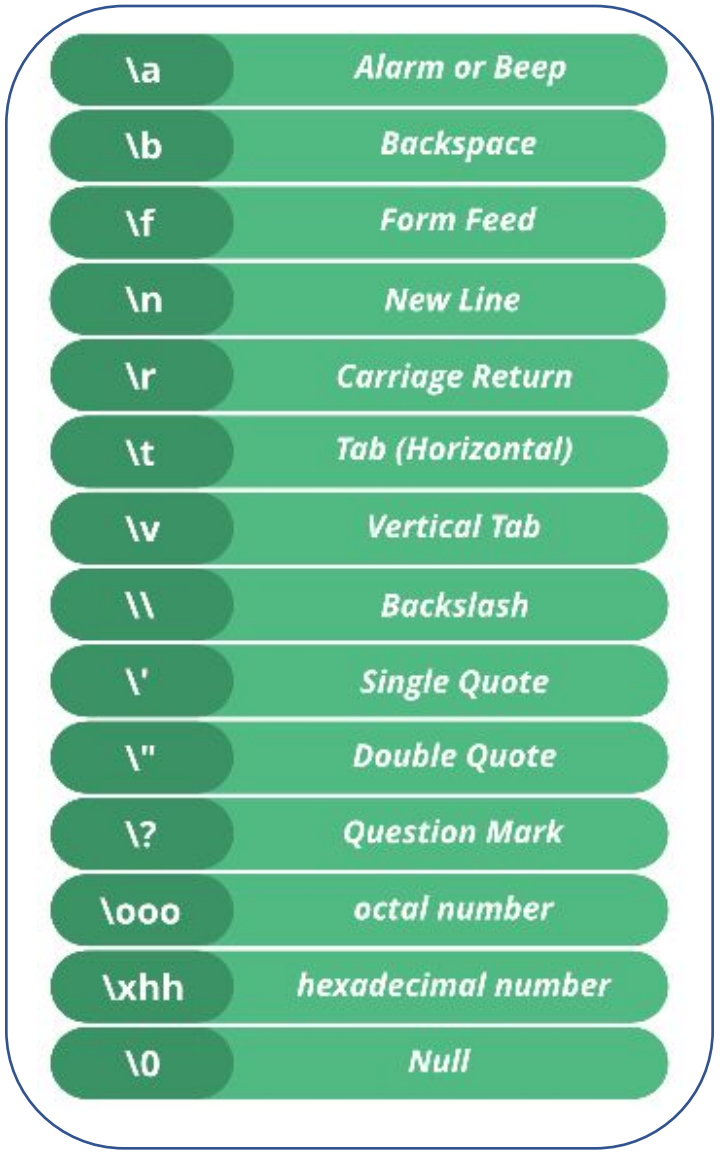
BITS Pilani

Hyderabad Campus

Constants


C Constants
Primary constants
Secondary constants
Numeric
Character
Array
Structure
Pointer
Union etc.
Integer
Real
Character
String
-245 0 340
56.2 -6. .2
'A' '5' ';'
"hi!"
-32768 to 32767
1.2E4
printf("%d", 'a')
"very good"
023 065
22e-3
"6+5*4"
0x3f 0XaB1
-0.3e+3
printf("%c", '97')
"a" != 'a'

Constants Continued…



Escape sequences

Are these two same?

‘a’

“a”

Integer Constants

1. Consists of a sequence of digits, with possibly a plus or a minus sign before it.
   - Embedded spaces, commas and non-digit characters are not permitted between digits.

- Maximum and minimum values (for 32-bit representations)

  Maximum :: 2147483647

  Minimum :: – 2147483648

  Range is -2^{31} to $2^{31} - 1$

Floating Point Constants

1. Can contain fractional parts
2. Very large or very small numbers can be represented.

23000000 can be represented as 2.3e7

Two different notations:

1. Decimal notation

25.0, 0.0034, .84, -2.234

2. Exponential (scientific) notation

3.45e23, 0.123e-12, 123E2

Character Constants

- Singular!
- One character defined in the ASCII character set.
- Surrounded by the single quotation mark.
  - 'A' , 'a', '$', '4'
  - Some special backslash characters (Escape sequences)
    - `'\n'` new line `'\t'` horizontal tab
    - `'\''` single quote `'\"'` double quote
    - `'\\'` backslash `'\0'` null

ASCII Value for Characters

- **ASCII code is the numerical representation of a character**

Characters	ASCII Value
‘A’ – ‘Z’	65 - 90
‘a’ – ‘z’	97 - 122
‘0’ – ‘9’	48 - 57
‘\0’	0

String Constants

- A sequence characters surrounded by double quotation marks.
- Considered a single item.
- Examples: (note word processor converts " to “ or ”
  - "India"
  - “I (?)like C Programming.”
  - “123”
  - “CAR”
  - “car”

Differences from character constants:

- 'C' and "C" are not equivalent.
- ‘C’ has an equivalent integer value while “C” does not.

Punctuation

- **Semicolons, colons, commas, apostrophes, quotation marks, brackets, braces and parentheses.**

; : , ' " [ ] { } ( )

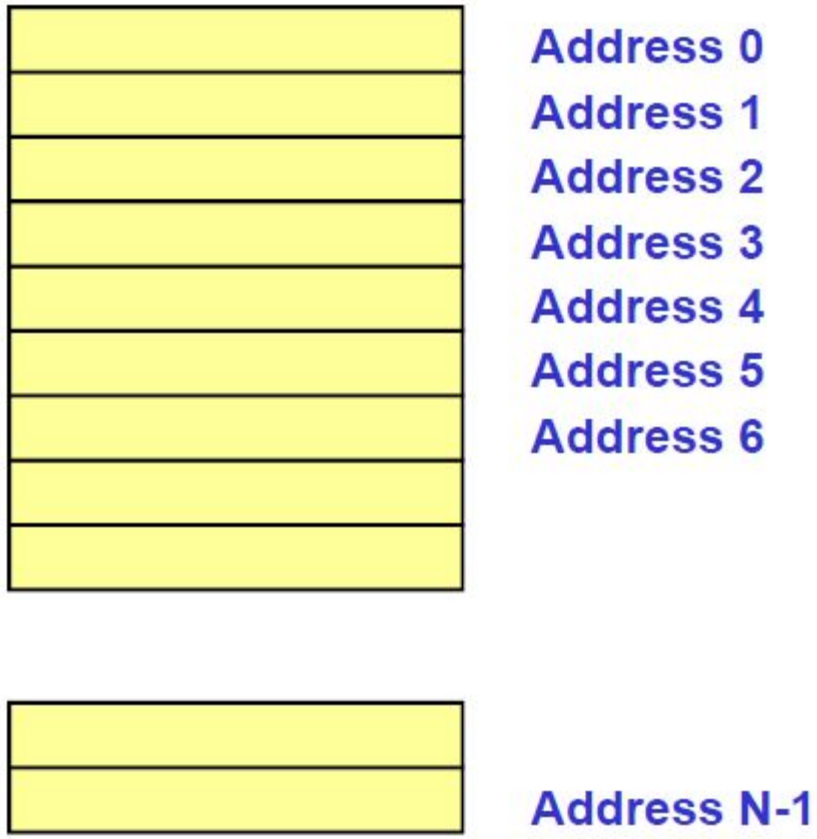
Variables and Constants

- **Most important concept for problem solving using computers.**

- **All temporary results are stored in terms of variables**
  - **The value of a variable can be changed.**
  - **The value of a constant do not change.**

- **Where are they stored ?**
  - **In main memory.**

Memory map



- list of storage locations
- every variable is mapped to a particular memory address

Data Types in C

Data Type	Description	Bytes (Typically)	Range
int	Stores integer	4	-2147483648 to 2147483647
char	Stores single character	1	-128 to 127
float	Stores real number	4	-3.4E+38 to +3.4E+38
double	Stores real number but with more precision than float	8	-1.7E+308 to +1.7E+308
void	Associated with no data type	-	

Depends on the compiler & its abstraction implementation

Data Type Qualifiers

- Usage : qualifier data_type
- Types of qualifiers:
  - short
  - long
  - signed
  - unsigned

Examples:		(ShortHand)	
1. unsigned int	⇔	unsigned	
2. short int	⇔	short	(typically 2 bytes)
3. long int	⇔	long	(typically 8 bytes)

Types of variable

- We must *declare the type of every variable we use in C.*
- Every variable has a *type (e.g. int) and a name.*
- Declarations of types should always done before use

1. int
2. char
3. float
4. double

Declaration of Variables

There are two purposes:

1. It tells the compiler what the variable name is.

2. It specifies what type of data the variable will hold.

General syntax:

data-type variable-list;

- **Examples:**
- **int salary, bonus;**
- **int x, y, z;**
- **float simple_interest;**
- **char ch, option;**

Format Specifier

Data Type	Format Specifier
int	%d
char	%c
float	%f
double	%lf
unsigned int	%u
long int	%ld or %li
long long int	%lld or %lli
long double	%Lf or %LF

Expressions

Represent

- **combination of data items interconnected by operators**

Ex :

```
a + b
x = y
c = a + b
x <= y
i++
```

Operators

There are operators for:

- Assignment
- Arithmetic
- Relational
- Logical
- Bitwise
- obtaining the size of an object (sizeof)
- obtaining the address of an object (&)
- referencing an object through its address (*)

Assignment Operator

Used to assign values to variables, using the assignment operator (=).

- **General syntax:**

  variable_name = expression;

- **A value can be assigned to a variable at the time the variable is declared.**

  int speed = 30;

  char flag = 'y';

- **Several variables can be assigned the same value using multiple assignment operators.**
  - **a = b = c = 10;**
  - **flag1 = flag2 = 'y';**
  - **speed = flow = 40;**

Continued…

Compound Assignments:

- **Used to simplify the code**
- **Shorthand Assignment**

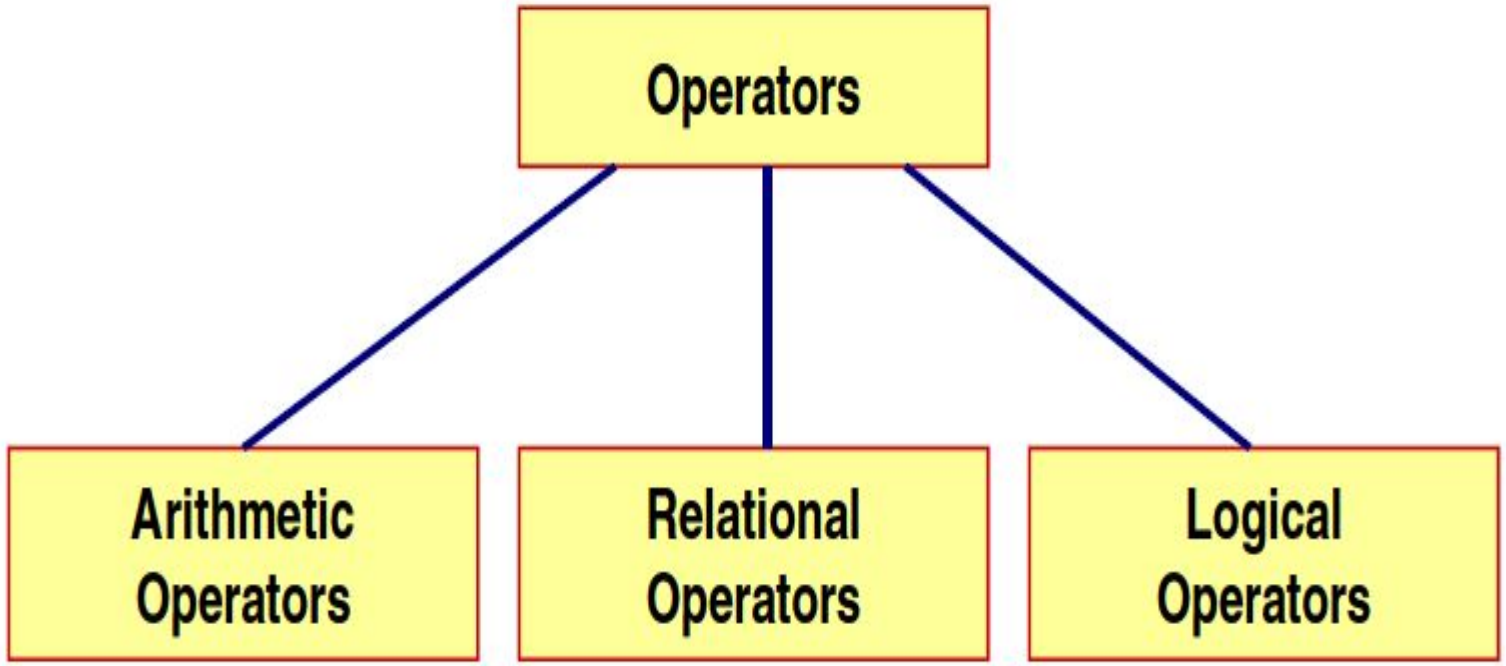
Ex1 : x = x + 10 □ x += 10

Ex2 : x = x – 100 □ x -= 100

var = var operator expr □ var operator= expression

Note: Don’t put space between operator and =

Operators in Expressions



Arithmetic Operators

- **Addition: +**
- **Subtraction: – (Can be Unary like x = -x;)**
- **Division: /**
- **Multiplication: ***
- **Modulus: %**
- **Decrement: -- (Unary)**
- **Increment: ++ (Unary)**

Operator Precedence

Operator category	*Operators*	*Associativity*
unary operators	- ++ -- ! sizeof (*type*)	R → L
arithmetic multiply, divide and remainder	* / %	L → R
arithmetic add and subtract	+ -	L → R
relational operators	< <= > >=	L → R
equality operators	== !=	L → R
logical *and*	&&	L→ R
logical *or*	||	L → R
assignment operators	= += -= *= /= %=	R → L

- **Parenthesis may be used to change the precedence of operator evaluation.**
- **Associativity - *order*** in which consecutive operations within the same precedence group are carried out

Examples: Arithmetic expressions

a + b * c – d / e ((a + (b * c)) – (d / e))

a * – b + d % e – f (((a * (– b)) + (d % e)) – f)

a – b + c + d (((a – b) + c) + d)

x * y * z ((x * y) * z)

a + b + c * d * e ((a + b) + ((c * d) * e))

Integer Arithmetic

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.

- Integer arithmetic always yields integer values.

Integer op Integer ? Integer

Modulus Operator

- Binary Operator
- Gives a remainder
- It works on only integer argument
- If number is negative, varies from one implementation to another

- But most compilers implement it in a = (a/b)*b + a%b
- Ex : 8%3 = 2, -8%3 = -2, -8%-3 = -2 , 8%-3 = 2

Arithmetic Operations

```
int a = 10 , b = 3;
```

Expression	Value
a + b	13
a – b	7
a * b	30
a / b	3
a % b	1

```
char c1 = 'A', c2 = '5'; // 'A' : 65, '5' : 53
```

Expression	Value
c1	65
c1 + c2	118
c1 + c2 + 5	123
c1 + c2 + '5'	171

Increment & Decrement Operators

- Unary Operator
- Prefix or postfix

Ex :

++x □ x = x + 1, x++ □ x = x + 1

--x □ x = x – 1, x-- □ x = x – 1

About Increment/Decrement

- Operator written before the operand (++i, --i))
  - Called pre-increment/decrement operator.
  - Operand will be altered in value *before* it is utilized for its intended purpose in the program.

- Operator written after the operand (i++, i--)
  - Called post-increment/decrement operator.
  - Operand will be altered in value *after* it is utilized for its intended purpose in the program.

Output

```
int a = 5;
int b = ++a;
printf("a = %d, b = %d\n", a, b);
```

```
int a = 5;
int b = a++;
printf("a = %d, b = %d\n", a, b);
```