CSE101-Lec#3

Components of C

Identifiers and Keywords

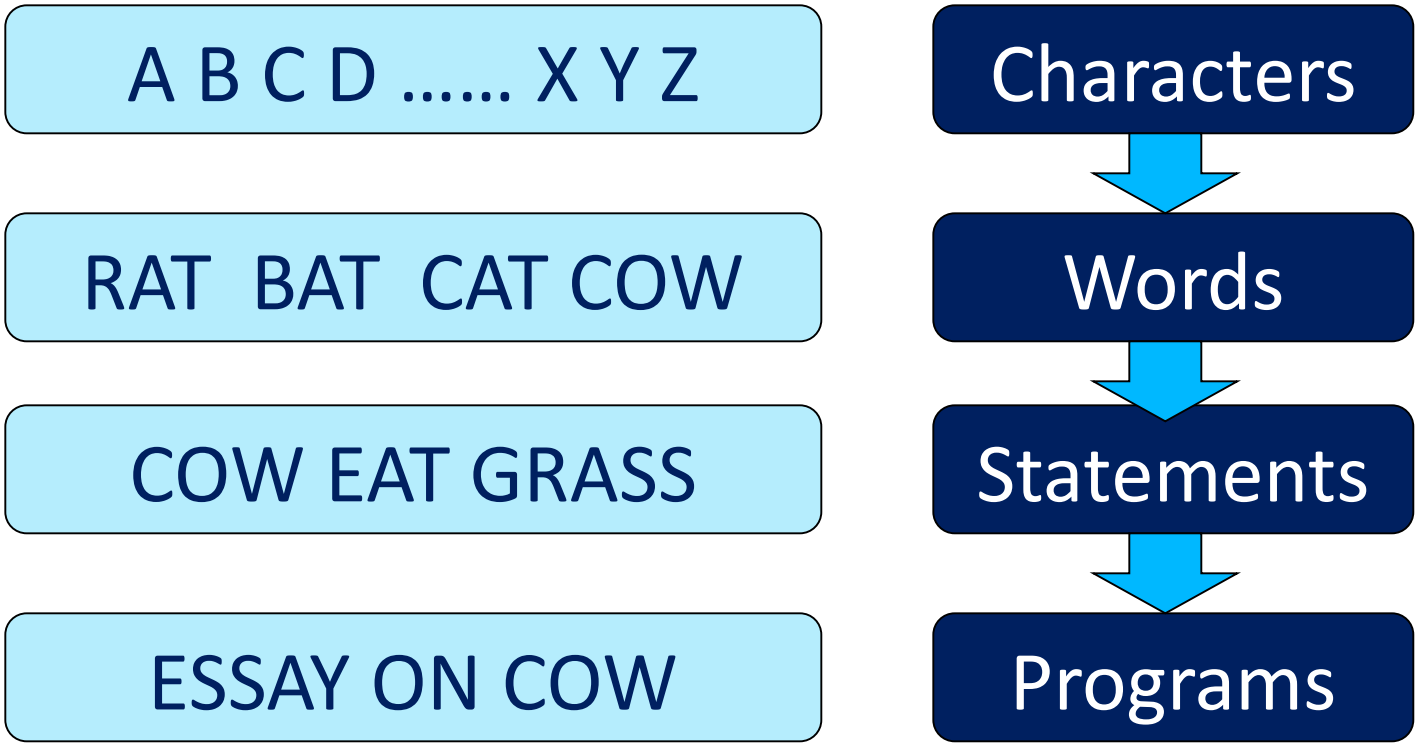
Data types

OUTLINE

- In this lecture we will cover
  - Character set
  - Identifiers
  - Keyword
  - Data types

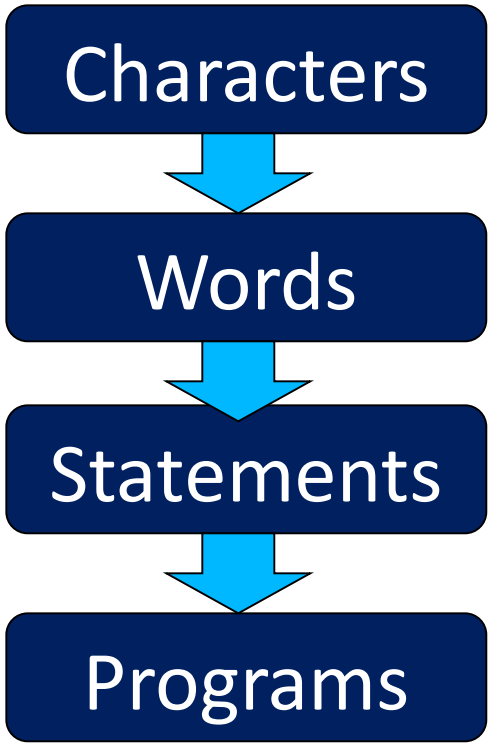
Language: its influence in our life

- Let us look to what we are doing since our childhood, how did we learnt ENGLISH- A recap



Introduction to C

- Like every language C programming language requires basic building blocks to communicate with the computer.
- So we require
  - Character set
  - Words(keywords and identifiers)
  - Statement (instructions)
  - Program



Character Set

- The character set of C represents alphabet, digit or any symbol used to represent information.

Types	**Character Set**
Uppercase Alphabets	A, B, C, … Y, Z
Lowercase Alphabets	a, b, c, … y, z
Digits	0, 1, 2, 3, … 9
Special Symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /
White spaces	Single space, tab, new line.

Token

- Every single element in a C Program is Token

Token

- **Smallest unit in a program/statement.**
- It makes the compiler understand what is written in the program.
- Example: main, printf , name,), etc.
- Tokens are broadly classified as:
  - Identifiers
  - Keywords
  - Constants
  - Variables
  - Strings
  - Operators
  - Special character

Identifiers

- So to identify things we have some name given to them .
- Identifiers are the fundamental building blocks of a program
- Used to give **names** to variables, functions, constant, and user defined data.
- They are user-defined names and consist of a sequence of letters and digits

Rules for naming an Identifier

1. An identifier name is any combination of 1 to 31 alphabets, digits or underscores.

2. The first character in the identifier name must be an alphabet or underscore.

3. No blanks or special symbol other than an underscore can be used in an identifier name.

4. Keywords are not allowed to be used as identifiers.

Some Identifiers

Tool_spanner;
tool_spanner;

both are different

FORMULA1;

engine_1;

Wrong identifiers name

1_engine;

break;

@car-roof;

C Keywords

- Keywords are the reserved words whose meaning has already been explained to the C compiler.
- We cannot use these keywords as variables.
- Each keyword is meant to perform a specific function in a C program.
- There are 32 keywords in C language.
- All keywords are written in lowercase only

Eg: The **name** of person can never be **home**, **eat**, **sleep**, **run**, etc because these words have some predefined meaning to perform some task.

List of C Keywords

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

Data Types

- Data type means the type of value a variable will have.
- It also defines memory space for a particular variable in computer.
- The type of value of variable can be alphabets or numbers.
- The numbers can be further divided as the integer or rational number.

- Lets see a mathematics problem:

My-Car

1. If the radius of car wheel is 15inch then what will the distance traveled after one rotation of that wheel?

Sol: Given-

radius = 15 — 15 — Integer(int in C)

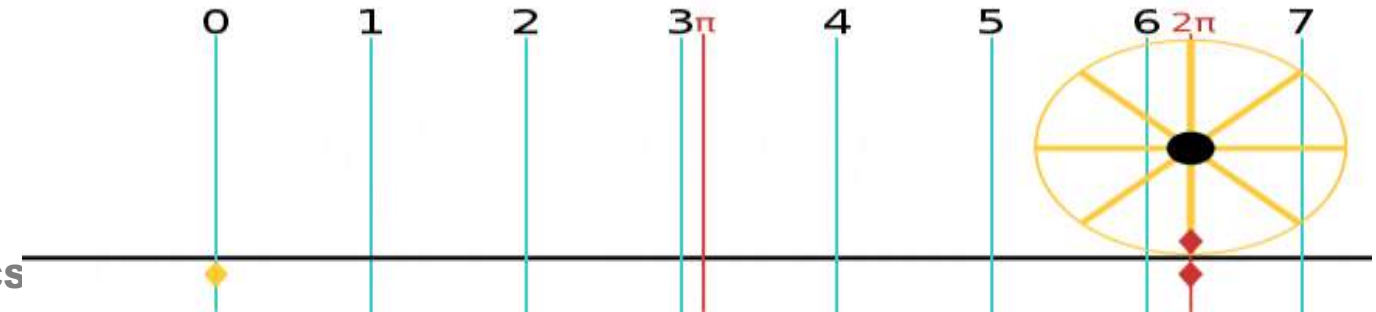
dist_travelled = ?

So, Circumference of circle = 2 * pi * r

dist_travelled = 2 * 3.14 * radius — 3.14 — Real (float in C)

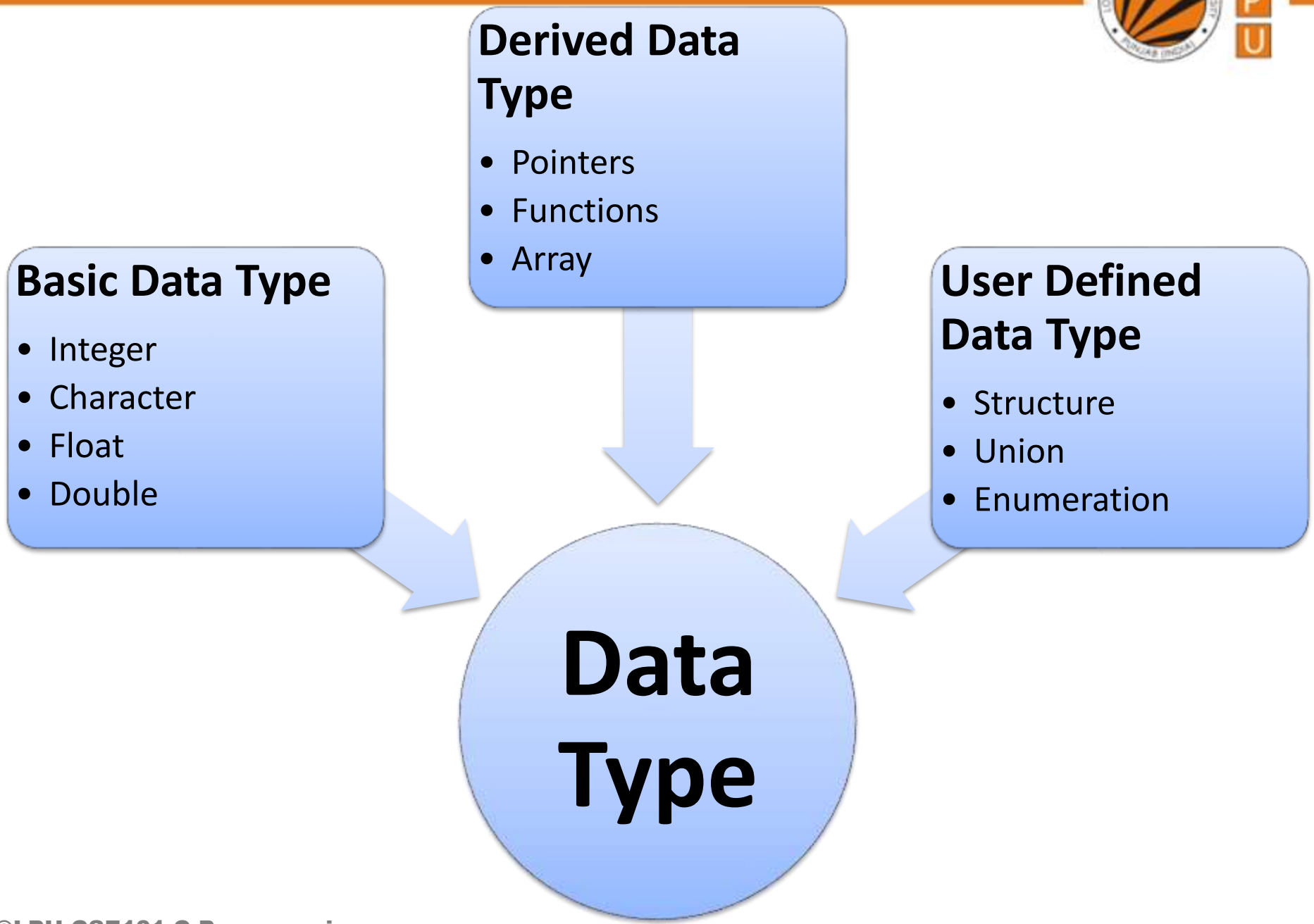
dist_travelled = 6.28 * 15

dist_travelled = 94.2 Ans. — 94.2 — Real (float in C)



Classification of Data Types

- In C data type is broadly classified as:
  - Basic data types
  - Derived data types
  - User defined data types


Derived Data Type
• Pointers
• Functions
• Array
Basic Data Type
• Integer
• Character
• Float
• Double
User Defined Data Type
• Structure
• Union
• Enumeration
Data Type

List of Data Types

Type	Size (bytes)	Minimal range
char	1	-128 to 127
unsigned char	1	0 to 255
int	4	-32768 to 32767
unsigned int	4	0 to 65535
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	3.4e-38 to 3.4e+38 with 6 digits of precision
double	8	1.7e-308 to 1.7e+308 with 15 digits of precision
long double	10	3.4e-4932 to 1.1e+4932 with 20 digits of precision

Integer

- It is used to store positive and negative counting numbers, as well as zero.

{...,-2,-1,0,1,2,...}

- The numbers written in green box of My-Car problem are the integers.

15 84 34 97

- The **type modifiers** for the integer data type are: signed, unsigned, short, long .
- Signed types represent positive and negative numbers.
- Unsigned represent zero and positive numbers only.
- Long and short represent the range of integer number

Short Integer

- Occupies 2 bytes in memory.
- Format specifier is %d or %i.
- Range is -32768 to 32767.
- By default int variable is short signed int.

```
int cost=100;
short int si;
```

Long Integer

- Occupies 4 bytes in memory.
- Format specifier is %ld.
- Range is -2147483648 to 2147483647

```
long radius=123456;
long int value;
```

Signed Integer

- Occupies 2 bytes in memory
- Format specifier is %d or %i
- There are also long signed integers having range from -2147483648 to 2147483647
- Example:

```
int firstvalue=10;
long int WaterLevel;
```

Unsigned Integer

- Occupies 2 bytes in memory
- Format specifier is %u.
- There are also long unsigned int with range 0 to 4294967295
- Example:

```
unsigned long count=567898;
unsigned short int page;
```

Float

- Floating point numbers are real numbers that, unlike integers, may contain fractional parts of numbers, like 1.446, -112.972, 3.267e+27.
- It is used to store real numbers with single precision i.e. a precision of 6 digits after decimal point.

- Format specifier is **%f.**
- The **type modifier** for float are float, double and long double.

- The rational number written in red box of My-Car problem are the float numbers.

3.14 94.2

Type	Float	Double	Long double
Storage Size	4 byte	8 byte	10 byte
Value range	3.4e-38 to 3.4e+38	1.7e-308 to 1.7e+308	3.4e-4932 to 1.1e+4932
Precision	6 decimal places	15 decimal places	20 decimal places
Example	pi=3.141592	3.141592741012573	3.141592653589793238 46

Character

- It stores a single character of data belonging to the C character set.

- The alphabets written in blue box of My-Grades problem are the character.

A	D	A	B	C

- It occupies 1 byte of memory.
- Format specifier is **%c**.
- The range is 0 to 255 for unsigned char.
- The range is -127 to 127 for signed char.
- Each char type has an equivalent integer interpretation, ASCII value, so that a char is really a special kind of short integer.

char choice='y';

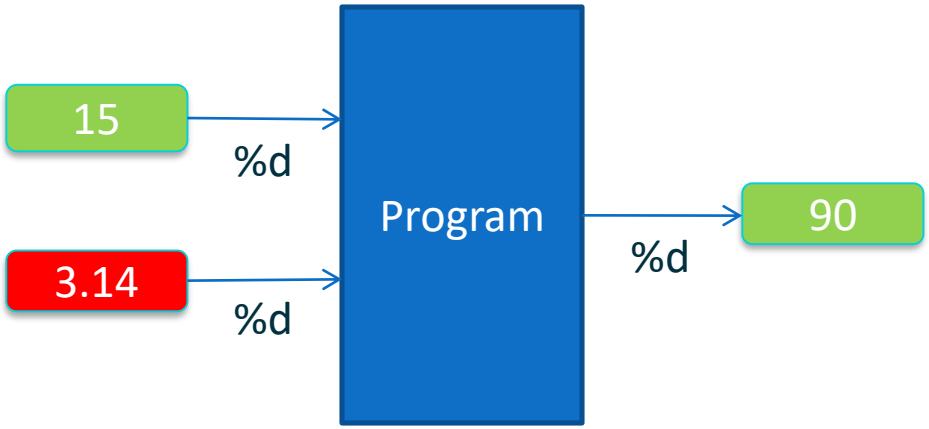
Format Specifier

- Specifies the format according to which the value will be printed on screen in C.

Example:

- %d : signed integer
- %ld: long integer
- %u : unsigned integer
- %c : single character
- %f : float
- %s : string
- %i : int

- Remember car example?



My-Car

1. If the radius of car wheel is 15inch then what will the distance traveled after one rotation of that wheel?

Sol: Given-

radius = 15 inch — 15 Integer(int in C)

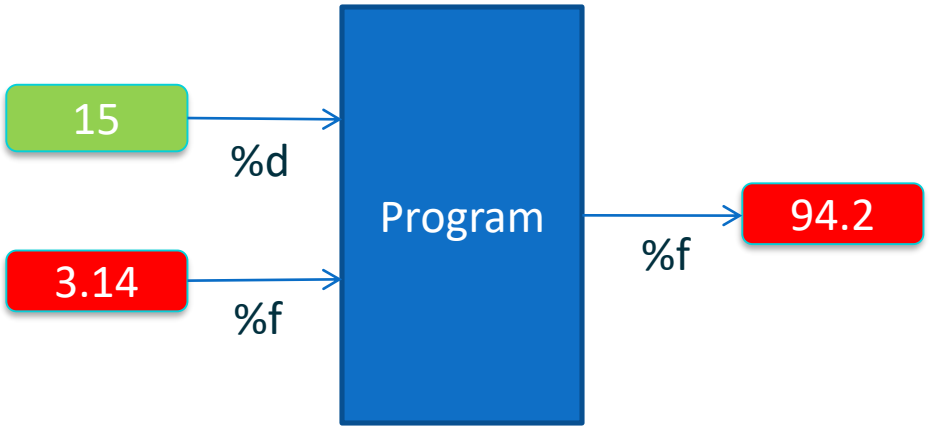
dist_travelled = ?

So, Circumference of circle = 2 * pi * radius

dist_travelled = 2 * 3.14 * radius — 3.14 Real (float in C)

dist_travelled = 6.28 * 15

dist_travelled = 94.2 inch Ans. — 94.2 Real (float in C)



MCQ

A program is made up of individual syntactic elements called

- A)Classes
- B)Functions
- C)Tokens
- D)None of them

MCQ

Q What is the output of this C code?

```
int main()
{
float x = 'a';
printf("%f", x);
return 0;
}
```

A. a

B. run time error

C. a.0000000

D. 97.000000

MCQ

Q Which of the following is not a valid variable name declaration?

a) int _v1;

b) int v_1;

c) int 1_v;

d) int _1v

MCQ

Q What will be the output?

- #include <stdio.h>
  int main()
  {
  int main = 5;
  printf("%d", main);
  return 0;
  }
  a) compile-time error
  b) run-time error
  c) run without any error and prints 5
  d) experience infinite looping

Next Lecture: Constants
Variables
Expressions
Operators

cse101@lpu.co.in

CSE101-Lec#4

Constant
Variable
Expression
Operators

Tokens

- We have seen that Tokens are broadly classified as:
  - Identifiers
  - Keywords
  - **Constants**
  - **Variables**
  - Strings
  - Operators
  - Special character

Constants

- The entity which do not change throughout the execution are called constants.
- Types of constants:
  - Integer constant
  - Character constant
  - Floating point constants
  - String constants

Name of person remains same through out the life, example: Amit, Shubnam, etc.

- **Character constants**
  - Constants enclosed in single quotes(‘ ’).
  - It can be any letter from character set.

Example : ‘\n’, ‘\t’ or ‘a’

- **String Constants**
  - Set of zero or more characters enclosed in double quotes (eg: “ ”)
  - It is represented as sequence of characters within double quotes.

Example : “This is C programming”

- **Integer Constants**
  - When the constant contains only digits without any decimal part

Example : 5;
-987;

- **Floating Constant**
  - Constants that contains number with decimal points

Example : 3.14;
309.89

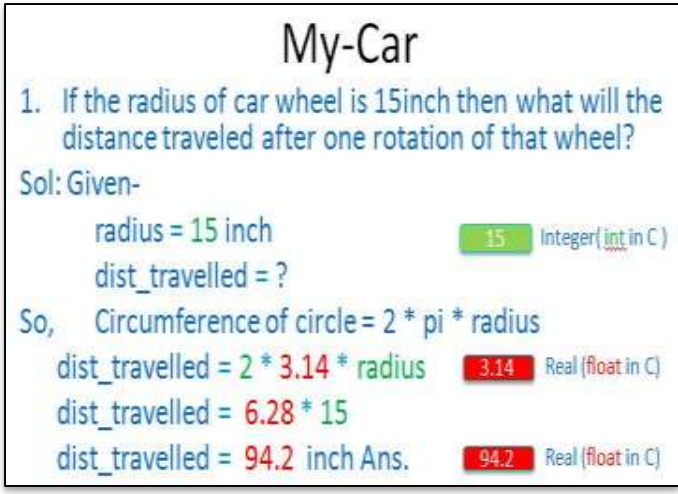
My-Car

In My-Car problem the constant value is 3.14 which is the value of pi and always same.

- pi = 3.14

Therefore:

dist_travelled = 2 * pi * radius.



➢ pi is a floating point constant.

Variables

- Variable is an entity which may change.
- Variable is used to hold result and reserve memory for the data.

Syntax

```
datatype variable_name;
```

The naming of variable is done by following the same rules of identifier naming.

Eg. What is your **hobby**?
The answer could be **reading**, **dancing**, **drawing**, etc.
So the answer to such questions may change during the life time of the person

Rules for naming a Variable

1. An variable name is any combination of 1 to 31 alphabets, digits or underscores.

2. The first character in the variable name must be an alphabet or underscore.

3. No blanks or special symbol other than an underscore can be used in an variable name.

4. Keywords are not allowed to be used as variables.

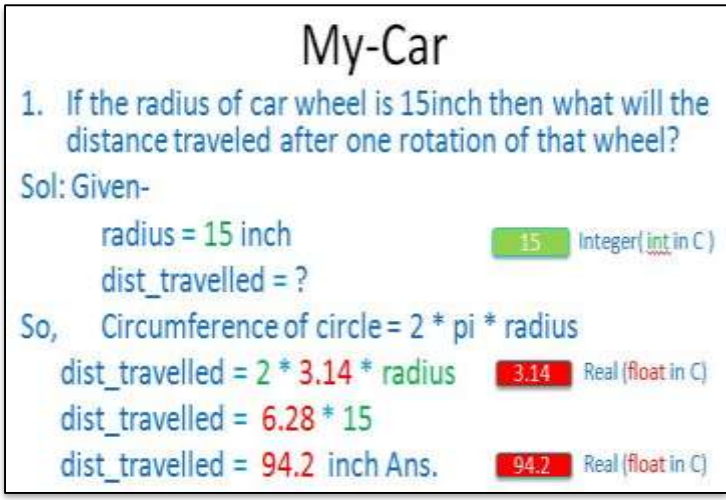
Variables

In My-Car problem the variable was

- radius and dist_travelled

It can also be named as

- radius_wheel or r1 or car_wheel_radius
- Distance or d1 or dist_by_1rotation



Variables

Let us build some variables:

For speed of car we need to know

- Distance traveled
- Time taken to travel the distance

$$s1 = d1 \div t1$$

Variables to be declared as

- Speed, **s1**, speed_of_car
- Distance, **d1**, dist
- Time, **t1**, time_of_travel

Variable Initialization

- Assigning some value to the variable at time of creation of variable is known as variable initialization.

Syntax

```
datatype variable_name =  value;
```

Example: int radius= 15;
float pi = 3.14;
char grade = 'A';

Expressions

- Expressions are the statements or the instruction given to computer to perform some operation.
- Every expression results in some value that can be stored in a variable.
- Following are few example of expressions in program:
  - Expression to calculate speed of a car.
    - Speed=distance/time
  - Expression to find similarity of two things.
    - c=value1>value2

- Expressions in C are basically **operators** acting on **operands**.
- An **operand** is an entity on which operation is to be performed.

Example: addition of two numbers, 5+8, these numbers will be operands.

- An **operator** specifies the operation to be applied on operands.

Example: The addition, subtraction, etc will be operators

- Expressions are made of one or more operands.
- Statements like :

  a = b + c,

  300 > (8 * k)

Types of Expressions

- The type of expression depend upon the type of operator used in the expression.
- It can be:
  - Arithmetic operators.

    3 + 6 = 9

    4 * 2 = 8
  - Relational or logical operators.

    height_boy>=height_girl
  - Increment and decrement operator.

    count=count++

Find the output of the code

```
#include <stdio.h>
int main()
{
 int i = 1, 2, 3;
 printf("%d", i);
 return 0;
 }
```

a)1 c) prints 1,2,3

b)3 d) compile time error

- In this lecture we will study
  - Operators
  - Types of Operators

Operators

- Operator is the symbol which performs some operations on the operands.

5+5=10

+ and = are the operator and
5 and 10 are operands

Types of Operators

- **Types of operators are:**
  1. Arithmetic operator
  2. Unary operator
  3. Relational operator
  4. Logical operator
  5. Assignment operator
  6. Conditional operator
  7. Bitwise operator
  8. Special operator

Next Class: Operators

Types of operators...contd.

cse101@lpu.co.in

CSE101-Lec#5

Operators

Operators

- Operator is the symbol which performs some operations on the operands.

5+5=10

+ and = are the operator and
5 and 10 are operands

Types of Operators

- **Types of operators are:**
  1. Arithmetic operator
  2. Unary operator
  3. Relational operator
  4. Logical operator
  5. Assignment operator
  6. Conditional operator
  7. Bitwise operator
  8. Special operator

Description of Operators

➢ Arithmetic Operators

These are binary operators i.e. expression requires two operands

Operator	Description	Example (a=4 and b=2)
+	Addition of two operands	a + b = 6
-	Subtraction of two operands	a – b = 2
*	Multiplication of two operands	a * b = 8
/	Division of two operands	a / b = 2
%	Modulus gives the remainder after division of two operands	a % b = 0

MCQ

- **Operator % in C Language is called.?**

A) Percentage Operator

B) Quotient Operator

C) Modulus

D) Division

MCQ

- **Choose a right statement.**

int a = 10 + 4.867;

A) a = 10

B) a = 14.867

C) a = 14

D) compiler error.

Unary Operator

These operator requires only one operand.

Operator	Description	Example(count=1)
+	unary plus is used to show positive value	+count; value is 1
-	unary minus negates the value of operand	-count; value is -1
++	Increment operator is used to increase the operand value by 1	++count; value is 2 count++; value is 2
--	Decrement operator is used to decrease the operand value by 1	--count; value is 1 count--; value is 1

++count increments count by 1 and then uses its value as the value of the expression. This is known a **prefix operator**.
count++ uses count as the value of the expression and then increments count by 1. This is known as **postfix operator**.

MCQ

```
int main()
{
int c=--2;
printf("c=%d", c);
return 0;
}
```

a)1 b) -2 c) 2 d)error

MCQ

```
int main()
{
  int x = 4, y, z;
  y = --x;
  z = x--;
  printf("%d, %d, %d\n", x, y, z);
  return 0;
}
```

a) 4,3,3 b)3,3,3 c)2,3,3 d)4,4,3

Unary Operators

Q: In an exam there was 10 question each carry 1 mark for right answer and 0.50 marks were deducted for wrong answer. A student attempted 6 questions and out of that 5 questions were wrong. So what is the score of the student out of 10?

Sol: No. of questions attempted = 6

Marks deducted = 5 * 0.50 = 2.5

Marks for right answer = 1

Total marks = 1 – 2.5 = -1.5

Unary Minus indicates that value is negative.

Relational Operator

It compares two operands depending upon the their relation. Expression generates zero(false) or nonzero(true) value.

Operator	Description	Example (a=10 and b=20)
<	less than, checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) value is 1(true)
<=	less than or equal to, checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) value is 1 (true).
>	greater than, checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) value is 0 (not true).
>=	greater than or equal to, checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) value is 1 (true).
==	equality ,checks if the value of two operands is equal or not, if yes then condition becomes true.	(a == b) value is 0 (not true).
!=	inequality, checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(a != b) value is 1 (true).

Relational Operator

Q: Age of Sam is 20 and age of Tom is 19.

Verify the relationship between their age.

Sol: age of Sam = S1 = 20

age of Tom = T1 = 19

S1 < T1 = 0 (false)

S1 == T1 = 0 (false)

S1 > T1 = 1 (true)

So, Sam is elder than Tom.

Logical Operator

It checks the logical relationship between two expressions and the result is zero(false) or nonzero(true).

Operator	Description	Example
&&	Logical AND operator. If both the operands are true then condition becomes true.	(5>3 && 5<10) value is 1 (true).
||	Logical OR Operator. If any of the two operands is true then condition becomes true.	(5>3 || 5<2) value is 1 (true).
!	Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(8==8) value is 0 (false).

Assignment Operator

They are used to assign the result of an expression on right side to a variable on left side.

Operator	Description	Example(a=4 and b=2)
+=	a=a+b	a+=b; a=a+b = 6
-=	a=a-b	a-=b; a=a-b = 2
*=	a=a*b	a*=b; a=a*b = 8
/=	a=a/b	a/=b; a=a/b = 2
%=	a=a%b	a%=b; a=a%b = 0
<<=	a=a<<b	a=00000100 << 2 = 00010000
>>=	a=a>>b	A=00000100 >> 2 = 00000001
&=	a=a&b	(a=0100, b=0010) a&=b; a=a&b = 0000
|=	a=a|b	(a=0100, b=0010) a|=b; a=a|b =0110
^=	a=a^b	(a=0100, b=0010) a^=b; a=a^b = 0110

Assignment Operator

- To increase the cost of item soap by 50rs.

  Cost_soap = Cost_soap + 50;

  or Cost_soap += 50;

- To double the quantity of water in a bowl.

  Water_inBowl *= 2;

✓ Therefore assignment operator are used to store the changed value of the variable in the same variable.

➢ Conditional Operator

Conditional operator contains condition followed by two statements. If the condition is true the first statement is executed otherwise the second statement.

It is also called as **ternary operator** because it requires three operands.

Operator	Description	Example
?:	conditional expression, Condition? Expression1: Expression2	(a>b)? “a is greater”: “b is greater”

Conditional Operator

- Eligibility to cast vote

  (age>=18)? “can cast vote”: “cannot cast vote”;

- In C

  (age>=18)? printf(“can cast vote”) : printf(“cannot cast vote”);

MCQ

Choose a syntax for C Ternary Operator from the list.

A) condition ? expression1 : expression2

B) condition : expression1 ? expression2

C) condition ? expression1 < expression2

D) condition < expression1 ? expression2

What is the output of the C statement.?

```
int main()
 {
 int a=0;
 a = 5<2 ? 4 : 3;
 printf("%d",a);
 return 0;
}
```

A) 4 B) 3 C) 5 D) 2

Bitwise Operator

A bitwise operator works on each bit of data.

Logical Table				
a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Operator	Description	Example(a=1 and b=0)
&	bitwise AND	a & b = 0
|	bitwise OR	a| b = 1
^	bitwise XOR	a ^ b = 1
~	bitwise one's complement	~a = 0, ~b=1
<<	bitwise left shift, indicates the bits are to be shifted to the left.	1101 << 1 = 1010
>>	bitwise right shift, indicates the bits are to be shifted to the right.	1101 >> 1 = 0110

Some Special Operators

Operator	Description	Example
,	comma operator, can be used to link the related expressions together	int a, b, x;
sizeof ()	sizeof operator to find the size of an object.	int a; sizeof(a)=2
type	Cast operator, to change the data type of the variable	float x= 12.5; int a; a = (int) x; value of a is 12.

MCQ

Choose a right statement.

```
int main()
 {
 float c = 3.5 + 4.5;
printf("%d", (int)c);
 return 0;
 }
```

A) 8.0 B) 8.000000 C) 7 D) 8

Precedence of Operators

- The precedence of operators determine a rank for the operators. The higher an operator's precedence or priority, the higher ßinding"it has on the operands.

Example: So how the expression a * b + c will be interpreted?

(a * b) + c or a * (b + c),

here the first interpretation is the one that is used because the multiplication operator has higher precedence than addition.

Associativity of Operators

- Associativity tell us the order in which several operators with equal precedence are computed or processed in two directions, either from left to right or vice-versa.

Example: In the expression

a * b / c,

since multiplication and division have the same precedence we must use the associativity to determine the grouping. These operators are left associative which means they are grouped left to right as if the expression was

(a * b) / c.

Operator	Associativity	Type
() [] . -> ++(postfix) - - (postfix)	left to right	Highest
+ - ++ -- ! & * ~ sizeof (type)	right to left	Unary
* / %	left to right	multiplicative
+ -	left to right	additive
<< >>	left to right	shifting
< <= > >=	left to right	relational
== !=	left to right	equality
&	left to right	bitwise AND
^	left to right	bitwise OR
|	left to right	bitwise OR
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= &= |= ^= <<= >>= %=	right to left	assignment
,	left to right	comma

Next Class: Control Structures

cse101@lpu.co.in