Module-3
C Introduction

1. **Keywords**
2. **Identifiers**
3. **Variables, Constants, Literals**
4. **Datatypes & Type conversions**
5. **Operators**
6. **Comments**

Keywords

What you will Learn ?

a = 100

int a = 100;

Keywords

Keywords are predefined, reserved words used in Python programming.

We cannot use a keyword as a variable name, function name, or any other identifier.

List of Keywords

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

Identifiers

Identifiers

Identifiers are the name given to variables, classes, methods, etc.

Variable Identifiers

```
int age;
float temperature;
char initial;
double salary;
long studentID;
unsigned int flags;
short count;
```

Method/Function Identifiers

```c
int add(int a, int b) {
    return a + b;
}

float calculateCircleArea(float radius) {
    return 3.14159265 * radius * radius;
}

void printMessage() {
    printf("Hello, World!\n");
}
```

Rules

- Start with a Letter or Underscore
- Followed by Letters, Digits, or Underscores
- Avoid Starting with a Digit
- No Special Characters(!, @, $, etc.)
- Case-Sensitive (e.g: age, Age, and AGE)
- Avoid Using Keywords

Valid & Invalid Identifiers

- name
- age
- _count
- total_price
- is_student
- calculate_area
- student_info
- MAX_VALUE

- 3d_model (starts with a digit)
- @result (contains special character)
- for (a Python keyword)
- my-name (contains a hyphen, not allowed)

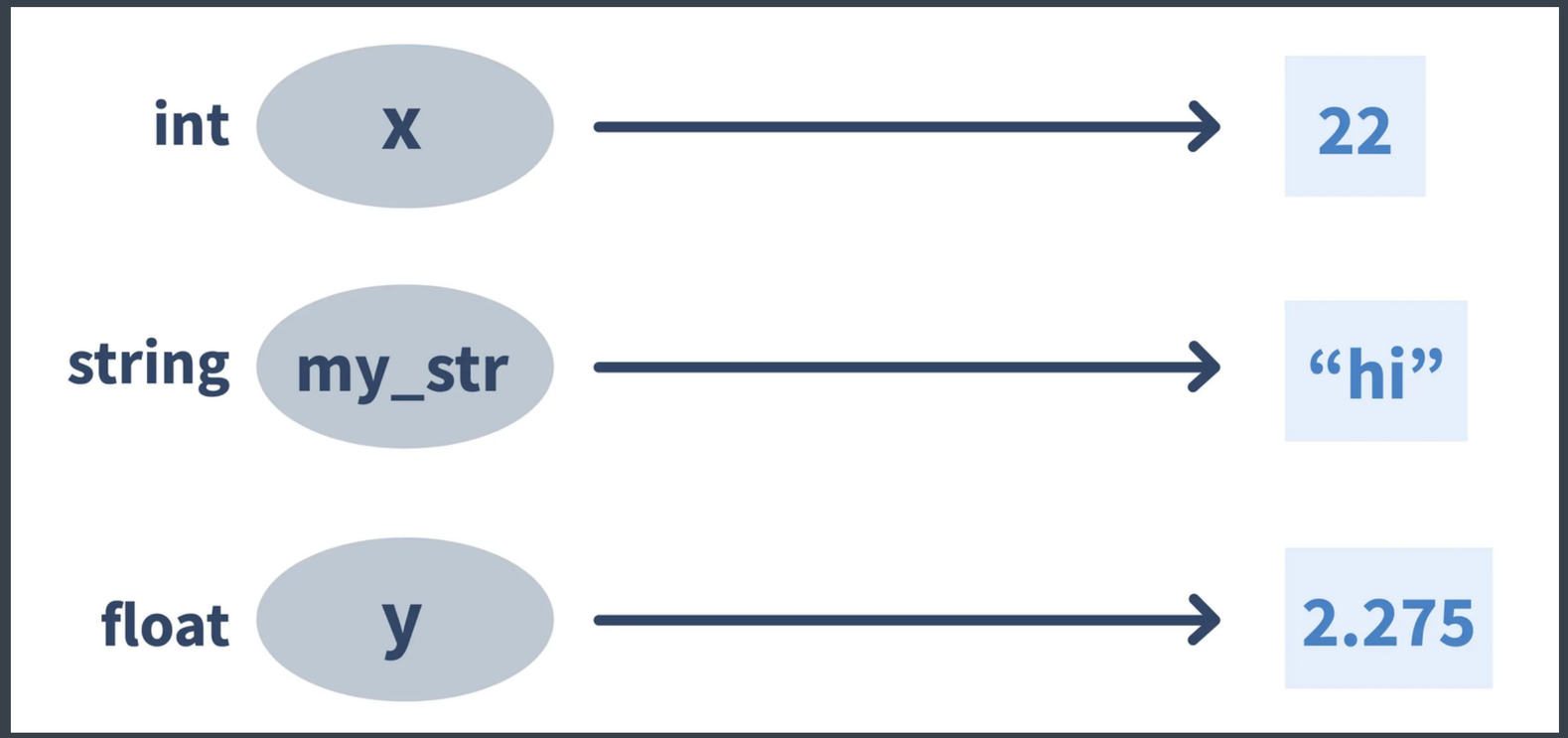
Variables

Variables

In programming, a variable is a container (storage area) to hold data.

```
int age;                // Integer Variable
float temperature;      // Floating-Point Variable
char initial;           // Character Variable
double salary;          // Double-Precision Floating-Point Variable
long studentID;         // Long Integer Variable
unsigned int flags;     // Unsigned Integer Variable
short count;            // Short Integer Variable
int scores[5];          // Array of Integers
int *ptr;               // Pointer Variable
typedef enum { false, true } bool;
bool isValid = true;    // Boolean Variable (using a typedef)
```

Variables



Rules

- Start with a Letter or Underscore
- Followed by Letters, Digits, or Underscores
- Avoid Starting with a Digit
- No Special Characters(!, @, $, etc.)
- Case-Sensitive (e.g: age, Age, and AGE)
- Avoid Using Keywords

Valid & Invalid Variables

- name
- age
- _count
- total_price
- is_student
- calculate_area
- student_info
- MAX_VALUE

- 3d_model (starts with a digit)
- @result (contains special character)
- for (a Python keyword)
- my-name (contains a hyphen, not allowed)

Variables vs Identifiers

Identifiers are names given to entities like variables, functions, etc., while variables are a specific type of identifier used to store and manipulate data in a program

Constants

Constants

A constant is a special type of variable whose value cannot be changed.

```
const int months_in_year = 12;
const double tax_rate = 0.075;
```

Literals

Literals

Literals are representations of fixed values in a program.
They can be numbers, characters, or strings, etc.

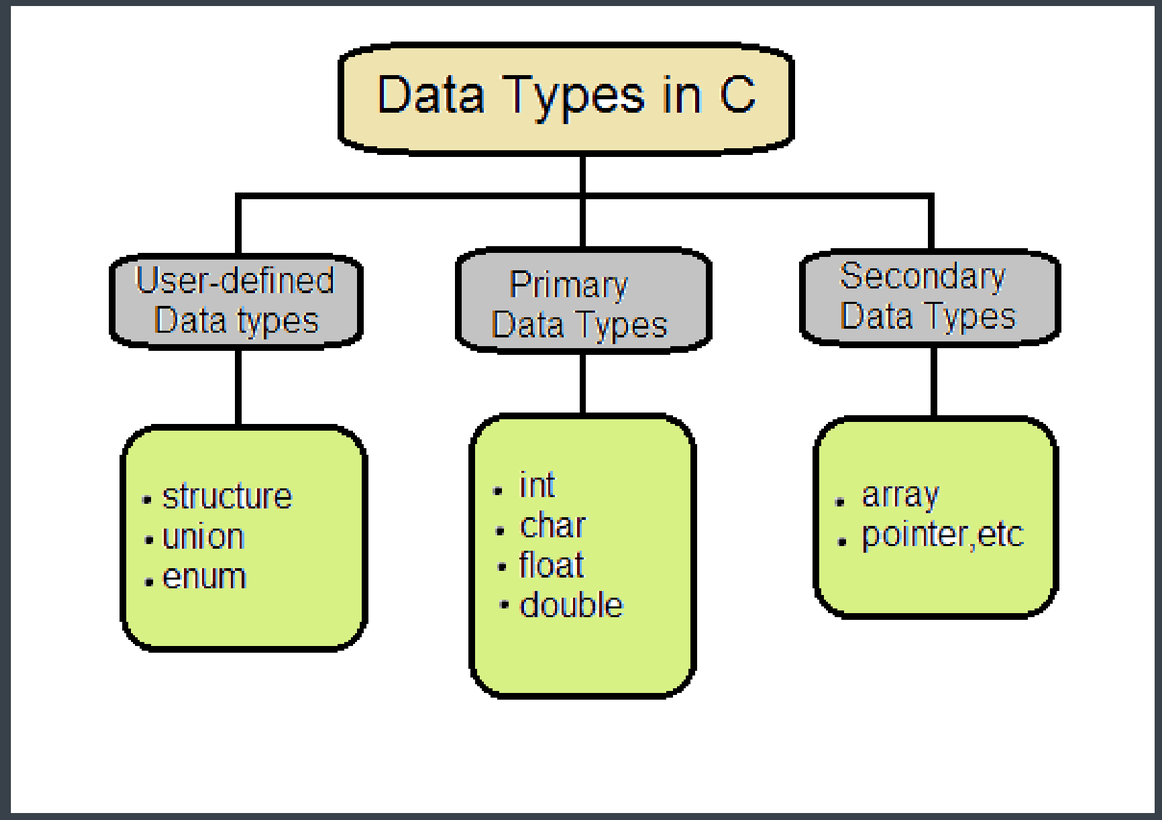
```c
int integerLiteral = 42;    // Integer Literal
float floatLiteral = 3.14; // Floating-Point Literal
char charLiteral = 'A';     // Character Literal
char *stringLiteral = "Hello, World!"; // String Literal
```

Types of Literals

1. Integer Literals
2. Floating-Point Literals
3. Character Literals
4. String Literals
5. Boolean Literals (C99 and later)
6. Pointer Literals
7. Wide Character and String Literals
8. Enumeration Constants

Data Types

What you will Learn ?



Primary data types

- int: Integer data type, used for whole numbers.
- float: Single-precision floating-point data type for real numbers.
- double: Double-precision floating-point data type for real numbers.
- char: Character data type, used for single characters

Secondary data types

- Array: A collection of elements of the same data type.
- Pointer: A variable that stores the memory address of another variable.

User defined data types

- enum: Used to define a set of named integer constants.
- Structure: A user-defined data type that groups variables of different data types.
- Union: Similar to a structure, but it shares memory space for its members.

Type conversions

Type Conversion

In programming, type conversion is the process of converting data of one type to another.
For example: converting int data to str.

Type Conversion

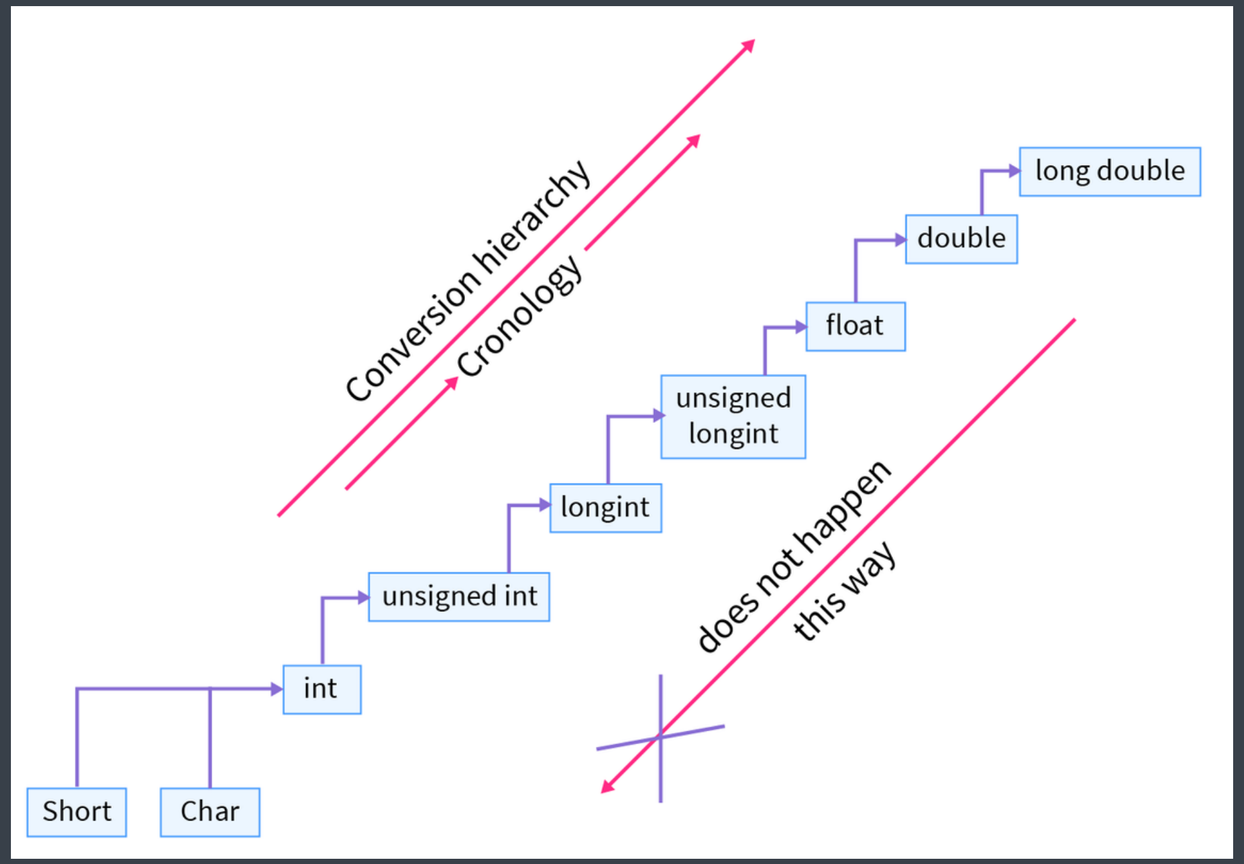
Implicit Type Conversion

Python automatically converts one data type to another.

Explicit Type Conversion

Users convert the data type of an object to required data type.

Conversion Hierarchy



Built-in type conversion functions

- atoi: Converts a string to an integer.
- atol: Converts a string to a long integer.
- atof: Converts a string to a double.
- itoa: Converts an integer to a string representation.

Implicit Type Conversion

```
int x = 5;
double y = 3.14;

double result = x + y;
```

Explicit Type Conversion

```
int a = 10;
double b = 3.1415;

int c = (int)b;
```

Introduction to ASCII Values

The "ASCII value" of a character is a number that represents that character in the computer's memory.

int() & char()

In C, you can work with ASCII values using character data types and the int data type.

```c
char ch = 'A';
int asciiValue = (int)ch;
printf("ASCII value of %c is %d\n", ch, asciiValue);
```

```c
int asciiValue = 65; // ASCII value for 'A'
char ch = (char)asciiValue;
printf("Character for ASCII value %d is %c\n", asciiValue, ch);
```

What you have Learnt ?

1. Datatypes
2. Type Conversion
3. ASCII Values

Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Bitwise Operators
- Conditional (Ternary) Operator
- Comma Operator
- Member and Pointer Operators
- Sizeof Operator
- Type Cast Operator

Arithmetic Operators

- + Addition: Adds two operands.
- - Subtraction: Subtracts the right operand from the left operand.
- * Multiplication: Multiplies two operands.
- / Division: Divides the left operand by the right operand (returns a float).
- % (Modulo): Computes the remainder of the division of the left operand by the right operand.

Arithmetic Operators

```
int a = 10, b = 4;
int sum = a + b;          // Addition
int difference = a - b; // Subtraction
int product = a * b;     // Multiplication
int quotient = a / b;    // Division
int remainder = a % b;   // Modulo (remainder)
```

Relational Operators

- == (Equal to): Compares if two values are equal.
- != (Not equal to): Compares if two values are not equal.
- < (Less than): Checks if the left operand is less than the right operand.
- > (Greater than): Checks if the left operand is greater than the right operand.

Relational Operators

- <= (Less than or equal to): Checks if the left operand is less than or equal to the right operand.
- >= (Greater than or equal to): Checks if the left operand is greater than or equal to the right operand.

Relational Operators

```c
int a = 5, b = 10;
if (a == b)
    printf("a is equal to b\n");
if (a != b)
    printf("a is not equal to b\n");
if (a < b)
    printf("a is less than b\n");
if (a >= b)
    printf("a is not greater than or equal to b\n");
```

Logical Operators

- and Logical AND: Returns True if both conditions are True.
- or Logical OR: Returns True if at least one condition is True.
- not Logical NOT: Returns True if the condition is False, and vice versa.

Logical Operators

```c
int a = 5, b = 10;
if (a > 0 && b < 15)
    printf("Both conditions are true\n");
if (a < 0 || b > 20)
    printf("At least one condition is true\n");
if (!(a == 10))
    printf("a is not equal to 10\n");
```

Assignment Operators

- = Assignment: Assigns the value on the right to the variable on the left.
- += Add and Assign: Adds the right operand to the variable on the left and assigns the result to the variable.
- -= Subtract and Assign: Subtracts the right operand from the variable on the left and assigns the result to the variable.

Assignment Operators

```c
a = 10;       // Assignment: a is now 10
printf("a = %d\n", a);

a += 2;       // Add and Assign: Increment a by 2 (a = a + 2)
printf("a = %d\n", a);

a -= 3;       // Subtract and Assign: Decrement a by 3 (a = a - 3)
printf("a = %d\n", a);
```

Assignment Operators

- *= (Multiply and Assign): Multiplies the variable on the left by the value on the right and assigns the result to the variable on the left.
- /= (Divide and Assign): Divides the variable on the left by the value on the right and assigns the result to the variable on the left.
- %= (Modulo and Assign): Computes the modulo of the variable on the left by the value on the right and assigns the result to the variable on the left.

Assignment Operators

```
a *= 4;        // Multiply and Assign: Multiply a by 4 (a = a * 4)
printf("a = %d\n", a);

a /= 2;        // Divide and Assign: Divide a by 2 (a = a / 2)
printf("a = %d\n", a);

a %= 3;        // Modulo and Assign: Compute the modulo of a by 3 (a = a
printf("a = %d\n", a);
```

Increment and Decrement Operators

++ (Increment):

- Increases the value of a variable by 1.
- Can be used as either a postfix operator (variable++) or a prefix operator (++variable).

-- (Decrement):

- Decreases the value of a variable by 1.
- Can be used as either a postfix operator (variable--) or a prefix operator (--variable).

Increment and Decrement Operators

```
int b = a++;  // Postfix Increment
int c = a--;  // Postfix Decrement
int d = ++a;  // Prefix Increment
int e = --a;  // Prefix Decrement
```

Bitwise Operators

- Bitwise AND (&): Sets each bit to 1 if both bits are 1.
- Bitwise OR (|): Sets each bit to 1 if at least one of the bits is 1.
- Bitwise XOR (^): Sets each bit to 1 if only one of the bits is 1.
- Bitwise NOT (~): Inverts the bits, changing 1 to 0 and 0 to 1.

Bitwise Operators

- Bitwise Left Shift (<<): Shifts the bits to the left by a specified number of positions.
- Bitwise Right Shift (>>): Shifts the bits to the right by a specified number of positions

```
int result_and = a & b;  // Bitwise AND
int result_or = a | b;   // Bitwise OR
int result_xor = a ^ b;  // Bitwise XOR
int result_not_a = ~a;   // Bitwise NOT (for 'a')
int result_shift_left = a << 2;  // Left Shift by 2 bits
int result_shift_right = a >> 1; // Right Shift by 1 bit
```

Comments

Comments

In computer programming, comments are hints that we use to make our code more understandable.

Comments are completely ignored by the interpreter.

Types of Comments

Single-line comments

These comments are written on a single line and are preceded by //. Anything following // on that line is treated as a comment.

Multi-line comments

These comments can span multiple lines and are enclosed by /* at the beginning and */ at the end.

Types of Comments

```
// This is a single-line comment
int a = 10; // This comment is after code
```

```
 /* This is a
    multi-line comment */
 int b = 20;
```