1 Types Of Parameter Passing

1.1 call-by-value

the parameter refer to the new copied values in this new memory . at the function conclusion this memory may simply be deallocated .

1.2 call-by-reference

in the call-by-reference evaluation strategy , when a function is called it recieves the memory locations of the arguments passed to it .

in this case , no extra memory need to be allocated .

2 Parameter Passing in Languages

2.1 C

predefined data type like : int , char , ... are passed with call-by-value in C .

call-by-reference simulated in C by employing the concept of a pointer ,

This is done with the operator * .

if a is a pointer , then $*a$ is the actual memory location pointed to .

2.2 C++

C++ allows one to declare reference types, for example `int& b = a;` defines an integer reference b, which simply acts as an alias for the integer a .

Additionally, C++ offers a *const* keyword if one would like the performance benefits of call-by-reference but still wants assurance that the arguments are never modified as in call-by-value .

2.3 Java

Java is almost like C++ .

but does not support the pointer manipulation and reference types offered in C++ .

in Java , variables are passed by value .

Java , passes Objects by reference .

2.4 Fortran

the default method of parameter passing is call-by-reference .

2.5 ALGOL

supports both call-by-value and call-by-reference .

3 Parameter Passing Modes in C

- Call by value parameter passing only
- Objects can be modified in a function by passing pointers to the object to the function
- Arrays and pointers are exchangeble in C: an array is automatically passed as a pointer to the array

4 Parameter Passing Modes in Fortran

- Call by reference parameter passing only

5 Parameter Passing Modes in Pascal

- Call by value and call by reference parameter passing
- Call by value is similar to C
- Call by reference is indicated by var parameters

```
1 swap(var a:integer, var b:integer)
2 var t;
3 begin
4 t := a; a := b; b := t
5 end
```

6 Parameter Passing Modes in C++

- Call by value and call by reference parameter passing
- Call by value is similar to C
- Call by reference is indicated by using & for formal parameters

```
swap(int &a, int &b)
{
    int t = a;
    a = b;
    b = t;
}
```

- Large objects should be passed by reference instead of by value to increase efficiency
- Arrays are automatically passed by reference (like in C)
- To avoid objects to be modified when passed by reference, const parameters can be used

```
store_record_in_file(const huge_record &r)
{
    ...
}
```

7 Parameter Passing Modes in Java

- Call by value and call by reference/sharing parameter passing
- Java adopts both value and reference models of variables
  - Variables of built-in types are passed by value
  - Class instances are passed by sharing

8 Parameter Passing Modes in Ada

- Call by value, call by result, and call by value/result parameter passing
- Indicated by Ada's in(by value), out(by result), and in out(by value/result) modes for formal parameters

```
procedure shift(a:out integer, b:in out integer,
c:in integer) is
begin
   a := b; b := c;
end shift;
```

- in mode parameters can be read but not written in the subroutine
- out mode parameters can be written but not read in the subroutine
- in out mode parameters can be read and written in the subroutine

9 Parameter Passing Modes in Python

Python uses a mechanism, which is known as "Call-by-Object", sometimes also called "Call by Object Reference" or "Call by Sharing".

10 C++ Variables

10.1 Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

```
type variable = value;
```

Listing 1: C++ example

11 Python Variables

11.1 Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 5
y = "John"
print(x)
print(y)
```

Listing 2: Python example

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Listing 3: Python example

11.2 Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Listing 4: Python example

12 Ada Variables

12.1 Declaring a Variable

To declare a variable, use the following formula:

```
VariableName : DataType;
```

Listing 5: Ada example

declare various variables :

```
VariableName1, VariableName2 : DataType1;
```

Listing 6: Ada example

12.2 Initializing a Variable

To initialize a variable :

```
VariableName : DataType := Value;
```

Listing 7: Ada example

assigning a value after declaring it :

```
VariableName : DataType
begin
    VariableName:= Value;
end
```

Listing 8: Ada example

13 Pascal

13.1 Variable Declaration in Pascal

All variable declarations are followed by the var keyword. A declaration specifies a list of variables, followed by a colon (:) and the type. Syntax of variable declaration is -

```
var
variable_list : type;
```

Listing 9: Pascal example

valid Pascal data type including : character, integer, real, boolean, any user-defined data type .

Some valid variable declarations are shown here -

```
var
age, weekdays : integer;
taxrate, net_income: real;
choice, isready: boolean;
initials, grade: char;
name, surname : string;
```

Listing 10: Pascal example

13.2 Variable Initialization in Pascal

Variables are assigned a value with a colon and the equal sign, followed by a constant expression. The general form of assigning a value is -

```
variable_name := value;
```

Listing 11: Pascal example

Variables can be initialized (assigned an initial value) in their declaration :

```
var
variable_name : type = value;
```

Listing 12: Pascal example

Some examples are -

```
age: integer = 15;
taxrate: real = 0.5;
grade: char = 'A';
name: string = 'John Smith';
```

Listing 13: Pascal example

14 Lisp

14.1 Global Variables

Global variables are generally declared using the defvar construct.

```
(defvar x 234)
(write x)
```

Listing 14: Lisp example

you can directly specify a value for a symbol with the setq construct :

```
->(setq x 10)
```

Listing 15: Lisp example

14.2 Local Variables

There are two other constructs - let and prog for creating local variables :

14.2.1 Example

```
(let ((y 1)
      (z y))
      (list y z))
```

Listing 16: Lisp example

Result → (1 2)

14.2.2 Example

```
(let ((str "Hello, world!"))
      (string-upcase str))
```

Listing 17: Lisp example

Result → "HELLO, WORLD!"

15 Definition of Compiler

- A compiler is a computer program that translates computer code written in one programming language (the source language) into another language (the target language).
- The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.

16 What is Decompiler?

A program that translates from a low-level language to a higher level one is a decompiler.

17 Source-to-Source compiler or transpiler

A program that translates between high-level languages is usually called a source-to-source compiler or transpiler

18 Alphabet, String, Language

Alphabet	finite set of symbols
String	finite sequence of sumbols
Language	set of strings on an alphabet

19 What is Parser?

Syntax analysis (also known as parsing) involves parsing the token sequence to identify the syntactic structure of the program. This phase typically builds a parse tree, which replaces the linear sequence of tokens with a tree structure built according to the rules of a formal grammar which define the language's syntax. The parse tree is often analyzed, augmented, and transformed by later phases in the compiler.

20 Types Of Compilers

20.1 native or hosted compiler

A native or hosted compiler is one whose output is intended to directly run on the same type of computer and operating system that the compiler itself runs on.

20.2 Cross compiler

The output of a cross compiler is designed to run on a different platform.

20.3 Source-to-Source compiler

Source-to-source compilers are a type of compiler that takes a high-level language as its input and outputs a high-level language.

20.4 Bytecode compiler

Bytecode compilers that compile to assembly language

20.5 Just-in-time compilers

Just-in-time compilers (JIT compiler) defer compilation until runtime.

20.6 Decompiler

A program that translates from a low-level language to a higher level one is a decompiler.