Chapter 3

Primitive Types

Almost all programming languages have **Primitive Types**

Java Integers

- Int
- Byte
- Short
- Long

Floating Points

- Float
- Double

IEEE Floating-Point

Language Support Complex

- C99
- Python
- Fortan

Each part have 2 parts: real like 7 and imaginary like 3j to be like (7+3j)

Note: Python calls complex Literal

Decimal

- Essential for COBOL
- C# have decimal

Boolean

Very simple.
It has two values: either "**True**" or "**False**".
It can be implemented as" **Bits**" but often as "**Bytes**".

Advantages: readability

Array

It's an aggregate of homogeneous data elements.

Array Design Issues

- What types are legal for subscripts (indexing)?
- Are subscribing expressions in element reference type checked?
- What are subscribing ranges bound?
- When does allocation take place?
- What's maximum number of subscribing?

Chapter 3

- Can an Array object be initiated?
- Are any kind of Slices supported?

Array Indexing

It's mapping from indices to elements.

Note: PL/I and Fortan and Ada use parentheses.

Array indexing types

C, Fortan, java use: Int
Ada use: Int and enum (Boolean and char)

Index range checking

Specify range checking: Java, ML, C#
Don't Specify range checking: C, C++, Perl
It does specify in the default state but you can turn specifying off: Ada

Subscript Binding and Array Categories

1. **Static**: subscripts are statically bound.

Advantages: efficiency.

2. **Fixed stack-Dynamic**: it's statically bound, but the allocation is done at the declaration time

Advantages: space efficiency.

3. **Stack-dynamic**: subscript ranges are dynamically bound, and the storage allocation is dynamic.

Advantages: flexibility.

4. **Fixed heap-dynamic**: storage binding is dynamic but fixed after allocation.

5. **Heap dynamic**: binding of subscript ranges and storage allocation is dynamic.

Advantages: Flexibility.

Chapter 3

C, C++

It's **static** when we use the modifier "**Static**".

It's **fixed stack-dynamic** when we don't use "**Static**".

They also support **Fixed heap-dynamic** arrays.

C#

It supports second array class which "ArrayList".

Perl, JavaScript, Python, Ruby

They support **heap-dynamic** arrays.

Array initialization

– C, C++, Java, C# example

```
int list [] = {4, 5, 7, 83}
```

– Character strings in C and C++

```
char name [] = "freddie";
```

– Arrays of strings in C and C++

```
char *names [] = {"Bob", "Jake", "Joe"];
```

– Java initialization of String objects

```
String[] names = {"Bob", "Jake", "Joe"};
```

Heterogenous Arrays

Elements don't need to have same data type

Supported By: **Python, Perl, JavaScript, Ruby**

Array initialization

C-based languages

```
– int list [] = {1, 3, 5, 7}
– char *names [] = {"Mike", "Fred","Mary Lou"};
```

Ada

```
– List : array (1..5) of Integer :=
    (1 => 17, 3 => 34, others => 0);
```

Python

– List comprehensions

```
list = [x ** 2 for x in range(12) if x % 3 == 0]
puts [0, 9, 36, 81] in list
```

Type Checking

It ensures that operands of an operator are compatible type.

Compatible Type: either legal for the operand, or allowed under language rules.

Type error: an application of operand of inappropriate type.

Chapter 3

if all bindings are static then, all type check is static.

If all bindings are dynamic then, all type check is dynamic.

It's **Strongly typed language** if type errors are always detected.

Advantages: allows the detection of the misuses of variables that result in type errors.

Strong Typing

Examples

Fortan is not: parameters, EQUIVALENCE.

C and **C++** are not: parameter type checking can be avoided.

Ada, **java**, and **C#**: almost (UNCHECKED CONVERSION is loophole)

Names

Design issues:-

- Case sensitivity
- Reserved words
- Length
- Special characters
- Special words

Variables

Abstraction of memory cell.

Variable attributes:-

- Name
- Type
- Address
- Lifespan
- Value
- Scope

Binding

It’s an association.

Such as between attribute and entry.

Or between symbol and operation.

Time binding: the time binding takes a place.

Chapter 3

Static binding: it occurs before run time and remains unchanged throughout program execution.

Categories of Variables by Lifetimes

Static bound to memory cells before execution begins and remains bound to the same memory cell throughout execution

Advantage: efficiency.

Disadvantages: Lack of flexibility.

Scope

It's the range of statement over which it's visible

Static scope

- Based on program text.
- Connect a name reference to a variable.

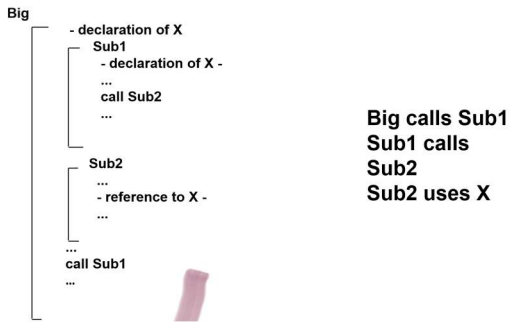
Block

Method of creating static scopes inside program units.

– Example in C:

```
void sub() {
  int count;
  while (...) {
      int count;
    count++;
    ...
  }
  …
}
```

Scope Example



- **Static scoping**

  Reference to X is to Big's X

- **Dynamic scoping**

  Reference to X is to sub's X

Advantages: convince.

Chapter 3

Disadvantages:

- While all subprograms are executing its variables are visible to all subprograms it calls.
- Static type checking is impossible.
- Poor readability.

Scope and lifetime

Scope and lifetime are strongly related.

Scope of variables: The scope means the lifetime of that variable. It means the variable can only be accessed or visible within its scope

```
#include <stdio.h>
void checkEvenOrNot(int num) // function to check even or not
{
   if (num % 2 == 0)
      goto even;  // jump to even
   else
      goto odd;  // jump to odd
even:
   printf("%d is even", num);
   return; // return if even
odd:
   printf("%d is odd", num);
}
int main()
{
   int num = 26;
   checkEvenOrNot(num);
   return 0;
```


Local Vs. Global in Python

```
v1 = "Hey, I am Global Variable."   #globalvariable
def func1():
   v2="Hey, I am Local Variable." #localvariable
   print(v2)
func1()    #calling func1

def func2():
   print(v1)
func2()    #callin func2
```

Output:

Hey, I am a Local Variable
Hey, I am Global Variable

In the above program, we have taken one global variable v1 and one local variable v2. Since v1 is global, it can be easily accessed in any function, and v2 is local; it is used only within its declared function. But if we try to use v1 in func1, it will give an error.

Chapter 3

Scope in Java

Local Vs. Global Variable in Java

In Java, there is no concept of global variables; since Java is an Object-oriented programming language, everything is a part of the Class. But if we want to make a variable globally accessible, we can make it static by using a **static** Keyword.

```
class Demo
{
        // static variable
        static int a = 10;
        // non-static or local variable
        int b = 20;
}


public class Main
 {
        public static void main(String[] args)
        {
                Demo obj = new Demo();
               // accessing the non-static variable
                System.out.println("Value of non-
static variable is: " + (obj.b));
               // accessing the static variable

        System.out.println("Value of static variable is:" + (Demo.a));
        }
}
```


Output:

```
Value of non-static variable is: 20
Value of static variable is:10
```

In the above program, we have used one local variable or non-static variable and one static variable. The local variable can be accessed by using the object of the Demo class, whereas the static variable can be accessed using the name of the class.