Unit II
Structuring the Data, Computations and Program

Elementary Data Types : Primitive data Types, Character String types, User Defined Ordinal Types, Array types, Associative Arrays, Record Types, Union Types, Pointer and reference Type.
Expression and Assignment Statements: Arithmetic expression, Overloaded Operators, Type conversions, Relational and Boolean Expressions, Short Circuit Evaluation, Assignment Statements, Mixed mode Assignment. **Statement level Control Statements:** Selection Statements, Iterative Statements, Unconditional Branching. **Subprograms:** Fundamentals of Sub Programs, Design Issues for Subprograms, Local referencing Environments, Parameter passing methods.
Abstract Data Types and Encapsulation Construct: Design issues for Abstraction, Parameterized Abstract Data types, Encapsulation Constructs, Naming Encapsulations.

Elementary Data Types

1. Introduction

A data type defines a collection of data values and a set of predefined operations on those values.

Descriptor:

A descriptor is the collection of the attributes of a variable.

Object:

The word object is often associated with the value of a variable and the space it occupies.

Types of Data Types:

i. Primitive Data Types

ii. Structured Data Types

Uses of the type system of a programming language:

i. Error detection

ii. Program modularization

iii. Documentation

Unit II
Structuring the Data, Computations and Program

2. Primitive data Types

Data types that are not defined in terms of other types are called primitive data types.

Also called as Built-in data types.

Examples:

a. Numeric Data Types (Integer, Floating-Point, Complex, Decimal etc.)

b. Boolean Data Types

C. Character Data Types

a. Numeric Types

i. Integer

- For example, Java includes four signed integer sizes: byte, short, int, and long.
- Some languages, for example, C++ and C#, include unsigned integer types, which are simply types for integer values without signs.
- A signed integer value is represented in a computer by a string of bits, with one of the bits (typically the leftmost) representing the sign.
- Most computers now use a notation called twos complement to store negative integers.
- In twos-complement notation, the representation of a negative integer is formed by taking the logical complement of the positive version of the number and adding one.
- Ones-complement notation is still used by some computers. In ones-complement notation, the negative of an integer is stored as the logical complement of its absolute value.

ii. Floating-Point

Floating-point data types model real numbers. Most languages include two floating-point types, often called float and double.

The collection of values that can be represented by a floating-point type is defined in terms of precision and range.

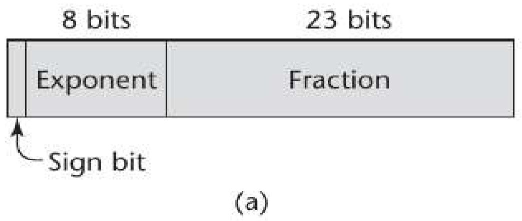
Precision is the accuracy of the fractional part of a value, measured as the number of bits.

Range is a combination of the range of fractions and, more important, the range of exponents.

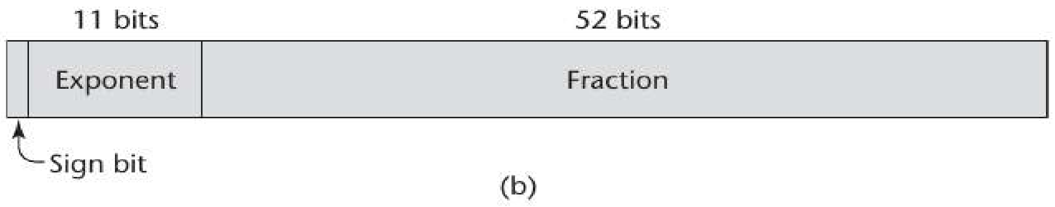
IEEE floating-point formats:

(a) Single precision,

(b) Double precision



(a)



(b)

iii. Complex

Some programming languages support a complex data type. For example, FORTRAN and Python.

In Python, the imaginary part of a complex literal is specified by following it with a j or J —for example, (7 + 3j). Languages that support a complex type include operations for arithmetic on complex values.

iv. Decimal

Larger computers that are designed to support business systems applications have hardware support for decimal data types.

Decimal data types store a fixed number of decimal digits, with the decimal point at a fixed position in the value.

COBOL, C# and F# languages have decimal data types.

Decimal types are stored very much like character strings, using binary codes for the decimal digits. These representations are called binary coded decimal (BCD).

b. Boolean Types

Their range of values has only two elements: one for true and one for false.

They were introduced in ALGOL 60 and have been included in most general-purpose languages designed since 1960.

c. Character Types

Character data are stored in computers as numeric coding. Traditionally, the most commonly used coding was the **8-bit code ASCII (American Standard Code for Information Interchange)**, which uses the values 0 to 127 to code 128 different characters.

Because of the globalization of business and the need for computers to communicate with other computers around the world, the ASCII character set became inadequate. In response, in 1991, the Unicode Consortium published the UCS-2 standard, a 16-bit character set. This character code is often called **Unicode**.

Unicode includes the characters from most of the world's natural languages.

For example, Unicode includes the Cyrillic alphabet, as used in Serbia, and the Thai digits.

After 1991, the Unicode Consortium, in cooperation with the International Standards Organization (ISO), developed a 4-byte character code named UCS-4, or UTF-32, which is described in the ISO/IEC 10646 Standard, published in 2000.

The first 128 characters of Unicode are identical to those of ASCII.

Java was the first widely used language to use the Unicode character set. Since then, it has found its way into JavaScript, Python, Perl, C#, and F#.

Unit II
Structuring the Data, Computations and Program

3. Character String types

A character string type is one in which the values consist of sequences of characters.

Strings and Their Operations

The most common string operations are assignment, concatenation, substring reference, comparison, and pattern matching.

Character string types could be supported directly in hardware; but in most cases, software is used to implement string storage, retrieval, and manipulation.

Implementation

i. C and C++

C and C++ use char arrays to store character strings.

e.g.

char mystring[]="COMP";

OR

char mystring[5]={'C','O','M','P','S'};

ii. Java

In Java, strings are supported by the String class, whose values are constant strings, and the StringBuffer class, whose values are changeable and are more like arrays of single characters.

e.g.

String s="COMPS";

String s1=new String ("COMPS");

StringBuffer s2=new StringBuffer("COMPS");

iii. C# and Ruby

C# and Ruby include string classes that are similar to those of Java.

iv. Python

Python includes strings as a primitive type i.e. **str**.

For character and substring references, they act very much like arrays of characters. However, Python strings are immutable, similar to the String class objects of Java.

v. F#

In F#, strings are a class.

vi. ML (Meta Language)

In ML, string is a primitive immutable type.

Perl, JavaScript, Ruby, and PHP include built-in pattern-matching operations. They use Regular Expressions for pattern matching.

Pattern-matching capabilities using regular expressions are included in the class libraries of C++, Java, Python, C#, and F#.

String Length Options

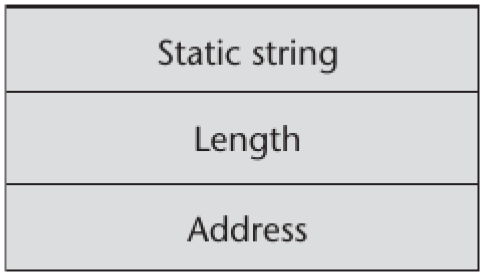
The length can be static and set when the string is created. Such a string is called a static length string. E.g. Java, Python.

Limited dynamic length strings are those strings which are allowed to have varying length up to a declared and fixed maximum set by the variable's definition. E.g. C, C++.

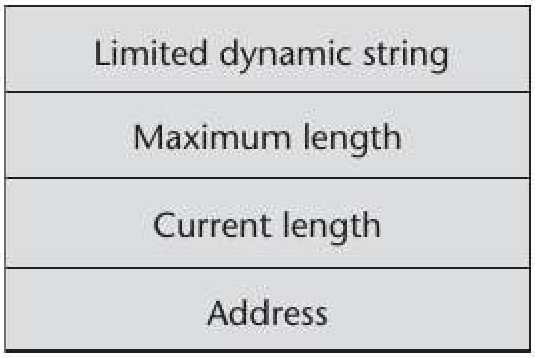
Dynamic length strings are those strings which have varying length with no maximum. E.g. JavaScript, Perl, and the standard C++ library.

Ada 95+ supports all three string length options.

Descriptor for Static Length Strings



Descriptor for Limited Dynamic Length Strings



Dynamic length strings require a simpler run-time descriptor because only the current length needs to be stored.

4. User Defined Ordinal Types

- An ordinal type is one in which the range of possible values can be easily associated with the set of positive integers.
- In Java, for example, the primitive ordinal types are integer, char, and Boolean.
- There are two user-defined ordinal types that have been supported by programming languages: enumeration and sub range.

i. Enumeration Types

An enumeration type is one in which all of the possible values, which are named constants, are provided, or enumerated, in the definition.

Values in enumeration type are called as **enumeration constants.**

Example:

C or C++ Language

enum colors {red, blue, green, yellow, black};

colors myColor = blue, yourColor = red;

C and Pascal were the first widely used languages to include an enumeration data type. C++ includes C's enumeration types.

C++ enumeration constants can appear in only one enumeration type in the same referencing environment.

In Ada, enumeration literals are allowed to appear in more than one declaration in the same referencing environment. These are called **overloaded literals**.

Enumeration types can provide advantages in both readability and reliability.

Examples:

C or C++ Language

enum colors {red, blue, green, yellow, black};

colors myColor = blue, yourColor = red;

Java

All enumeration types in Java are implicitly subclasses of the predefined class Enum.

e.g.

```
class MyEnum
{
enum colors { red, blue, green, yellow, black }
public static void main(String args[])
  {
        colors myColor = colors.blue, yourColor = colors.red;
        System.out.println("My Color is "+myColor.ordinal());
        System.out.println("Your Color is "+yourColor.ordinal());
```

```
    }
}
```

C# Language

enum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

These values are implicitly assigned the integer values 0, 1, 2, 3....

ML (Meta Language)

datatype weekdays = Monday | Tuesday | Wednesday | Thursday | Friday

F#

type weekdays = | Monday | Tuesday | Wednesday | Thursday | Friday

Languages Perl, JavaScript, PHP, Python,

Ruby, and Lua do not allow enumeration data type.

ii. Sub range Types

- A sub range type is a contiguous subsequence of an ordinal type.
- For example, 12.14 is a sub range of integer type.
- Sub range types were introduced by Pascal and are included in Ada.
- Sub range types enhance readability by making it clear to readers that variables of subtypes can store only certain ranges of values.
- Reliability is increased with sub range types, because assigning a value to a sub range variable that is outside the specified range is detected as an error

Example:

Ada Language

subranges are included in the category of types called subtypes.

type Days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

subtype Weekdays is Days range Mon..Fri;

subtype Index is Integer range 1..100;

All of the <u>operations</u> defined for the <u>parent type</u> are also defined for the <u>subtype</u>, except assignment of values outside the specified range

Array types

I. Definition:

An array is a <u>homogeneous aggregate of data elements</u> in which an individual element is identified by its position in the aggregate, relative to the first element.

II. Arrays and Indices

Specific elements of an array are referenced by means of a two-level syntactic mechanism, where the <u>first part is the aggregate name</u>, and the <u>second part is a possibly dynamic selector</u> consisting of one or more items known as **<u>subscripts or indices</u>**.

The selection operation can be thought of as a mapping from the array name and the set of subscript values to an element in the aggregate. Indeed, arrays are sometimes called <u>finite mappings</u>.

Symbolically, this mapping can be shown as

array_name(subscript_value_list) → element

Generally, in most of the programming languages subscripts are bounded by **brackets** like **[]**. But, in some programming languages, subscripts are bounded by **parentheses** like ().

For Example, in **Ada** language

Sum := Sum + B(I);

This results in <u>reduced readability</u> as parentheses are used for both <u>subprogram parameters and array subscripts</u>.

Most languages other than **Fortran and Ada** use brackets to delimit their array indices.

The **type of the subscripts** is often a **subrange of integers**, but **Ada** allows any **ordinal type** to be used as subscripts, such as Boolean, character, and enumeration. For example, in Ada one could have the following:

type Week_Day_Type is (Monday, Tuesday, Wednesday,

Thursday, Friday);

type Sales is array (Week_Day_Type) of Float;

Range errors in subscripts are common in programs, so requiring range checking is an important factor in the reliability of languages.

Many contemporary languages do not specify range checking of subscripts, but **Java, ML, and C#** do.

By default, **Ada** checks the range of all subscripts, but this feature can be disabled by the programmer.

Subscripting in **Perl** is a bit unusual in that although the names of all arrays begin with at signs (@), because array elements are always scalars and the names of scalars always begin with dollar signs ($), references to array elements use dollar signs rather than at signs in their names. For example, for the array **@list**, the second element is referenced with **$list [1]**.

One can reference an array element in Perl with a **negative subscript**, in which case the subscript value is an offset from the end of the array.

For example, if the array @list has five elements with the subscripts 0.4, **$list [-2]** references the element with the subscript 3.

A reference to a nonexistent element in Perl yields undef, but no error is reported

III. Subscript Bindings and Array Categories

In some languages, the lower bound of the subscript range is implicit. For example, in the C-based languages, the lower bound of all subscript ranges is fixed at 0; in FORTRAN 95+ it defaults to 1 but can be set to any integer literal.

In some other languages, the lower bounds of the subscript ranges must be specified by the programmer.

There are **five categories of arrays**, based on the binding to subscript ranges, the binding to storage, and from where the storage is allocated:

i. Static Array

A static array is one in which the subscript ranges are statically bound and storage allocation is static (done before run time).

E.g. Arrays declared in C and C++ functions that include the static modifier are static.

The advantage of static arrays is efficiency: No dynamic allocation or de-allocation is required.

The disadvantage is that the storage for the array is fixed for the entire execution time of the program.

ii. Fixed stack-dynamic array

- A fixed stack-dynamic array is one in which the subscript ranges are statically bound, but the allocation is done at declaration elaboration time during execution.
- E.g. Arrays that are declared in C and C++ functions (without the static specifier) are examples of fixed stack-dynamic arrays.\
- The advantage of fixed stack-dynamic arrays over static arrays is space efficiency.
- The disadvantage is the required allocation and de-allocation time.

iii. stack-dynamic array

A stack-dynamic array is one in which both the subscript ranges and the storage allocation are dynamically bound at elaboration time.

e.g.

Ada arrays can be stack dynamic, as in the following:

```
 Get(List_Len);
declare
List : array (1..List_Len) of Integer;
begin
. . .
end;
```

The advantage of stack-dynamic arrays over static and fixed stack-dynamic arrays is flexibility. The size of an array need not be known until the array is about to be used.

iv. Fixed heap-dynamic array

A fixed heap-dynamic array is similar to a fixed stack-dynamic array, in that the subscript ranges and the storage binding are both fixed after storage is allocated.

e.g.

- C and C++ also provide fixed heap-dynamic arrays.
- The standard C library functions malloc and free, which are general heap allocation and de-allocation operations, respectively, can be used for C arrays.
- C++ uses the operators new and delete to manage heap storage.
- In Java, all non-generic arrays are fixed heap-dynamic. C# also provides the same kind of arrays.
- The advantage of fixed heap-dynamic arrays is flexibility i.e. the array's size always fits the problem.
- The disadvantage is allocation time from the heap, which is longer than allocation time from the stack.

v. heap-dynamic array

A heap-dynamic array is one in which the binding of subscript ranges and storage allocation is dynamic and can change any number of times during the array's lifetime.

e.g.

- C# also provides generic heap-dynamic arrays, which are objects of the List class. These array objects are created without any elements, as in

List<String> stringList = new List<String> ();

- Elements are added to this object with the Add method, as in

stringList.Add("Michael");

- Java includes a generic class similar to C#'s List, named ArrayList . It is different from C#'s List in that subscripting is not supported get and set methods must be used to access the elements.
- The advantage of heap-dynamic arrays over the others is flexibility: Arrays can grow and shrink during program execution as the need for space changes.
- The disadvantage is that allocation and de-allocation take longer and may happen many times during execution of the program.

IV. Array Initialization:

i. C / C++

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

char name [] = "freddie";

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

ii. Java

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

String[] names = ["Bob", "Jake", "Darcie"];
```

iii. Ada

```
List : array (1..5) of Integer := (1, 3, 5, 7, 9);

Bunch : array (1..5) of Integer := (1 => 17, 3 => 34,

others => 0);
```

V. Array Operations

The most common array operations are assignment, catenation, comparison for equality and inequality, and slices.

The C-based languages do not provide any array operations, except through the methods of Java, C++, and C#.

Perl supports array assignments but does not support comparisons.

Python's arrays are called lists.

Python provides array assignment.

Python also has operations for array catenation (+) and element membership (in).

Comparison is done by (==) and (is).

VI. Rectangular and Jagged Arrays

A rectangular array is a multidimensional array in which all of the rows have the same number of elements and all of the columns have the same number of elements.

E.g. int a [3][4]

A jagged array is one in which the lengths of the rows need not be the same. For example, a jagged matrix may consist of three rows, one with 5 elements, one with 7 elements, and one with 12 elements. This also applies to the columns and higher dimensions.

e.g.

```
arr[][] = { {0, 1, 2},
            {6, 4},
            {1, 7, 6, 8, 9},
            {5}
          };
```

FORTRAN, Ada, C#, and F# support rectangular arrays.

C# and F# also support jagged arrays.

Unit II
Structuring the Data, Computations and Program

VII. Slices

Consider the following Python declarations:

```
vector = [2, 4, 6, 8, 10, 12, 14, 16]

mat = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]

 Slices:

vector[3:6] returns [8, 10, 12]

vector[0:7:2] returns [2, 6, 10, 14]

mat[0:2] returns [[1, 2, 3], [4, 5, 6]]
```

VIII. Implementation of Array Types:

Address of list[k] is calculated as follows:

address (list[k]) = address(list[0]) + k * element_size

Generalization:

address (list[k]) = address(list [lower_bound]) +((k - lower_bound) * element_size)

Row major order & Column major order

For example, if the matrix had the values

```
3   4   7

6   2   5

1   3   8
```

It would be stored in row major order as

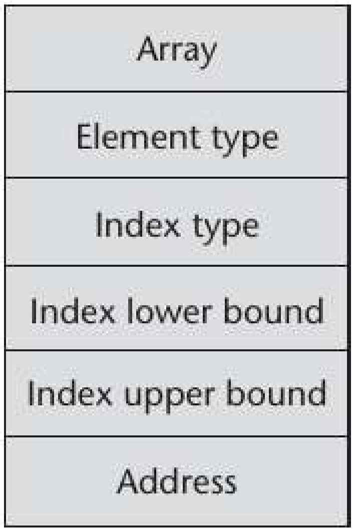
3, 4, 7, 6, 2, 5, 1, 3, 8

If the example matrix were stored in column major order, it would have the following order in memory:

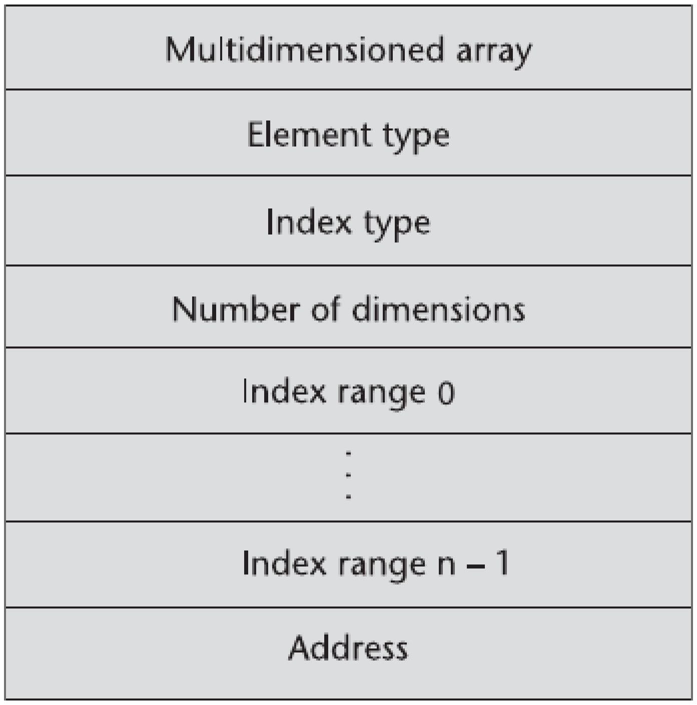
3, 6, 1, 4, 2, 3, 7, 5, 8

Column major order is used in **FORTRAN**, but other languages that have true multidimensional arrays use row major order.

Compile-time descriptor for single-dimensioned arrays:



A compile-time descriptor for a multidimensional array:



Unit II
Structuring the Data, Computations and Program

6. Associative Arrays

An associative array is an unordered collection of data elements that are indexed by an equal number of values called keys. **(Key-Value Pairs)**

Associative arrays are supported directly by **Perl, Python, Ruby, and Lua** and by the standard class libraries of **Java, C++, C#, and F#**.

e.g.

1. Java

In Java, Associative Arrays are implemented by Map interface and its classes HashMap, LinkedHashMap, TreeMap.

2. Perl

In Perl, associative arrays are called hashes.

```
%salaries = ("Gary" => 75000, "Perry" => 57000, "Mary" => 55750, "Cedric" =>
47850);
```

3. Python

Python's associative arrays, which are called dictionaries.

```
statecapital = {
 "Maharashtra": "Mumbai",
 "Karnataka": "Bangalore",
 "Telangana": "Hyderabad"
}
print(statecapital)
```

Unit II
Structuring the Data, Computations and Program

7. Record Types

Definition:

A record is an aggregate of heterogeneous data elements in which the individual elements are identified by names and accessed through offsets from the beginning of the structure.

Differences between Array and Record:

1. Arrays are used when all the data values have the same type and/or are processed in the same way.

Records are used when the collection of data values is heterogeneous and the different fields are not processed in the same way.

2. Array elements are accessed using Subscripts while Record elements are accessed using Field Names.

Implementation in Different Programming Languages:

i. Records in C, C++, C#

In C, C++, and C#, records are supported with the struct data type.

e.g.

Defining Linked List:

```
struct list
{
int info;
struct list *ptr;
};
```

Defining Linked List variable

struct list node;

Unit II
Structuring the Data, Computations and Program

Defining Binary Tree

```
struct btree
{
int info;
struct btree *lptr;
struct btree *rptr;
};
```

Defining Binary Tree variable

struct btree node;

ii. Records in COBOL

```
 01   EMPLOYEE-RECORD.
           02 EMPLOYEE-NAME.
                 05 FIRST PICTURE IS X(20).
                 05 MIDDLE PICTURE IS X(10).
                 05 LAST PICTURE IS X(20).
           02 HOURLY-RATE PICTURE IS 99V99.
```

iii. Records in Ada

```
type Employee_Name_Type is record
First : String (1..20);
Middle : String (1..10);
Last : String (1..20);
end record;
```

```
type Employee_Record_Type is record
Employee_Name: Employee_Name_Type;
Hourly_Rate: Float;
end record;
```

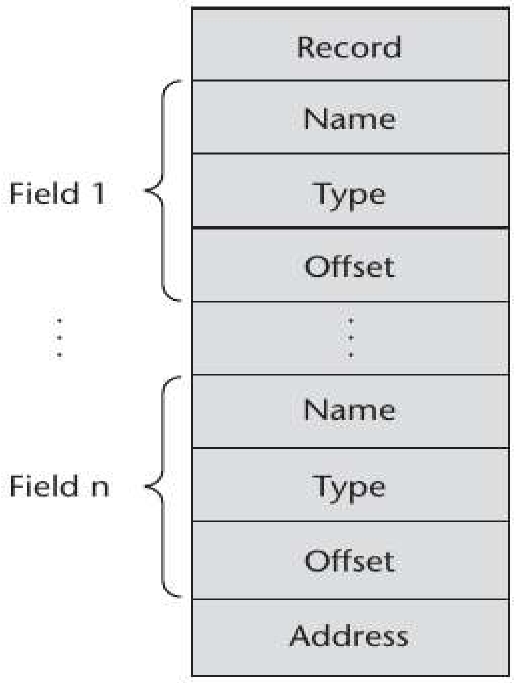
Employee_Record: Employee_Record_Type;

iv. Records in Java & C#

In Java and C#, records can be defined as data classes, with nested records defined as nested classes.

Data members of such classes serve as the record fields.

A compile-time descriptor for a record



8. Union Types

A union is a type whose variables may store different type values at different times during program execution.

Record vs Union

```
 struct sample
{
int x;
float y;
char z;
};


union sample
{
int x;
float y;
char z;
};
 union sample x;
```

Free Unions vs Discriminated Unions

C and C++ provide union constructs in which there is no language support for type checking. The unions in these languages are called free unions, because programmers are allowed complete freedom from type checking in their use.

```
 union sample
{
```

```
int a;
float b;
};
union sample myunion;
float x;
 myunion.a = 27;
x = myunion.b;
```

Type checking of unions requires that each union construct include a type indicator. Such an indicator is called a **tag**, or **discriminant**, and a union with a discriminant is called a **discriminated union**.

The first language to provide discriminated unions was ALGOL 68. They are now supported by Ada, ML, Haskell, and F#.

Unions in Ada

e.g.

```
type Shape is (Circle, Triangle, Rectangle);
type Colors is (Red, Green, Blue);


type Figure (Form : Shape) is
      record
            Filled : Boolean;
            Color : Colors;
            case Form is
                  when Circle =>
                        Diameter : Float;
```

```
                when Triangle =>
                        Left_Side : Integer;
                        Right_Side : Integer;
                        Angle : Float;
                when Rectangle =>
                        Side_1 : Integer;
                        Side_2 : Integer;
            end case;
        end record;
```

Figure_1 : Figure;

Figure_2 : Figure(Form => Triangle);

Here,

Figure_1 is unconstrained variant record.

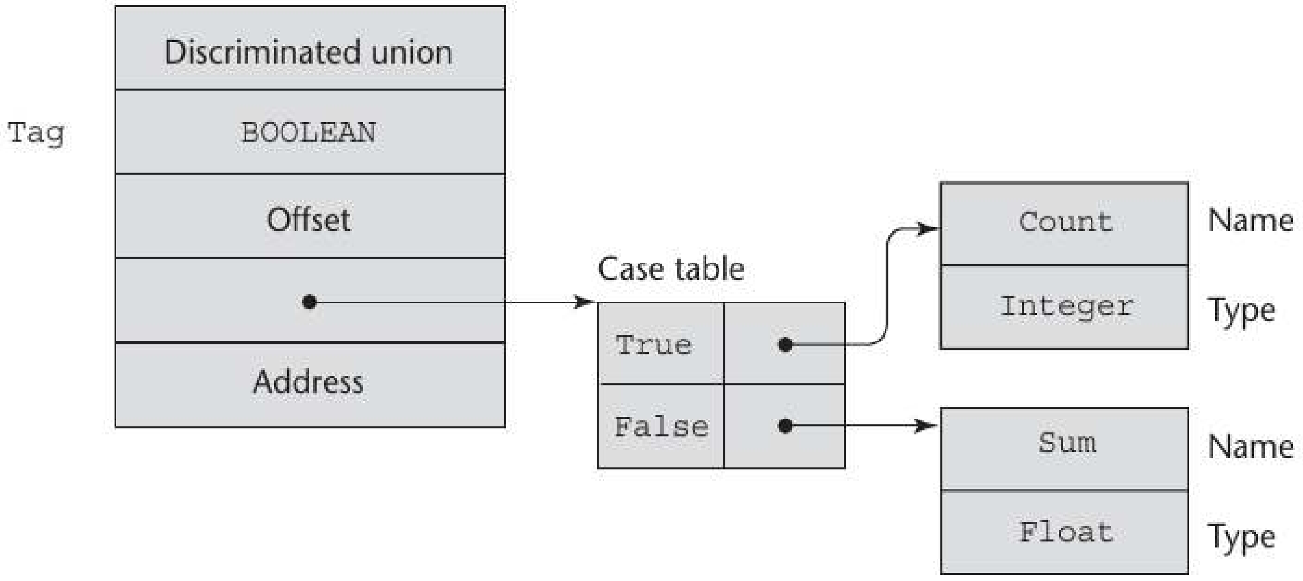
Figure_2 is constrained variant record.

Unions in F#

```
 type intReal =
| IntValue of int
| RealValue of float;;
```

A compile-time descriptor for a discriminated union

```
type Node (Tag : Boolean) is
        record
            case Tag is
                when True => Count : Integer;
                when False => Sum : Float;
        end case;
    end record;
```



9. Pointer and reference Type

Definition:

- A pointer type is one in which the variables have a range of values that consists of memory addresses and a special value, nil.
- A pointer can be used to access a location in an area where storage is dynamically allocated called a heap.
- Variables that are dynamically allocated from the heap are called **heap-dynamic variables**.

Dangling Pointer

A dangling pointer, or dangling reference, is a pointer that contains the address of a heap-dynamic variable **that has been deallocated**.

Example from C++ Language:

```
 int * ptr1;
int * ptr2 = new int[100];
ptr1 = ptr2;
delete [] ptr2;
```

Here, ptr1 and ptr2, both will be dangling pointers.

Java class instances are implicitly deallocated (there is no explicit deallocation operator), there cannot be dangling references in Java.

Reference Type:

A reference type variable is similar to a pointer, with one important and fundamental difference: A pointer refers to an address in memory, while a reference refers to **an object or a value in memory**.

Example:

```
int a = 0;
int &b = a;
b = 100;
```

In this code segment, variables a and b are aliases. b is reference to a.