Principle of Programming Language- Data Types

Data

- Data is collection of bits.

Simple Data Types

- Integers
- Reals
- Booleans
- New data types can be constructed from these data entities.

Are Data types Machine Independent?

- Integers
- Floats

Reasons to have static type checking

- It enhances execution efficiency.
- Improves translation efficiency.
- Static type checking allows many standard errors to be caught early improving writability.
- Static type checking improves security and reliability of a program by reducing the number of execution errors that can occur.
- Enhances Readability.

Reasons to have static type checking

- It removes ambiguity and type information can be used to resolve overloading.
- Correctness can be easily verified.
- Interface consistency can be easily verified.

Data Types and Type Information

- Program data can be classified according to their types.
- Eg:- -1 is of type int
- 3.14159 is of type double
- “hello” is of type string or group of characters

Data Type Definition

- Data type is a set of values.
- Eg:- int x means………x ∈ *Integers*
- i.e In Java x can take the value from

-2,147,483,648 to 2,147,483,647, since integers are always stored in 32-bit two's-complement form.

- A set of values also have group of operations that can be applied to the values. These operations are not mentioned explicitly with the type, but are part of its definition.

Data Type Definition

- Eg:- arithmetic operations on integers or reals.
- The subscript operation on arrays.
- The structure member operator ‘.’ on structure.
- These operations also have specific properties such as commutative associative etc.

  [e.g., $(x + 1) - 1 = x$ or $x + y = y + x$]

Data Type Definition

- Thus, to be even more precise, revise our first definition to include explicitly the operations, as in the following definition:

- A data type is a set of values together with a set of operations on those values having certain properties.

Type checking

- The process a translator goes through to determine whether the type information in a program is consistent is called type checking.

Type inference

- The process of attaching types to expressions is called type inference.

Type constructors

- The mechanism of constructing complex types from simple types is called type constructors. The type created are called as **user defined types**.
- int a[10];   // creates in C (or C++) a variable whose type is “array of int” (there is no actual array keyword in C)
- Names for new types are created using a **type declaration** (called a **type definition** in some languages).
- For example, the variable a, created above as an array of 10 integers, has a type that has no name (an **anonymous type**).
- To give this type a name, we use a **typedef** in C:
- typedef int Array_of_ten_integers[10];
- Array_of_ten_integers a;

Type Equivalence algorithm

- Compare two types to determine whether they are same.

Type System

- The type constructors, type equivalence algorithm, type inference and type checking are collectively known as type system.

Strongly Typed Languages

- If language definition specifies complete type system that can be applied statically then such a language is said to be Strongly typed language. (very strictly type-checked language)
- Eg:-Ada

Un typed or dynamically typed languages

- Languages without static type systems are called un typed languages.
- Eg:-Scheme, smalltalk,perl

Weakly typed languages

- The languages which are partially typed are called as weakly typed languages.
- eg:-C++,Java,C

Simple types and complex types

- Predefined types are primarily **simple types:** types that have no other structure than their inherent arithmetic or sequential structure. All the foregoing types except String and Process are simple.
- However, there are simple types that are not predefined: **enumerated types** and **subrange types** are also simple types.
- Enumerated types are sets whose elements are named and listed explicitly. A typical example (in C) is

```
enum Color {Red, Green, Blue};
```

- Simple types Examples:-int, float, double, boolean etc
- Complex types Examples:- class, struct, arrays etc

Simple types and complex types

What is the output of the following code?

```
#include <stdio.h>
enum Color {Red,Green,Blue};
enum NewColor {NewRed = 3, NewGreen = 2, NewBlue = 2};
main(){
    enum Color x = Green;
    enum NewColor y = NewBlue;
    x++;
    y--;
    printf("%d\n",x);
    printf("%d\n",y);
    return 0;
}
```

Java omits the enum construction altogether.

Type constructors

- Since data types are sets, set operations can be used to construct new types out of existing ones.
- Such operations include Cartesian product, union, power set, function set, and subset.

Set Operations

- Cartesian product
- Union
- Powerset
- Function set
- Subset

Cartesian product

- Given two sets U and V we can form the cartesian product as
- UXV={(u,v)|u is in U and v is in V}
- Eg:-struct

Cartesian product

```
struct IntCharReal{
int i;
char c;
double r;
};
```

In cartesian product we refer to members by their position while in structure we refer to members by their name.

Union

- Unions can be useful in reducing memory allocation requirements when different data items are not needed simultaneously.
- This is because unions are generally allocated space equal to the maximum of the space needed for the individual components.

Types of Union

- **Discriminated union**
- **Un discriminated union**

Discriminated union

- A union is **discriminated** if a **tag** or **discriminator** is added to the union to distinguish which type the element is
- Eg: Anonymous union in C++ with struct

```
enum Disc {IsInt,IsReal};
struct IntOrReal{ // C++ only!
enum Disc which;
union{
int i;
double r;
}; // no member name here
};
```

Undiscriminated union in C

```
union IntOrReal{
int i;
double r;
};
```

discriminated union in C

```
enum Disc {IsInt,IsReal};
struct IntOrReal{
enum Disc which;
union{
int i;
double r;
} val;
};
```

Usage of **discriminated union in C**

```
IntOrReal x;
x.which = IsReal;
x.val.r = 2.3;
. . .
if (x.which == IsInt) printf("%d\n",x.val.i);
else printf("%f\n",x.val.r);
```

Subset

- In mathematics a subset can be specified by giving a rule to distinguish its elements.
- Eg:- Positive integers={x|x is an integer and x>0}
- Eg:-Inheritance

Functions

- The set of all functions can give rise to new types in two ways:
- As an array type
- As a function type

Functions

- typedef int TenIntArray [10];
- typedef int IntArray [];

Legal declarations

- TenIntArray x;
- int y[5];
- int z[] = {1,2,3,4};
- IntArray w = {1,2};

Functions

- In C the size of an array cannot even be a computed constant—it must be a literal
- const int Size = 5;
- int x[Size]; /* illegal in C, ok in C++ (see Section 8.6.2) */
- int x[Size*Size] /* illegal in C, ok in C++ */

Functions

```
int f(int size) {
int a[size]; /* illegal  both in C and C++*/
. . .
}
```

Functions

- Java takes a rather different approach to arrays. Like C, Java array indexes must be nonnegative integers starting at 0. However, unlike C, Java arrays are always dynamically (heap) allocated. Also, unlike C, in Java the size of an array can be specified completely dynamically (but once specified cannot change unless reallocated).

JAVA

- Java allows widening conversion and it do not allow narrowing conversion.

References

Text book

- Kenneth C. Louden “Programming Languages Principles and Practice” second edition Thomson Brooks/Cole Publication.

Reference Books:

- Terrence W. Pratt, Masvin V. Zelkowitz “Programming Languages design and Implementation” Fourth Edition Pearson Education.
- Allen Tucker, Robert Noonan “Programming Languages Principles and Paradigms second edition Tata MC Graw –Hill Publication.