innovate achieve lead

BITS Pilani

Lecture 7
Primitive and Constructed Data Types

Primitive Data objects

- Directly manipulated by the underlying machine
- Integers and other primitive values are the first class citizens
- Operations on basic values are built into the languages
- Programmer defined data objects are constructed from simpler types

Storage representation of the data object : Compile time Layout

- Independent of the physical location of the data in memory
- Usually described in terms of the size of the block of memory required
- Layout design of the data type in a language is decided at the language design phase
- The relative offset of the values can be computed at the compile time
- Implementation of layout as actual memory allocation is an issue looked after by the Operating System.

Primitive data type

- A primitive data type is provided by a programming language as a basic building block
- Contains a single data value
- Most languages have most of the following data types
  - Integer
  - Floating Point Number
  - Character
  - Boolean (not primitive in C)

Primitive data type

- Values are determined by the underlying machine
- The machine word size determines the range of values
- Values associated with primitive data types can be used freely
  1. Can be compared for equality, can be assigned to any other variable, can be passed as parameters
  2. Operations on the values correspond to simple machine instructions

Primitive data types supported in C

- C permits the programmer to choose among four types of integer data and two types of real data
  - int
  - Short
  - Long
  - char
  - float
  - double
- C does not support boolean data type (can be defined as an enumeration)

Primitive data types supported by Pascal

- boolean - true or false
- char - Character
- real - Numbers with decimal points or exponents
- integer - Whole numbers
- set - A group of values

Layout of Primitive data types

- Layout is a plan for fixing relative locations logically corresponding to the variable names.
- Laid out physically at run time by using the machine representation of the values at the actual locations.
- On most machines
  - A char is one byte
  - An integer is one word long
  - A real number fits in two contiguous words

Operations on primitive data type

- Primitive operations on integers are +,-,*,/,%
- Programmer defined operations are implemented in the form of functions or procedures
- An operation can be defined as a mathematical function

  $+: \text{integer} \times \text{integer} \rightarrow \text{integer}$

Signatures of operations on primitive data type

- Binary arithmetic operators
  - +, -, *, %, / : integer x integer → integer (or real – a design issue)
- Unary arithmetic operators
  - +, - : integer → integer
- Binary relational operators
  - <, <=, ==, >=, >, !=: integer x integer→ boolean
- Binary logical operators
  - &&, ||, ! :boolean x boolean → boolean

Character data type

- A character type contains a single letter, digit, punctuation mark, or control character.
- ASCII character set has 128 characters
- Hence require only 7 bits to represent each character uniquely
- Uses a byte of space

Enumerated data types

- The enumerator names are usually identifiers that behave as constants in the language.

<u>In C language</u>

- enum {Sun=0,Mon, Tues, Wed, Thurs, Fri, Sat} week;
- Variable declaration   week day;
- The elements of an enumeration are ordered

<u>In Pascal language</u>

```
        var suit: (clubs, diamonds, hearts, spades);
```

Layout of enumerated data type

- Values and variables of an enumerated type are usually implemented as
  - fixed-length bit strings,
  - format and size compatible with some integer type

Composite (Constructed) data types

- Constructed in a program using its programming language's primitive data types and other composite types
  - Arrays
  - Records
    - Fixed fields (example **struct** in C)
    - Variable fields(example **union** in C)
  - Sets
  - Pointers
  - Abstract data types

Arrays

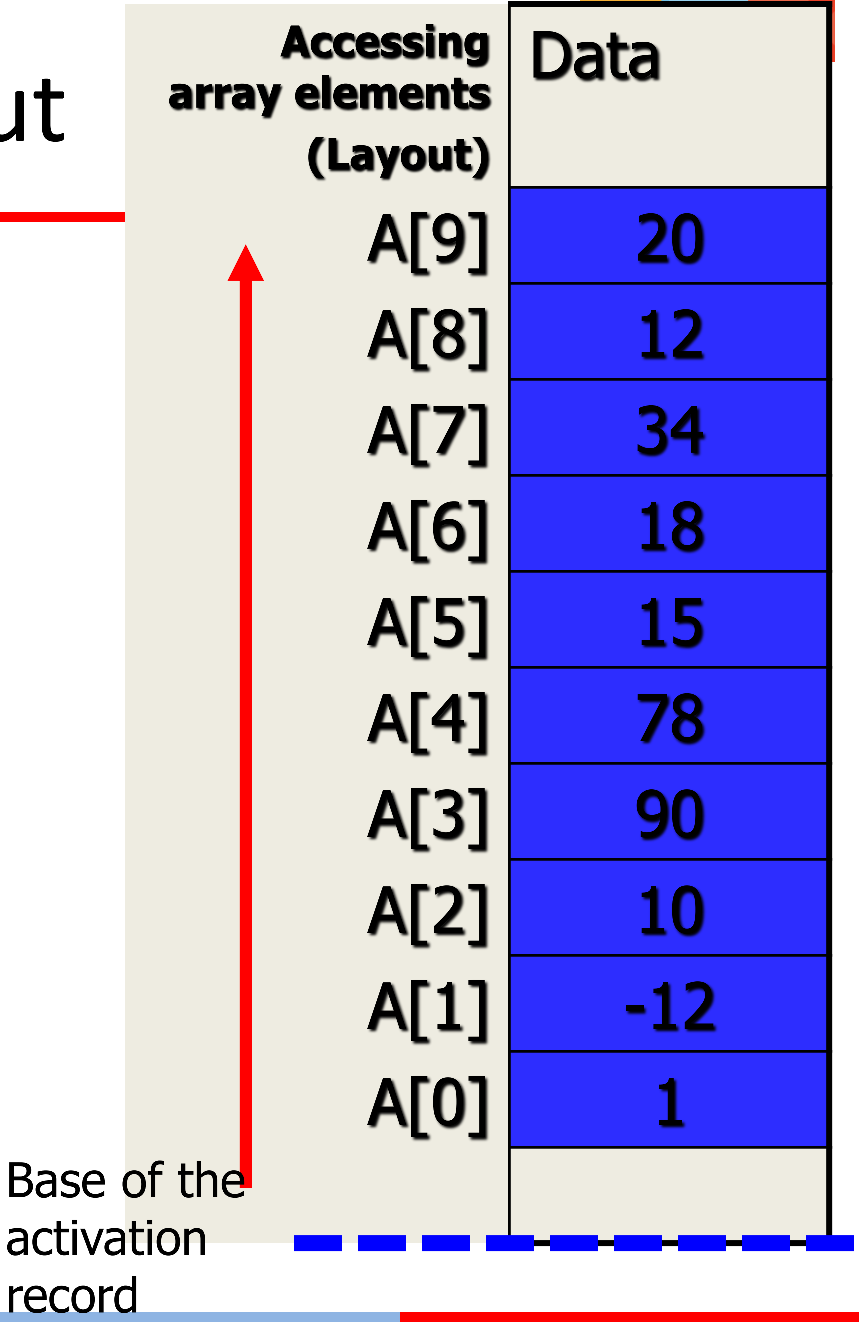
- Describes a collection of *elements* of the same type.
- Array elements are laid out in consecutive machine locations
- Each elements of an array occupies the same amount of space

Advantage of using array data type

- Array access is not sequential.
- Array element access time is not dependent on its actual location. (Random access)
- All data belonging to the same context from user perspective resides together.
- An array element's address can directly be computed at compile time, therefore all elements can have their relative positions fixed at compile time.

Array layout

- int A[10];



Storage Layout of an array : Example

int A[10];

(in C Language)

Var A: Array[12 ..21] of integer

(in pascal Language)

Let the array elements take the following values

20, 12, 34, 18, 15, 78, 90, 10, -12, 1

Accessing array elements (Pascal Layout)	Data
A[12]	20
A[13]	12
A[14]	34
A[15]	18
A[16]	15
A[17]	78
A[18]	90
A[19]	10
A[20]	-12
A[21]	1

Relative address of an array element A[i] (in C Language)

- Any element of the array can be accessed in constant time (random access)
- subrange is 0 to n-1 (if size of the array is n)
- A[i] resides in the location that is at a relative address = i * w
- As i can also be computed at run time, the size of the element (w) through its type is known at compile time, therefore **i * w can be computed in constant time** to access the value of A[i]

Array data type

- Array data type allows the element indices to be computed at run time.
- Example: int A[30]; // in C language

```
for(int i=0;i<10;i++)
        A[(i+1)*2] = i*4;
```

  is a valid C statement as the index is computed at run time.
- An error occurs if (i+1)*2 =30, i.e. if the loop is executed for i<15 (semantic error)