

13. Data Representation

13.1 User-defined data types

Purpose

- To create a **new data type**.
- To allow data types **not available** in a programming language to be constructed.

Non-composite types

- Data types constructed by the **programmer**.
- User-defined types that contain **only one data type** in its definition.

Enumerated

- A **user-defined, non-composite** data type with an **ordered list of possible values**.

```
TYPE <Identifier> = (Value1, Value2, ...)
```

- Enumerated acts as an *iterator*.

```
TYPE lastName = (Zhao, Qian, Sun, Li, ...)
DECLARE thisName, nextName : lastName
thisName <- Zhao
nextName <- thisName + 1
thisName <- nextName + 2
OUTPUT thisName, nextName // "Li Qian"
```

- Has an **implied order** → can refer to the next value in list.

```
yourBirth <- September
myBirth <- October
OUTPUT myBirth > yourBirth // True
```

- The enumerator names are usually *identifiers* that behave as constants; the enumerators are assigned arbitrary values.
- **Use cases of enumerated:**
 1. Data that has limited possible values.
 2. Data that has an implied order.
 3. Data that doesn't change frequently.

Pointer

- A **user-defined, non-composite** data type used to **reference a memory location**.
 - *Type* is the data type of the value that the pointer points to.
 - The object's type must correspond with the pointer's type.

```
TYPE <Identifier> = ^<Type>
```

- Refer to the address of a variable as follows.

```
TYPE intPointer = ^INTEGER
DECLARE n1 : INTEGER
DECLARE p1 : intPointer
n1 <- 108
p1 <- ^n1 // left "^" denotes finding the address of n1
OUTPUT p1 // 0x005ab, the address stored in p1
```

- We can also **de-refer** the pointer.

```
TYPE intPointer = ^INTEGER
DECLARE n1, n2 : INTEGER
DECLARE p1 : intPointer
n1 <- 36
p1 <- ^n1
n2 <- p1^ * 2 // right "^" denotes finding the value at p1
OUTPUT n2
```

- **Benefits of pointers:**
 1. *Dynamic memory allocation:* Allocating memory at run time.
 2. *Dynamic memory management:* Use for arrays, trees, etc; manipulate this data without copying.
 3. *Efficiency:* Allow passing large data structures by reference to them instead of copying the entire data.

Composite types

- Data types constructed by the **programmer**.
- User-defined types that **refer to any other data type** in its type during definition.

Record

```
TYPE <Identifier>
  DECLARE <Identifier> : <Type>
  DECLARE <Identifier> : <Type>
  ...
ENDTYPE
```

Class/Object

- A composite data type that includes **variables of given data types** and **methods**.

Set

```
TYPE <Set Identifier> = SET OF <Base Type>
DEFINE <Identifier> (Value1, Value2, ...) : <Set Identifier>
```

- A **given collection** of **unordered elements**.
 - No order; no duplicates.

13.2 File organization and access

File Organization

- Includes **serial**, **sequential**, and **random**.

Serial

- Physically stores records of data in a file in the order **they were added to the file**.
- **Benefits:**
 - New records can easily be appended.
 - **No need to re-sort** data every time new data is added.
 - Readings are stored in a **chronological order**.
 - Allowed the readings to be read in the order they were taken.
 - *Contextualize*: Because the the amount of values in the questions isn't large, so searching may only take limited amount of time.

Sequential

- Physically stores records of data in a file in a **given order**.
 - The order is usually based on the key field.
- *Describe how records are organizes and accessed in a sequential file:*
 - Records are stored in a **particular order**.
 - The order is determined based on the **value in the key field**.
 - Records accessed one after the other,
 - Records can be found by searching from the beginning of the file, record by record, until the required record is found or key field value is exceeded.

Random

- Physically stores records of data in a file in **any available position**.
 - It is based on a *hashing algorithm*.

Key Field \rightarrow hashing calculation \rightarrow Address

- **Collision** may happen when the hashed values are replicated.

File Access

- Includes **sequential** and **direct**.

Sequential Access

- When a particular record is searched, a *linear search* is carried out.
 - Every record needs to be checked until that record is found, or the whole file has been searched and the record has not been found.
- Used for **serial** and **sequential** file organization.
- For *Serial*: Whenever an **empty memory location** is reached or the whole file has been scanned through, as data is appended in a sequential manner.
- For *Sequential*: The check shall stop when the key field of the current record is greater than that of the record to be searched for.

Direct Access

- Used for **sequential** and **random** file organization.
- For *sequential*: An **index table** is created, which pairs key fields with their respective file positions. When the position of a specific key field is found, one can directly jump to the data at the position.
 - The index table is always **small in size**, making the address lookup faster.
- For *random*: The hashing algorithm is used on the key field to calculate the address of the file location where a given record is stored.

Hashing Algorithm

- A mathematical function that generates a **fixed-size output** from an input of variable size.
- **Collision**:
 - A collision occurs when the record key doesn't match the stored record key (for searching).
 - This means that the determined storage location has already been used for another record.

Closed Hash (Open Addressing)

- A *Collision resolution* strategy where collisions are resolved by storing the collided value in the next available storage space.
- There are two strategies for querying the next available location:
 - Linear Probing:

$$\text{Index} = H(k) = H(k) + 1 = H(k) + 2 = H(k) + 3 = \dots$$

- Quadratic Probing (used for *congestive scenarios*)

$$\text{Index} = H(k) = H(k) + 1 = H(k) + 2 = H(k) + 4 = \dots$$

Open Hash

- A *Collision resolution* strategy where an overflow is set up and the record is stored in the *next available location* in the **overflow area**.

Separate chaining

- A *Collision resolution* strategy where collisions are resolved by storing all colliding keys in the same slot (using linked list, etc).

13.3 Floating-point numbers, representation and manipulation

$$\text{Floating Point Number} = M \times 2^E$$

Mantissa M	$\frac{-128}{128}$	$\frac{64}{128}$	$\frac{32}{128}$	$\frac{16}{128}$	$\frac{8}{128}$	$\frac{4}{128}$	$\frac{2}{128}$	$\frac{1}{128}$
Exponent E	-128	64	32	16	8	4	2	1

- **Exponent** determines the range that the number is in, i.e., [0.5, 1], [1, 2], [2, 4]...
 - **Exponent:** The range of the number.
- **Mantissa** splits the range into different pieces, and chooses the nearest piece to represent the number, i.e., 1/128 of the range, 64/128 of the range.
 - **Mantissa:** The precision of the number.

Normalizations

- **Positive numbers** shall have a mantissa starting with 0.1.
- **Negative numbers** shall have a mantissa starting with 1.0.
- If numbers are not normalized...
 1. *Precision Lost:* Less bits can be used to include information in the mantissa.
 - E.g., considering 0.001×2^3 and 0.1×2^1
 2. *Bits lost off at right hand end.*
 3. *Multiple representations of a single number:* Inefficient for calculations.
- **Problem** with normalization: The value 0 cannot be represented.

Problems

- **Some normalized values cannot be stored** → Mantissa isn't large enough (e.g., representing 513 in a 10-bit mantissa system).
 1. More bits required in the mantissa (specify the number of bit, i.e., 11 bits)
 2. Specify the normalized value (i.e., 0.1000000001).
 3. Results in **overflow** → value greater than the range to be represented.
 - **Solution:** use more bits for mantissa, and less bits for exponent.
- **Rounding Error:**
 - There is no exact binary conversion for some numbers.
 - More bits are needed to store the number than are available (e.g., the precision is not enough to represent a number with many bits after the decimal point).
- **Overflow:** when the exponent/mantissa has become too large to be represented using the number of bits available.
- **Underflow:** when the exponent/mantissa has become too small to be represented using the number of bits available.