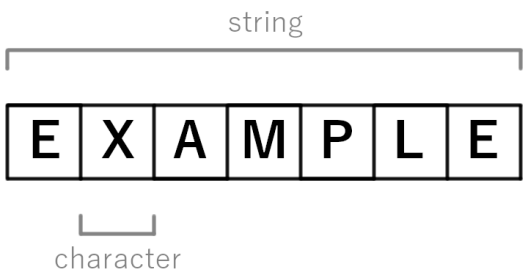
The National School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي



Data Structure & Algorithms 1

CHAPTER 4
STATIC DATA STRUCTURE (PART3): STRING & STRUCT

Sep – Dec 2023

Outline

- **Strings**
  - Definition
  - Declaration
  - Example
  - Strings in C++
- **Struct**
  - Definition
  - Declaration
  - Example
  - Struct in C++

STRINGS

- **CONSTANT String:**

It is a sequence of characters enclosed in apostrophes:

- **Examples**:
  - "Give an integer between 1 and 10:"
  - "This program gives the Nth term of the FIBO sequence"
  - "The equation has no solutions"

STRINGS

- **VARIABLE String:**

It is a 1D array of characters:

Sentence[6]

Sentence	H	e	l	l	o		W	o	r	l	d
	0	1	2	3	4	5	6	7	8	9	10

- The access to an element is done by specifying the name of the string followed by the index in square brackets, IDENTICAL to that of a one-dimensional array (1D).

STRINGS

- `A = Title[2]`
- `B = Title[i]`
- `C = Title[i + j]`
- `Title[2] = k`
- `Title[i] = Z`
- `Title[i DIV 2] = V`

STRINGS

Remark:

- The maximum length of a string is 255.
- An additional position contains the length of the string. (The length of the string is stored in a char (Byte, i.e., max = 255))
- The dimension of a string is its length, i.e., the number of characters it comprises. By default, the dimensional length of a string is 255.

STRINGS

- **FORMAT:**

  ```
  String my_string
  OR
  String my_string [max_size]
  ```

- **Example:**
  - `String P`
  - `READ (P)`
  - `IF P[i] = ‘A’ THEN count = count + 1`

STRINGS
Standard Functions & Procedures

```
String Function Copy(String str, Integer P, Integer N)
```

Description:
This function returns a sub-string of `N` characters starting from the position `P` of the string `str`

Example:

```
STR = Copy ("NationalSchool", 8, 6)  //STR = "School"
```

STRINGS
Standard Functions & Procedures

```
String Function Concat(Strinf str1, String str2 )
```

Description:
This function returns the concatenation of two strings `str1` and `str2`

Example:

```
STR1 = “School”
STR2 = Concat(“National ”, STR1)  //STR2 = “National School”
```

STRINGS
Standard Functions & Procedures

```
Integer Function Pos(Strinf subStr, String str )
```

Description:
This function returns the first position of the substring `SubStr` in the string `str.` If the substring `subStr` does not exists in the string `str,` the function returns -1.

Example:

```
p = Pos("School", "National School") //p = 9
p = Pos("R", "Program")             //p = -1
```

STRINGS
Standard Functions & Procedures

```
Integer Function Length(String str )
```

Description:
This function returns the length of the string `str`.

Example:

```
N = Length("National School")    //N = 15
```

STRINGS
Standard Functions & Procedures

```
Procedure Delete(Var String str, Integer P, Integer N)
```

Description:
Delete `N` characters from `str` starting from the position `P`.
If the string contains less than N char from P, delete till the end of the string.

Example:

```
String str = "National School"
Delete(str, 8, 7)   //str = "National"
```

STRINGS
Standard Functions & Procedures

```
Procedure Insert(String str2, Var String str1, Integer P)
```

Description:

Inserts a string `str2` into another string `str1` at position `P`.

Example:

```
String str1 = “Hi! how are you?”
String str2 = “ Mahmoud”
insert(str2, str1, 2)   //str1 = “Hi Mahmoud! how are you?”
```

STRINGS
Standard Functions & Procedures

```
Procedure Str(Integer/Real N, Var String str)
```

Description:

Convert a numerical value `N` into a string `str`.

Example:

```
Str (2012, str) // str = '2012' ;
Str (14.52, str) // str = '1.4520000000E+01’
```

STRINGS
Standard Functions & Procedures

```
Procedure Val(String str, Var Integer N, Var Integer Err)
```

Description:
Convert a string `str` to a numerical value `N`. Additionally, it provides an error code `Err` indicating whether the operation was carried out successfully.

Example:

```
Val ("2021", n, err) // n = 2021 and err = -1;
VAL ("45A6", n, err) // n=0 and  err= 2
n is Integer: Val ("3.14", n, err ) // n = 0 and err = 1
n is Real:    Val ("3.14", n, err ) // n = 3.14 and err = -1
```

STRINGS

Application : Palindrome

Knowing that a palindrome is a phrase that can be read in both directions, construct the algorithm that allows us to determine if a phrase is a palindrome or not.

Example:

- Engage le jeu que je le gagne
- Name now one man

STRINGS

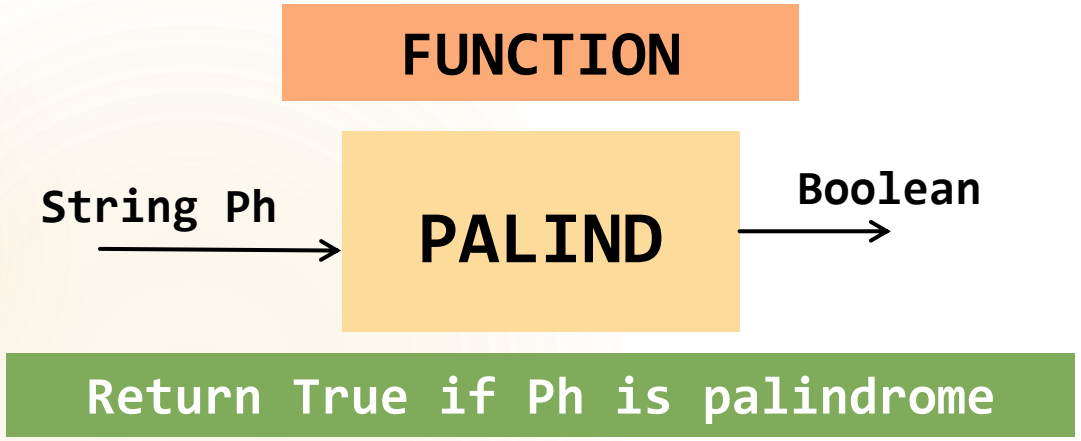
Application : Palindrome

Modular breakdown:

- We are going to build a module that checks if a given **`string`** is a palindrome.
- We will also need the standard functions:
  - Length, the length of our initial string
  - Upercase, convert everything to uppercase since we do not know if the characters are in uppercase, lowercase, or a mix of both
  - Ord: to determine the alphabetical order number of a character in the ASCII list in order to consider only characters ranging from the 65th to the 90th position ('A', 'B', 'C', ..., 'Z').

STRINGS

Application : Palindrome



Construction of PALIND

Analysis

The basic idea is:

- To rewrite the original string forwards (`Str1`) and backward (`Str2`) while removing all characters other than alphabetic characters.
- Then, if `Str1` equals `Str2`, the two strings are identical, indicating a palindrome.

STRINGS

Application : Palindrome

- We start with two empty character strings, `Str1` and `Str2`:
- Traverse the original string `Ph` (`i` ranging from 0 to `Length(Ph-1))`.
  - If `Ph[i]` is an alphabetic character (from A to Z),

    `If (ORD(Upercase(Ph[i])) >= 65) AND (ORD(Upercase(Ph[i])) <= 90),`

    - `Str1 = Str1 + Upercase(Ph[i])` (writing it forward),
    - `Str2 = Upercase(Ph[i]) + Str2` (writing it backward).
  - If `Str1 == Str2`, then `Ph` is a palindrome.

STRINGS

Application : Palindrome

```
Boolean FUNCTION PALIND (String Ph)
Variables
        String Str11, Str2
        Integer i
BEGIN
   Str1 = ‘’
   Str2 = ‘’
   FOR i FROM 0 To Length (Ph) - 1 DO
      IF (ORD(Upercase (Ph[i])) >= 65) AND(ORD (Upercase (Ph[i])) <= 90) THEN
         Str1 = Str1 + Upercase(Ph[i])
         Str2 = Upercase(Ph[i]) + Str2
      ENDIF
   END FOR
   IF (Str1 == Str2) THEN PALIND = TRUE
   ELSE PALIND = False
END
```

STRINGS in C++

Template class **basic_string**

- String manipulation (copying, searching, etc.)
- Include **<string>**

- **string** initialization
  - `string s1( "Hello" ); // constructor`
  - `string s2( 8, 'x' );  // repeated char`
    - 8 'x' characters
  - `string month = "March“ // assignment op`
  - `string s;  // Default empty string`
  - `Char arr = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’};`
    - `string s(arr, 5);  //with char array`

STRINGS in C++

No **implicit** conversion from **`int`** or **`char`**

- The following definitions are errors
  - `string error1 = 'c';`
  - `string error2( 'u' );`
  - `string error3 = 22;`
  - `string error4( 8 );`
- However, can assign to one **`char`** if declared
  - `s = 'n';`

STRINGS in C++

string features

- Not necessarily **`null`** terminated
- **length** member function: **`s1.length()`**
- Use **`[]`** to access individual characters: **`s1[0]`**
  - **`0`** to **`length-1`**
- **string** not a pointer
- Many member functions take start position and length
  - If length argument too large, max chosen
- Stream extraction
  - **`cin >> stringObject;`**
  - **`getline( cin, s)`**
    - Delimited by newline

STRINGS in C++

String Assignment and Concatenation

Assignment

- `s2 = s1;`
  - Makes a separate copy
- `s2.assign(s1);`
  - Same as `s2 = s1;`
- `myString.assign(s, start, N);`
  - Copies `N` characters from `s`, beginning at index `start`
- Individual characters
  - `s2[0] = s3[2];`

STRINGS in C++

String Assignment and Concatenation

- Range checking
  - `s3.at( index );`
    - Returns character at `index`
    - Can throw `out_of_range` exception
- Concatenation
  - `s3.append( "pet" );`
  - `s3 += "pet";`
    - Both add `"pet"` to end of `s3`
  - `s3.append( s1, start, N );`
    - Appends `N` characters from `s1`, beginning at index `start`

STRINGS in C++

Comparing strings

Overloaded operators

- `==, !=, <, >, <=` and `>=`
- Return `bool`

- `s1.compare(s2)`
  - Returns positive if `s1` lexicographically greater
    - Compares letter by letter
    - `'B'` lexicographically greater than `'A'`
  - Returns negative if less, zero if equal
  - `s1.compare(start, length, s2, start, length)`
    - Compare portions of `s1` and `s2`
  - `s1.compare(start, length, s2)`
    - Compare portion of `s1` with all of `s2`

STRINGS in C++

Substrings & Swapping

- Function **`substr`** gets substring
  - **`s1.substr( start, N );`**
  - Gets **`N`** characters, beginning with index **`start`**
  - Returns substring

- **`s1.swap(s2);`**
  - Switch contents of two strings

STRINGS in C++

Characteristics

- Member functions
  - **`s1.size()`** and **`s1.length()`**
    - Number of characters in string
  - **`s1.capacity()`**
    - Number of elements that can be stored without reallocation
  - **`s1.max_size()`**
    - Maximum possible string size
  - **`s1.empty()`**
    - Returns **`true`** if empty
  - **`s1.resize(newlength)`**
    - Resizes string to **`newlength`**

STRINGS in C++

Finding Strings and Characters in a string

- Find functions
  - If found, index returned
  - If not found, **`string::npos`** returned
    - Public static constant in class string
  - **`s1.find( s2 )`**
  - **`s1.rfind( s2 )`**
    - Searches right-to-left
  - **`s1.find_first_of( s2 )`**
    - Returns first occurrence of any character in **`s2`**
    - **`s1.find_frist_of( "abcd" )`**
      - Returns index of first **`'a'`**, **`'b'`**, **`'c'`** or **`'d'`**

STRINGS in C++

Finding Strings and Characters in a string

- Find functions
  - **`s1.find_last_of( s2 )`**
    - Finds last occurrence of any character in **`s2`**
  - **`s1.find_first_not_of( s2 )`**
    - Finds first character NOT in **`s2`**
  - **`s1.find_last_not_of( s2 )`**
    - Finds last character NOT in **`s2`**

STRINGS in C++

Finding Strings and Characters in a string

- **`s1.erase( start )`**
  - Erase from index **`start`** to end of string, including **`start`**
- Replace
  - **`s1.replace( begin, N, s2)`**
    - **`begin`**: index in **`s1`** to start replacing
    - **`N`**: number of characters to replace
    - **`s2`**: replacement string
  - **`s1.replace( begin, N, s2, index, num )`**
    - **`index`**: element in **`s2`** where replacement begins
    - **`num`**: number of elements to use when replacing
  - Replacement can overwrite characters

STRINGS in C++

Inserting Characters into a string

- **`s1.insert( index, s2 )`**
  - Inserts **`s2`** before position **`index`**
- **`s1.insert( index, s2, index2, N );`**
  - Inserts substring of **`s2`** before position **`index`**
  - Substring is **`N`** characters, starting at **`index2`**

STRUCT

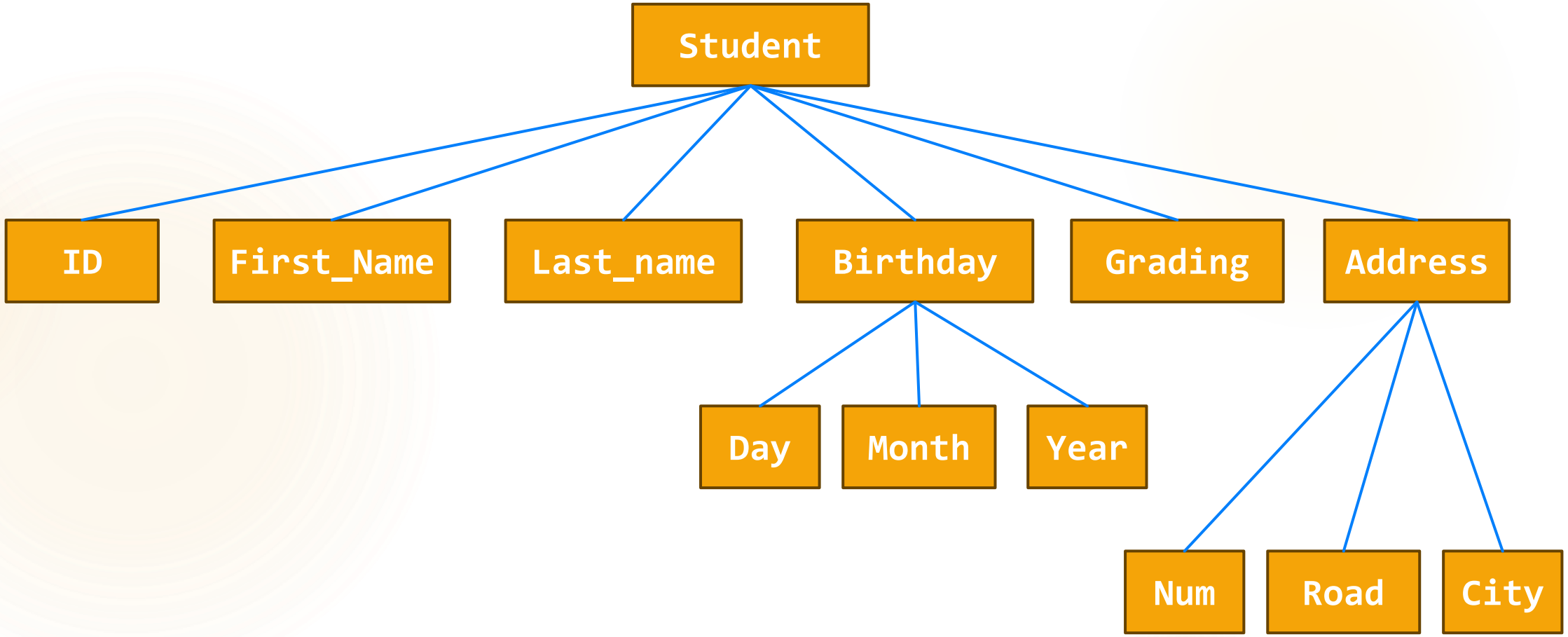
STRUCT
Definition

In programming, a **struct** is a collection of elementary elements, which may vary in types and are referred to as “member."

- These member are organized under a **common** object name.
- Each field can be either a **simple** element or a **structure** itself.

The purpose of a struct is to <u>encapsulate</u> a set of information related to a specific object within a unified structure.

STRUCT
Example



STRUCT
Declaration

The declaration of a struct is done by specifying the keyword "**STRUCT**," followed by the description of the fields, and concluded with the keyword "**END**."

```
STRUCT Birthday
     Integer day
     Integer month
     Integer year
END
Variable
     Birthday Student_birthday
```

STRUCT

Initializing a Struct:

```
MyStruct example = {42, 'A', 3.14}
```

Accessing:

Use the dot **(.)** operator to access members.

```
Integer day = Student_birthday.day
```

Updating Struct Members:

Modify values using the dot operator.

```
Student_birthday.day = 5
```

STRUCT

Declaration in C++

Data struct can be declared in C++ using the following syntax:

```
Struct type_name {
     member_type1 member_name1;
     member_type2 member_name2;
     member_type3 member_name3;
     .
     .
} object_names;
```

STRUCT
Declaration in C++

- `type_name` is a name for the structure type
- `object_name` can be a set of valid identifiers for objects that have the type of this structure.
- Within braces {}, there is a list with the data members, each one is specified with a type and a valid identifier as its name.

```
Struct type_name {
   member_type1 member_name1;
   member_type2 member_name2;
   member_type3 member_name3;
   .
   .
} object_names;
```

```
struct product {
   int weight;
   double price;
};

product Milk;
product banana, melon;
```

STRUCT
Declaration in C++

- If objects are specified on the end of the structure, the name of the type is optional
- The access to members can be accessed directly.
- The syntax for that is simply to insert a dot (`.`) between the object name and the member name.

```
apple.weight
apple.price
```

```
struct {
  int weight;
  double price;
} apple, banana,
melon;
```