# IUBAT – International University of Business Agriculture and Technology

## Assignment -01

**CSE- 461**

**Programming Language and Structure**

**Semester- Fall 2024**

**Section- D**

| Submitted to | Submitted by |
|---|---|
| **Md Nazir Ahmed**<br>Lecturer<br>CSE department, IUBAT | **Md. Bakhtiar Fahim**<br>ID - 21203060<br>Program - BCSE |

Date of Submission: 18. 12. 2024

## Briefly describe primitive data types with code. [6.2]

**Numeric Types:**

- Integer: Fixed-point numbers supported in multiple sizes (e.g., int, long).
- Floating-point: Approximations of real numbers.
- int num = 42; float pi = 3.14f;

**Boolean Types:**

- Represent true/false values.
- isOn = true;

**Character Types:**

- Used for single characters or strings.
- char = 'A'

---

## Answer design Issues of character string types, strings and their operations, string length options, evaluation and implementation with code. [6.3]

**Design Issues:**

- Should strings be a special kind of character array or a primitive type?
- Should strings have static or dynamic length?

**Common Operations of string:**

- **Assignment:** Copying one string to another.
- **Concatenation:** Combining two strings into one.
- **Substring Reference:** Accessing a part of a string.
- **Comparison:** Lexicographic comparison of strings.
- **Pattern Matching:** Finding matches based on defined patterns (e.g., regular expressions).

**String Length Options:**

1. **Static Length Strings:** Fixed length at compile-time.
2. **Limited Dynamic Length Strings:** Varying up to a maximum limit
3. **Dynamic Length Strings:** No fixed maximum limit.

**Evaluation**

1. **Writability:** string types improve simplicity and reduce errors in code. Libraries for string handling (e.g., C standard string library) are helpful but prone to security issues (e.g., buffer overflow in strcpy).
2. **Cost and Complexity:** Adding primitive string types has minimal impact on language or compiler complexity.

**Implementation:**

1. **Static Length String (Python):**

```
static_string = "Hello, World!"
print(f"Static String: {static_string}")
```

2. **Limited Dynamic Length String (C):**

```
#include <stdio.h>
#include <string.h>

int main() {
   char str[20];
   strcpy(str, "Hello");
   printf("String: %s\n", str);
   return 0;
}
```

3. **Dynamic Length String (C++):**

```
#include <iostream>
#include <string>

int main() {
   std::string dynamicStr = "Hello";
   dynamicStr += ", World!";
   std::cout << "Dynamic String: " << dynamicStr << std::endl;
   return 0;
}
```

4. **Dynamic Allocation (Java):**

```
public class Main {
   public static void main(String[] args) {
      String dynamicStr = "Hello";
      dynamicStr += ", World!";
      System.out.println("Dynamic String: " + dynamicStr);
   }
}
```

---

## Answer the design issues of enumeration, implement enum with code and evaluation. [6.4]

**Design Issues:**

- Is an enumeration constant allowed to appear in more than one type definition, and if so, how is the type of an occurrence of that constant in the program checked?
- Are enumeration values coerced to integer?
- Are any other types coerced to an enumeration type?

**Implementation Example:**

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

**Evaluation:**

1. **Readability:** Named constants (e.g., RED, BLUE) are more meaningful than numeric literals (e.g., 0, 1).
2. **Reliability**: Does not allow unintended operations (e.g., arithmetic) in most modern implementations (Java, C#). Moreover, Type-checking prevents assignments of out-of-range or invalid values (e.g., Colors myColor = 999; is illegal).

---

## Answer the design issues of the array types and briefly describe subscript bindings and array categories. [6.5]

**Design Issues:**

- What types are legal for subscripts?

- Are subscripting expressions in element references range checked?
- When are subscript ranges bound?
- When does array allocation take place?
- Are ragged or rectangular multidimensioned arrays allowed, or both?
- Can arrays be initialized when they have their storage allocated?
- What kinds of slices are allowed, if any?

**Subscript Bindings:**

The process of associating an array index (or subscript) with an element in the array.

1. **Static:** The subscript range is fixed at compile-time
2. **Dynamic:** The subscript range can change during the program's execution

**Array Categories:**

1. **Static Arrays**: have statically bound subscript ranges and storage allocated before runtime. Their main advantage is efficiency, as no dynamic allocation or deallocation is needed. However, their size is fixed for the entire program execution, limiting flexibility.
2. **Fixed Stack-Dynamic Arrays**: have statically bound subscript ranges, but their storage is allocated at declaration elaboration time during execution, using the stack. This allows memory to be reused across subprograms or blocks that are not active simultaneously, improving space efficiency. The trade-off is the additional time required for allocation and deallocation.
3. **Fixed Heap-Dynamic Arrays**: have subscript ranges and storage bindings fixed at runtime upon user request, but they use heap memory instead of the stack. This makes them more flexible as their size can be tailored to fit the problem. However, allocating memory from the heap takes longer than stack allocation.
4. **Heap-Dynamic Arrays**: have fully dynamic subscript ranges and storage allocation, which can change multiple times during the program's lifetime. This provides maximum flexibility, allowing arrays to grow or shrink as needed. The downside is that frequent allocation and deallocation can be time-consuming and inefficient.

---

## Describe design issues and evaluation of record types. [6.7]

**Design Issues:**

- What is the syntactic form of references to fields?
- Are elliptical references allowed?

**Evaluation:**

- Strong readability and writability due to intuitive field names. Records use descriptive field names, making it easier for programmers to understand the purpose of each field.
- High Reliability with type safety and predictable behavior ensuring robust programs.

---

## Describe design issues, implement union with code and evaluation of union types. [6.10]

**Design Issues:**

- Should unions support type safety?
- How is memory shared among fields?

**Implementation:**

```
union Data {
   int i;
   float f;
};
Data d;
d.i = 10;
```

**Evaluation:**

- Memory-efficient reducing cost
- Potentially unsafe and harder to manage. Modern languages like Java and C# prioritize safety and reliability by avoiding unions altogether.

---

## Describe design issues, implement pointers in c and c++ with code. [6.11]

**Design Issues:**

- How are dangling pointers and memory leaks handled?
- Should pointers be used for array indexing?

**Implementation in C:**

```
int x = 10;
int *p = &x;
```

**Implementation in C++:**

```
int x = 20;
int &ref = x;
```