📘 Compiler Design – UNIT III

🔥 MOST EXPECTED 14-MARK QUESTIONS

🎯 RGPV Exam Ready Answers (Easy + Detailed)

🔥 Q1. Explain Symbol Table. What information is stored in it?

✍️ Answer:

📌 Definition

A Symbol Table is a data structure used by the compiler to store information about identifiers used in a program.

Identifiers include:

```
Variables
Functions
Arrays
Objects
Labels
```

🌟 Purpose of Symbol Table

✅ Stores identifier information

✅ Helps in semantic analysis

✅ Supports type checking

✅ Helps code generation

🌟 Information Stored in Symbol Table

Field	Description
Name	Identifier name
Type	Data type
Scope	Local/Global
Address	Memory location
Size	Size of variable
Value	Current value

🌟 Example

Program:

```
int a;
float b;
```

Symbol Table:

Identifier	Type	Address
a	int	2000
b	float	3000

🌟 Functions of Symbol Table

1. Stores Variable Information

Stores data about identifiers.

---

2. Type Checking

Checks compatible data types.

---

3. Scope Management

Checks local and global scope.

---

4. Memory Allocation

Helps assign memory addresses.

🌟 Data Structures Used

Data Structure	Advantage
Linear List	Simple implementation
Hash Table	Fast searching
Tree	Ordered storage

🌟 Advantages

✅ Fast lookup

✅ Efficient compilation

✅ Easy identifier management

📌 Conclusion

Symbol Table is an important compiler data structure used for storing and managing identifier information efficiently.

🔥 Q2. Explain Type Checking and Type Conversion with Example.

✍️ Answer:

🌟 Type Checking

📌 Definition

Type Checking is the process of verifying whether operands and operators are compatible with each other.

🌟 Purpose

✅ Detect type errors

✅ Ensure program correctness

✅ Prevent invalid operations

🌟 Example

Correct:

```
int a = 10;
```

Incorrect:

```
int a = "Hello";
```

Compiler generates:

```
Type Mismatch Error
```

🌟 Types of Type Checking

Type	Description
Static Type Checking	Performed during compilation
Dynamic Type Checking	Performed during execution

🌟 Advantages

✅ Early error detection

✅ Better reliability

✅ Safe execution

🌟 Type Conversion

📌 Definition

Type Conversion means converting one data type into another.

🌟 Types of Type Conversion

Type	Description
Implicit Conversion	Automatic conversion
Explicit Conversion	Manual conversion

🌟 Example of Implicit Conversion

```
int a = 10;
float b = a;
```

🌟 Example of Explicit Conversion

```
float x = 5.6;
int y = (int)x;
```

🌟 Advantages of Type Conversion

✅ Supports mixed type operations

✅ Increases flexibility

🌟 Disadvantages

❌ Data loss may occur

📌 Conclusion

Type Checking ensures valid operations, while Type Conversion allows operations between different data types.

🔥 Q3. Explain Storage Allocation Strategies.

✍️ Answer:

📌 Definition

Storage Allocation Strategies determine how memory is allocated to variables and data during program execution.

🌟 Types of Storage Allocation

Type	Description
Static Allocation	Memory fixed before execution
Stack Allocation	Memory allocated during function calls
Heap Allocation	Dynamic memory allocation

🌟 1. Static Allocation

📌 Definition

Memory is allocated before program execution starts.

🌟 Example

```
int a;
```

🌟 Advantages

✅ Fast access

✅ Simple implementation

🌟 Disadvantages

❌ No flexibility

🌟 2. Stack Allocation

📌 Definition

Memory is allocated automatically during function calls.

🌟 Example

```
void fun()
{
   int x;
}
```

🌟 Advantages

✅ Efficient function handling

✅ Automatic memory release

🌟 Disadvantages

❌ Limited size

🌟 3. Heap Allocation

📌 Definition

Memory is allocated dynamically during execution.

🌟 Example

```
int *p;
p = (int*)malloc(sizeof(int));
```

🌟 Advantages

✅ Flexible memory usage
✅ Dynamic allocation

🌟 Disadvantages

❌ Memory leakage possible

📌 Conclusion

Different allocation strategies are used according to program requirements for efficient memory management.

🔥 Q4. Explain Dynamic Storage Allocation and Fragmentation.

✍️ Answer:

🌟 Dynamic Storage Allocation

📌 Definition

Dynamic Storage Allocation allocates memory during program execution according to need.

🌟 Example

```
int *p;
p = (int*)malloc(sizeof(int));
```

🌟 Advantages

✅ Flexible memory management

✅ Efficient memory usage

🌟 Disadvantages

❌ Memory leakage

❌ Fragmentation problem

🌟 Fragmentation

📌 Definition

Fragmentation occurs when free memory becomes divided into small unusable parts.

🌟 Types of Fragmentation

Type	Description
Internal Fragmentation	Extra unused memory inside allocated block
External Fragmentation	Free memory scattered outside blocks

🌟 Example of External Fragmentation

```
Used | Free | Used | Free | Used
```

Free memory exists but not in continuous form.

🌟 Causes of Fragmentation

✅ Frequent allocation/deallocation

✅ Variable sized memory requests

🌟 Solutions

✅ Memory compaction

✅ Paging

✅ Segmentation

📌 Conclusion

Dynamic Storage Allocation provides flexibility but may create fragmentation problems.

🔥 Q5. Explain Run Time Environment with Memory Layout.

✍️ Answer:

📌 Definition

Run Time Environment is the memory management environment used during program execution.

🌟 Functions

✅ Function handling
✅ Memory allocation
✅ Parameter passing
✅ Dynamic storage management

🌟 Memory Layout

```
Code Area
↓
Global Data Area
↓
```

```
Heap
↓
Stack
```

🌟 Components

---

1. Code Area

Stores program instructions.

---

2. Global Data Area

Stores global and static variables.

---

3. Heap

Stores dynamically allocated memory.

---

4. Stack

Stores:

- Local variables
- Function calls
- Parameters

🌟 Advantages

✅ Efficient execution

✅ Better memory management

📌 Conclusion

Run Time Environment manages all memory resources required during execution.

🔥 Q6. Explain Parameter Passing Methods.

✍️ Answer:

📌 Definition

Parameter Passing transfers data between calling function and called function.

🌟 Types of Parameter Passing

Method	Description
Call by Value	Copy of variable passed
Call by Reference	Original variable passed

🌟 1. Call by Value

📌 Definition

Copy of actual parameter is passed to function.

🌟 Example

```
void fun(int x)
```

Changes do not affect original variable.

🌟 Advantages

✅ Original data remains safe

🌟 Disadvantages

❌ Extra memory used

🌟 2. Call by Reference

📌 Definition

Address/reference of original variable is passed.

🌟 Example

```
void fun(int &x)
```

Changes affect original variable.

🌟 Advantages

✅ Efficient memory usage

✅ Faster execution

🌟 Disadvantages

❌ Original data may change

📌 Conclusion

Parameter Passing methods control communication between functions.

🔥 Q7. Explain Type System and Type Equivalence.

✍️ Answer:

🌟 Type System

📌 Definition

Type System is a set of rules used to define and manage data types in programming languages.

🌟 Functions of Type System

✅ Defines valid operations

✅ Detects type errors

✅ Improves safety

🌟 Example

```
int a = 10;
float b = 5.5;
```

Expression:

```
a + b
```

Allowed due to type conversion.

🌟 Type Equivalence

📌 Definition

Type Equivalence determines whether two data types are considered identical.

🌟 Types of Type Equivalence

Type	Description
Name Equivalence	Same name means same type
Structural Equivalence	Same structure means same type

🌟 Example

```
int a;
int b;
```

Both are equivalent types.

🌟 Advantages

✅ Simplifies type checking

✅ Improves compatibility checking

📌 Conclusion

Type System controls data types, while Type Equivalence compares types during compilation.