NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Engineering

Homework – 01

Name	: Joy Kumar Ghosh
Student ID	: 2211424 6 42
Course No.	: CSE 425
Course Title	: Concepts of Programming Language
Section	: 1
Date	: 07 October 2024

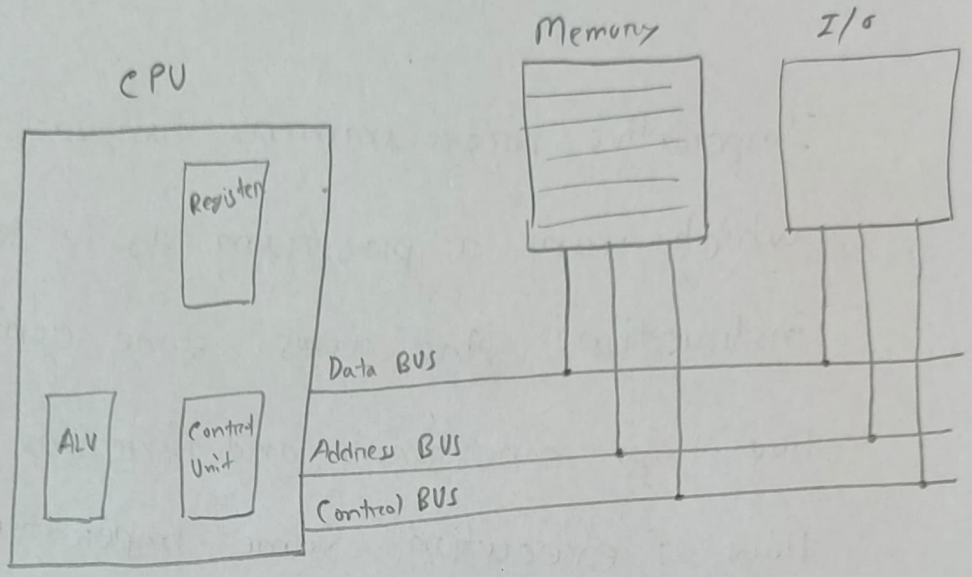
Ans. to the ques. no. 01

Imperative programming languages are those which runs a program as a sequence of instructions. And uses some control structures like loops, conditionals, and functions to manage the flow of execution. Some imperative languages are C, Java, and Python.

Imperative programming language is closely tied to the von Neumann architecture, where programs are executed step by step in a sequence.

In von Neumann architecture, CPU, memory and I/O devices are seperated and connected through some wires known as BUS. Instructions are stored in memory, its transfer to the CPU first. then CPU executed the instruction. As instruction transfered and executed one after another in a sequence, this architecture and Imperative programming

are the same concept.



von Neumann architecture use a single BUS for all component, that's why sometimes CPU ~~re~~ stay on idle state. Specially when its read data from I/O.

Ans. to the ques.no.02

'If' statement is used to execute a block of code for one time only. But 'while' statement used to ~~run~~ execute a block multiple time as long as condition is true.

Task, that performed by if statement, can also be done by using while statement, but its not

convenient, also readability will not be good.

As, we can implement using 'while' statement, so 'if' statement is not mandetory. But we need to do some extra line of code.

Example:

```
int main()
{
    int x = 5;
    if (x > 0){
        printf ("x is positive");
    }
}
```

using only 'while' statement:
with the help of 'break'.

```
int main()
{
    int x = 5;
    while (x > 0){
        printf ("x is positive");
        break;
    }
}
```

⇒ if we want to use only the while loop:

```
int main()
{
    int x = 5;
    bool condition = (x > 0);
    while (condition){
        printf ("x is positive");
        condition = False;
    }
}
```

As well as we can also implement the "if-else" statement by using two "while" statement. Therefore, 'if' statement is not mandetory in c but its very convenient to use and provide good readability.

Ans. to the ques. no. 03

Language design criteria'.

(i) Generality:

Generality means to avoid special cases in the availability or use of construct. So, we need to combine closely related construct into a more general one.

Example:

- in 'c' nested if-else allowed but nested function definition is not allowed.
- in 'c' "==" used to compare equality. But two array can't be compared using this operator.

(ii) Uniformity:

Uniformity refers to consistency in appearance, and behavior of language constructs. When similar things in a language behave in a similar way, the language is uniform.

Example: lack of uniformity

- in "C++" a semicolone is necessary after a class definition, but it is not allowed for function definition.

```
int function()
{
   ---
   ---
}
```

```
class class_name
{
   ---
   ---
};
```

(iii) Extensibility:

There should be some general mechanism to add new features to a language by the users or programmers.

Example:
- define of new data type

- creating new libraries.
- adding function to library.
- Specially 'Python' PL provides these flexibility.

(iv) Restrictability:

Language design should make it possible for a programmer to be able to write program with minimum knowledge of the language or language constructs. Complex features should be generalized.

Example:

- 'C' PL is little bit hard to learn for too much syntax restriction. most of the cases, the reason for a failed program is a missing semicolon.
- 'Python' is very easy to learn as there is no use of complex syntax. or semicolon.

If we evaluate 'Python' programming language according to these four design criteria, Python is a good programming language that has all the four criteria.

Ans. to the ques no. 04

There are ~~normally three~~ many types of error occur in a program. These are:

① Syntax Error:

Usually happen in 'C' PL, new programmers frequently forget about the semiclone and the program failed to build. Some-times we may mis some operator.

Example:

```
int main()
{
    int a = 10;
    int b = 20          → missing ;
    int c = 30;
    printf("sume:", a+b+c);
}
```

```
int main()
{
    int a = 10;
    int b = 20;
    if (a = b){          → comparison operator missing
        printf("Equal");
    }
}
```

(ii) Run time error:

Run time errors occur during the execution of a program. It's happen due to invalid input, often devided by zero. That's whr we need some exception handling mechanism in language design to control the error during run time and avoid unexpected crashes.

Example:

```
Read two number a,b;
function_devide (a,b){
    return a/b;
}
```

if user provide "0" as input ~~of b~~ of b, then it will be a run time error.

(iii) Logical Error:

This error may happen in the constructions of algorithm.

Example:

```
function factorial (n)
    if n==0
        return 1
    else
        return n* factorial (n-1)
```

infinite recursion if n is negative.

(iv) Type error: Occurs in function call.

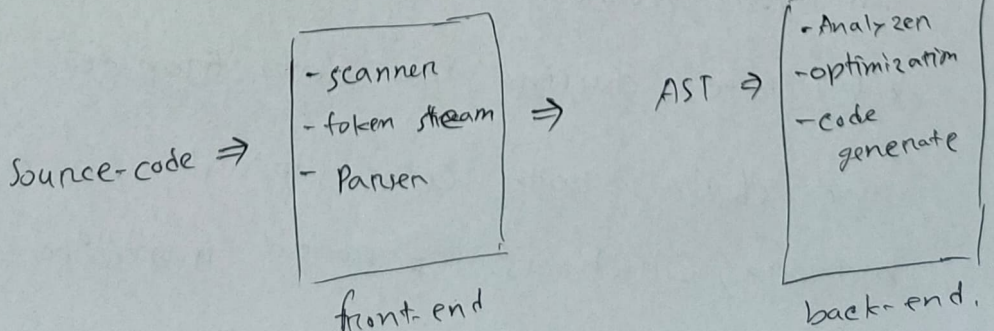
```
int function_addition (int a, int b)
  {
     return a+b;
  }

int main() {
{
   - - -
   - - -
   - -
   function_addition (5, 7.5);
   return 0;
}
```

type error

Ans. to the ques. no. 05

a) Compilers separated into front-end and back-end.



Front end of a compiler scan the source code for any syntax error using the regular grammer and context free gramer. Then build an abstract syntax tree (AST).

Back-end start from the input as AST, start analysis for optimization. Build TAC as followed by intermediate code generator. Then the executable machine code.

For this separation, front end part now can be use for multiple platforms. And back-end can be customized according to the hardware architecture. Some times, different front end share the same back-end constructs.

(b)

In von Neumann architecture, CPU stay on ~~2nd~~ idle state during the data transfer. Instruction and data both are stored on a single memory, which is seperated from the CPU and

connected through a BUS. Memory access time is much slower than CPU process time, so and CPU can access one memory location at a time. That's why, CPU stay on idle state during data transfer.

(c)

Symbol table contains the ~~list of~~ lebels of ~~this~~ source file and static data that can be reffered. It helps to perform some task like name resolution, type checking, code generation, error detection.

(d)

Portability of a programming language refers to the ability of program written in a specific language to be compiled and executed on different hardware and software platform without significant modifications.

Example: Java.

Ans. to the ques. no. 06

For designing a program language, I would prefer static scoping strategy. Because it improves the readability, and maintability. As programmers need to spend much time in debugging, static scoping will be esier for debug. That's why, static scoping is good for programmers.

Output of the given code:

Static Scoping:

6 → not available in the current block, value taken from the global declaration
7 → declared in the same block
8 → declared in the same block
5 → declared in the same block
8 → declared in the parent block
7 → declared in the same block
6 → declared in the global

Dynamic Scoping:

6
7
8
5
8
7
6

Here, each variable is declared in a nested block. In dynamic scoping, if the value is not present in the current block, then it check the invocation point of that block. That's why both output are the same.

Ans to the ques. no. 07

Short-code: Short-code first emerged in 1949. Instructions were written as sequence of 1's and 0's. It was the one of the earliest attempts to create a higher level programming environment.

Advantage:

- simplified compared to direct machine code

Limitation:

- hardware specific
- limited portability
- difficult to maintain

Assembly Language: Mnemonic symbols are used for instructions codes and memory locations. There was an assembler translator, which translates the symbolic assembly language code to binary machine code.

Advantage:

- direct control over hardware
- efficient use of resources.
- fast execution.

~~Disa~~

Limitation:

- complexity level high
- difficult to maintain

Fortran: General purpose, high-level programming language designed for scientific computation. Early versions were close to assembly.

Advantage:
- highly efficient language for scientific computation.

ALGOL 58-60: First general purpose language designed for scientific and engineering applications.

Advantage:
- introduce the concept of data type, compound statement, identifiers length, any dimension array, nested statement.
- orthogonality.

Disadvantage:
- Too much orthogonality

C/C++: 'C' is general purpose high-level language, best for efficiency, and used in system level most. 'C++' is the extension version of 'C' with object oriented concept.

Advantage:
- efficient and portable.

Limitation:
- lack of built in garbage collection feature.

Java: General purpose, object-oriented programming language designed for platform independence.

Advantage:
- write once, run everywhere
- built-in memory management

Limitation:
- Little bit slower due to interpreted nature.

Python: General purpose, high-level programming language, that can be used in wide range.

Advantage:
- Good readability, simplicity, easy to learn.
- good for data analysis, AI
- extensive standard library
- lots of flexibility to the programers.

Limitation:
- Slower than compiled language.

LZSP/Scheme: It is known as the List processing language, which is one of the oldest high level language and second best programming language designed for artificial intelligence. The only data type was 'LZST'.

Advantage:
- powerful for functional program & AZ.
- easy to learn

Limitations:
- hardware specific.
- Hardware decendent this language.
- slower compared to the modern language.