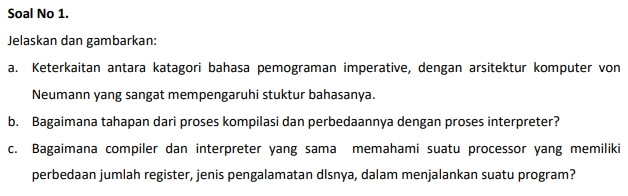
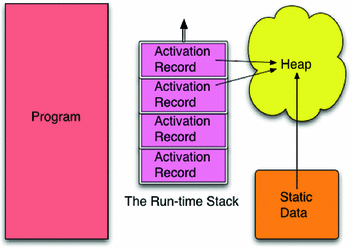
1. **KASUS (100 %)**

**Kombinasi B (1, 3, 5, 6, 8, 9, 10)**



**Answer :**

1. The programs grew in size it was necessary to provide the means for applying additional structure to them. In the early days a function was often called a sub-routine. Functions, procedures, and sub-routines were introduced by languages like Algol 60 so that programs could be decomposed into simpler sub-programs, providing a way for programmers to organize their code. Terms like top-down or bottom-up design were used to describe this process of subdividing programs into simpler sub-programs. This process of subdividing programs was often called **structured programming,** referring to the decomposition of programs into simpler, more manageable pieces. Most modern languages provide the means to decompose problems into simpler subproblems. **We often refer to this structured approach as the imperative model of programming**.



**Linkage Between Imperative model & Von Neumannc**

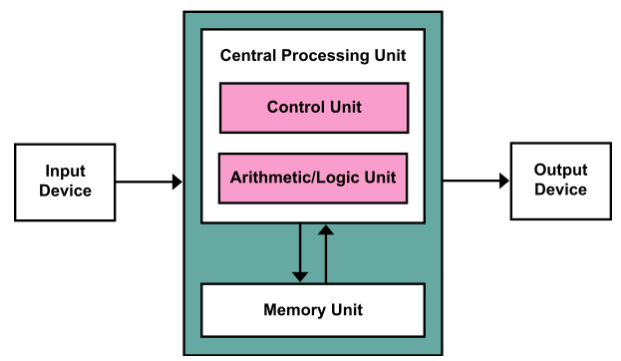
To implement functions and function calls in the von Neumann architecture, it was necessary to apply some organization to the data of a program. In the imperative model, memory is divided into regions which hold the program and the data. The data area is further subdivided into the static or global data area, the run-time stack, and the heap.

In the late 1970s and 1980s people like Niklaus Wirth and Bjarne Stroustrup were interested in developing languages that supported an additional level of organization called Object-Oriented Programming, often abbreviated OOP. Object-oriented programming still uses the imperative model of programming. The addition of a means to describe classes of objects gives programmers another way of organizing their code into functions that are related to a particular type of object.

When a program executes it uses a special register called the stack pointer (SP) to point to the top activation record on the run-time stack. The run-time stack contains one activation record for each function or procedure invocation that is currently unfinished in the program. The top activation record corresponds to the current function invocation. When a function call is made an activation record is pushed onto the run-time stack. When a function returns, the activation record is popped by decrementing the stack pointer to point to the previous activation record.

An activation record contains information about its associated function. The local variables of the function are stored there. The program counter’s value before the function call was made is stored there. This is often called the return address. Other state information may also be stored there depending on the language and the details of the underlying **von Neumann architecture**. For instance, parameters passed to the function may also be stored there.

Like the original von Neumann architecture, the primary goal of the imperative model is to get data as input, transform it via updates to memory, and then produce output based on this imperatively changed data. The imperative model of computation parallels the underlying von Neumann architecture and is used by many modern languages. Some variation of this model is used by languages like Algol 60, C++, C, Java, VB.net, Python, and many other languages.



1. The stages of the compilation process are:
2. **Lexical Analysis**

* The lexical analyzer is also called a lexer or a scanner. The lexer receives a stream of characters from the source program. It groups them into tokens. Source Program Stream of Tokens Lexical Analyzer.

1. **Syntax Analysis**

* A syntax analyzer is called a parser. The parser receives a stream of tokens from the lexer. It groups them into phrases that match grammatical patterns. Stream of Tokens Abstract Syntax Tree Parser.

1. **Syntactic Analysis**

* The parser outputs an abstract syntax tree representing the syntactical structure of the pattern.

1. **Intermediate Code Generation**

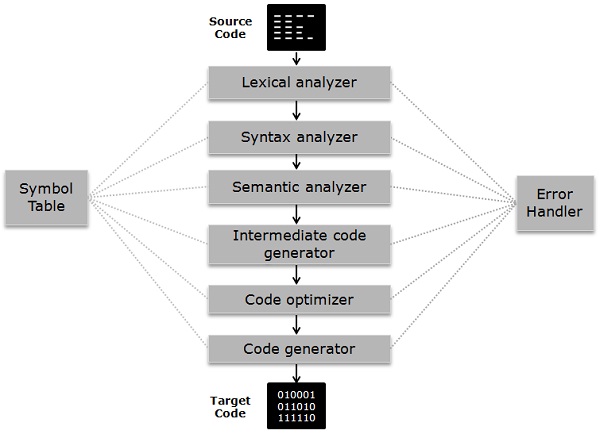
* Intermediate code is code that represents the semantics of a program, but is machine- independent. This marks the boundary between the front end and the back end. The front end is language-specific and machine-independent. The back end is language-independent and machine-specific.

1. **Optimization**

* A program may be optimized for speed or for size. An optimizer reviews the code, looking for ways to reduce the number of operations and the memory requirements.

1. **Machine Code Generation**

* The code generator receives the (optimized) intermediate code. It produces either machine code for a specific machine, or assembly code for a specific machine and assembler. If it produces assembly code, then an assembler is used to produce the machine code.

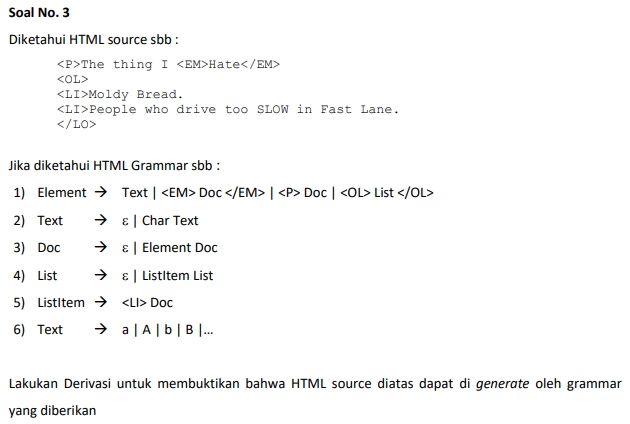


|  |  |  |
| --- | --- | --- |
| BASIS FOR COMPARISON | COMPILER | INTEPRETER |
| INPUT | It takes an entire program at a time | It takes a single line of code or instruction at a time. |
| OUTPUT | It generates intermediate object code. | It does not produce any intermediate object code. |
| WORKING MECHANISM | The compilation is done before execution. | Compilation and execution take place simultaneously. |
| SPEED | Comparatively faster | Slower |
| MEMORY | Memory requirement is more due to the creation of object code. | It requires less memory as it does not create intermediate object code. |

**Compiler and interpreter both are intended to do the same work but differ in operating procedure**, Compiler takes source code in an aggregated way whereas Interpreter takes constituent parts of source code, i.e., statement by statement.

Although both compiler and interpreter have certain advantages and disadvantages like Interpreted languages are considered as cross-platform, i.e., the code is portable. It also doesn’t need to compile instruction previously unlike compiler which is time-saving. Compiled languages are faster regarding the compilation process.

1. Every processor or processor family has its own instruction set. Instructions are patterns of bits that by physical design correspond to different commands to the machine. Thus, the instruction set is specific to a class of processors using (mostly) the same architecture. Successor or derivative processor designs often include all the instructions of a predecessor and may add additional instructions. Occasionally, a successor design will discontinue or alter the meaning of some instruction code affecting code compatibility to some extent; even nearly completely compatible processors may show slightly different behavior for some instructions, but this is rarely a problem. Systems may also differ in other details, such as memory arrangement, operating systems, or peripheral devices. Because a program normally relies on such factors, different systems will typically not run the same machine code, even when the same type of processor is used. Interpreter directly executes the instructions in the source programming language while a compiler translates those instructions into efficient machine code. An interpreter will typically generate an efficient intermediate representation and immediately evaluate it.



**Answer :**

Element \rightarrow <P> Doc

Doc \rightarrow <P> Element Doc

Element \rightarrow <P> The thing I Doc

Doc \rightarrow <P> The thing I Element Doc

Element  \rightarrow  <P> The thing I <EM>Hate</EM> Doc

Doc   \rightarrow <P> The thing I <EM>Hate</EM> Element Doc

Element \rightarrow   <P> The thing I <EM>Hate</EM> <OL> List </OL>

List   \rightarrow <P> The thing I <EM>Hate</EM> <OL> ListItem List </OL>

ListItem \rightarrow <P> The thing I <EM>Hate</EM> <OL> <LI> Doc List </OL.>

Doc   \rightarrow  <P> The thing I <EM>Hate</EM> <OL> <LI> Element Doc List </OL.>

Element \rightarrow <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. Doc List </OL.>

Doc \rightarrow <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. Doc List </OL.>

List \rightarrow   <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. Doc ListItem List</OL.>

ListItem   \rightarrow  <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. Doc <LI> Doc List<OL>.

Element\rightarrow  <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. Doc <LI> Element Doc List<OL>

Element\rightarrow  <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. Doc <LI> People who drive too SLOW in fast Lane. Doc List<OL>

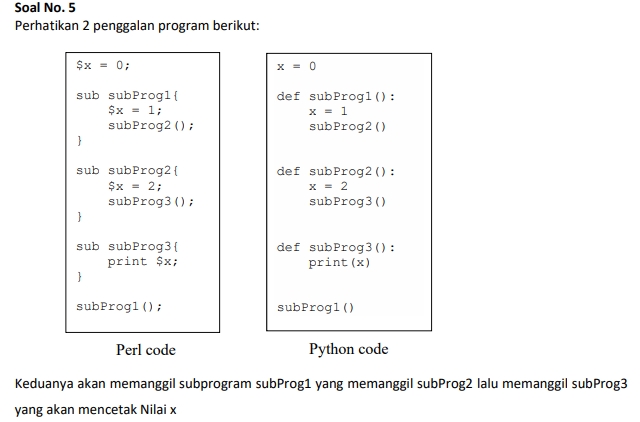
Doc \rightarrow Epsilon

Doc \rightarrow   <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. <LI> People who drive too SLOW in fast Lane. List<OL>

List \rightarrow Epsilon

LIst \rightarrow   <P> The thing I <EM>Hate</EM> <OL> <LI> Moldy Bread. <LI> People who drive too SLOW in fast Lane.<OL>

**Hence, the HTML source code is generated by grammar.**



1. Penggalan program yang mana yang memiliki dynamic scope dan yang mana yang memiliki static scope?

**Answer:**

**The Perl code** is the code fragment **that has global dynamic scoping**. As in dynamic scoping, the global variable refers to the data from the most recent environment or block.

The **Python's code fragment** **has the static scoping** as static scoping first looks up for the top level then the local block level.

1. Nilai x yang mana yang akan di cetak oleh Perl code?

**Answer:**

**The value printed will be 2**

As the global variable x will store the value of x from the most recent block of code.

So the value of x will change like :

**0(global)->1(subProg1)->2 (subProg2)**

1. Nilai x yang mana yang akan di cetak oleh Python code?

Answer:

**The value printed in the python program will be 0**

As it has static scoping and static scoping first looks up for the top-level environment than the local block level. Here global was available as **x=0** so it got printed.

1. Jelaskan tentang variable scope : static dan dynamic !

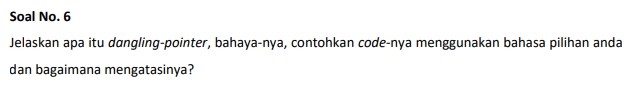
**Answer:**

**Static**

In static scoping the program the looks the variable in the top-level block(local block) if the variable is not available in the local block, it searches it in the global block, if found well and good otherwise throw an error.

**Dynamic**

In dynamic scoping, the global identifier stores the value from the most recent environment i.e. when the control passes from one function to another the value from the most recent block gets stored in the global identifier if declared.



**Answer:**

**A  pointer is pointing to non-existing memory location or unauthenticated address in memory is called dangling pointer.**

Example :

// dangling pointer

#include <stdlib.h>

#include <stdio.h>

int main()

{

    int \*ptr = (int \*)malloc(sizeof(int));

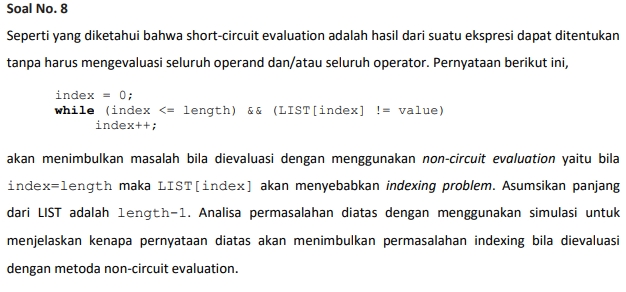
**// free(ptr) membuat jadi Dangling Pointer**

    free(ptr);

**// Cara Mengatasinya dengan ptr = NULL;**

    ptr = NULL;

}



**Answer :**

* Let us assume that **length of LIST is length – 1** and we **have non-circuit evaluation method**.
* Then let us simulate the loop from index = 0 to index = length - 1 (since, length is length -1, it means last index would be length -2)

**Iteration : index = 0**

both condition is satisfied. index <=length i.e.,(length - 1) and LIST[index] has some element that is compared for inequality to value

**Iteration : index = 1**

Again, both conditions satisfied, index <= length i.e.(length - 1), and LIST[index] has some value.

**Iteration : index = 2**

Again, both conditions satisfied, index <= length i.e.,(length - 1), and LIST[index] has some value.

**Iteration : index = 3**

Again, both conditions satisfied, index <= length i.e.,(length - 1), and LIST[index] has some value.

**.**

**.**

**.**

**Similarly we go on till**  
**Iteration : index = length - 2**

index < = length i.e., length-2 <= length -1 is True,

and LIST[index] also has some value as last index is length -2 as explained above.

**Now when,**

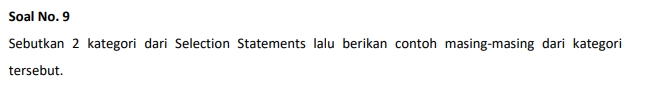
**index = length -1**

index = length - 1 then due to non-circuital evaluation, the next expression will also be evaluated **AND**

index = length-1 will cause indexing problems because no element exits at location length -1. As stated above last index of element is length - 2. So it will generate an error like array index is not there or out of bounds.

**But if there was short circuit, after the first expression is false, we do not have to evaluate the second expression**.

**So we wont reach till the second expression where LIST[index] value is compared with some other value for inequality. And it would not have thrown any error**.



Answer:

1. **Single selection statement or one way selector:**  
   Example of this is **if statement**

It selects or ignores a single action or single group of actions.

**Syntax:**  
if (condition){

//statements

}

**Example:**

if ( Age>= 60 ){

System.out.println( “Give pension” );

}

1. **Double selection statement or two way selector:**  
   Example of this is **if...else statement**  
     
   It performs a set of action if a particular condition is true and another set of action if particular condition is false.

**Syntax:**  
if (condition){

//statements

}  
else{

//statements

}

**Example:**

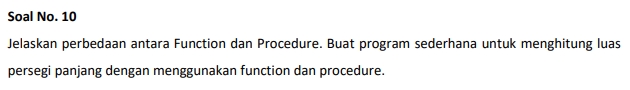
if (Age>= 60){

System.out.println( “Give pension” );

}  
else{

//statem System.out.println( “Do not Give pension” );ents

}



Answer:

|  |  |
| --- | --- |
| Function | Procedures |
| Have return type and return value | Return values using parameters. |
| Doesn't allow output parameters | Procedure allows both input & output parameters. |
| Cannot Manage Transaction inside function | Can Manage transaction inside procedure |
| Cannot call stored procedures from a function | Can call a function from stored procedure |
| Call a function using select statement | Cannot call a procedure using select statement |
| Deals with an expression | Doesn't deal with an expression |
| Doesn't support try-catch block | Support try-catch block |
| Can call through SQL Queries | Can't be call through SQL Queries |

**Sample Java Program to find area of Rectangle using function:**

**import** java.util.Scanner;

**public** **class** Area {

**public** **static** **void** main(String[] args) {

**int** length,width;

Scanner scan = **new** Scanner(System.***in***);

length = scan.nextInt();

width = scan.nextInt();

**int** area = *getArea*(length,width);

System.***out***.println(area);

}

**public** **static** **int** getArea(**int** l,**int** w){

**return** l\*w;

}

}

**Sample Java Program to find area of Rectangle using Procedures:**

**import** java.util.Scanner;

**public** **class** Area {

**public** **static** **void** main(String[] args) {

**int** length,width;

Scanner scan = **new** Scanner(System.***in***);

length = scan.nextInt();

width = scan.nextInt();

**int** area = length\*width;

System.***out***.println(area);

}

}