QUESTÃO 1

As linguagens de baixo nível são aquelas que mais se aproximam da forma com que o computador representa dados e instruções. Geralmente, cada ordem dada nestas linguagens representa uma instrução executada pelo microcomputador. Quando essa linguagem é utilizada a velocidade de execução dos programas é maior e o tamanho deles é menor. A linguagem de baixo nível mais conhecida é o Assembly.

As linguagens de alto nível são aquelas que mais se aproximam da linguagem humana, ou seja, muito mais próxima do programador do que do dispositivo. Essa é uma linguagem bem mais amigável ao programador devido à sua facilidade de entendimento. Um exemplo de linguagem de alto nível é a linguagem SDL (Specification Design Language).

Algumas da diferenças entre as linguagens de baixo e alto nível: Em alto nível é bem mais fácil de programar e dar manutenção; as linguagens estão dirigidas a solucionar problemas com o uso de estruturas dinâmicas de dados. Enquanto que em baixo nível os programas tem, normalmente, um performance melhor; mas a linguagem depende totalmente da máquina, ou seja, o mesmo programa não pode ser utilizado em outras máquinas.

O compilador é o programa que traduz o código de um programa escrito em uma linguagem de alto nível para um equivalente em linguagem de máquina. A tradução é lenta, mas a execução é rápida. O compilador é composto por várias fases: Análise léxica + Análise sintática + Análise semântica + Gerador de código.

QUESTÃO 2

Em perl

Modo de compilar/executar o programa: chmod +x questao2.pl
Linha de comando para executar o programa: ./questao2.pl
Código-fonte:

```
#!/usr/bin/perl

print "Digite N:\n";

my $n = <>;

my $res = 1.0;

my $i = 1;

for($i=1;$i<=$n;$i++){

        if($i%2 == 1){

                $res = $res*(((2*$i)-1)**((2*$i)-1))/((2*$i)**(2*$i));

        } else {

                $res = $res+(((2*$i)-1)**((2*$i)-1))/((2*$i)**(2*$i));

        }
        $shr=0;

        $j=0;
}

printf "%.15f",$res;
```

Em c:

Modo de compilar/executar o programa: chmod +x questao2.c
Linha de comando para executar o programa: ./questao2.c
Código-fonte:

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <math.h>


int main()
{
    int i,n;
    float soma=1;
    scanf("%d", &n);
    for(i=1;i<=n;i++){
        if(i%2)
            soma=soma*(float)pow(2*i-1,2*i-1)/pow(2*i,2*i);
        else
            soma=soma+(float)pow(2*i-1,2*i-1)/pow(2*i,2*i);
    }
    printf("%f\n",soma);
    return 0;
}
```

Em c ++

Modo de compilar/executar o programa: chmod +x questao2.cpp
Linha de comando para executar o programa: ./questao2.cpp
Código-fonte:

```
#include <iostream>
#include <math.h>


using namespace std;


int main()
```

```
{
    int i,n;
    float soma=1;
    cin >> n;
    for(i=1;i<=n;i++){
        if(i%2)
            soma=soma*(float)pow(2*i-1,2*i-1)/pow(2*i,2*i);
        else
            soma=soma+(float)pow(2*i-1,2*i-1)/pow(2*i,2*i);}
    cout << soma << endl;
    return 0;
}
```

Em Python

Modo de compilar/executar o programa: python questao2.py
Linha de comando para executar o programa: ./questao2.cpp
Código-fonte:

```
n=int(input("entre com o valor de n: "))
soma=1
for i in range(1,(n+1)):
if(i%2==1):
soma=soma*(((2*i)-1)**((2*i)-1))/((2*i)**(2*i))
else:
soma=soma+(((2*i)-1)**((2*i)-1))/((2*i)**(2*i))
print(soma)
```

QUESTÃO 3

Em C

Modo de compilar/executar o programa: chmod +x questao3.c
Linha de comando para executar o programa: ./questao3.c
Código-fonte:

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

int main() {

char nomes[10][15], aux[15];
int quant, i, j;

printf("Quantidade de nomes: ");
scanf("%d", &quant);

for(i=0; i<quant; i++)
	scanf("%s", nomes[i]);

for(i=0; i<quant; i++) {
	for(j=0; j<quant; j++) {
		if(strcmp(nomes[i], nomes[j]) < 0) {
		strcpy(aux, nomes[i]);
		strcpy(nomes[i], nomes[j]);
		strcpy(nomes[j], aux);
		}
	}
}


for(i=0; i<2; i++)
	printf("%s\n", nomes[i]);

return 0;
}
```

Em C++

Modo de compilar/executar o programa: chmod +x questao3.cpp
Linha de comando para executar o programa: ./questao3.cpp
Código-fonte:

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

using namespace std;

int main() {

char nomes[10][15], aux[15];
int quant, i, j;

cout << "Quantidade de nomes: ";
cin >> quant;

for(i=0; i<quant; i++)
	cin >> nomes[i];

for(i=0; i<quant; i++) {
	for(j=0; j<quant; j++) {
		if(strcmp(nomes[i], nomes[j]) < 0) {
		strcpy(aux, nomes[i]);
		strcpy(nomes[i], nomes[j]);
		strcpy(nomes[j], aux);
		}
	}
}


for(i=0; i<2; i++)
	cout << "\n" << nomes[i];
	return 0;
}
```

Em Python

Modo de compilar/executar o programa: python questao3.py
Linha de comando para executar o programa: ./questao3.cpp
Código-fonte:

```
i=0
tam=int(input("Quantidade de nomes: "))
nomes=[]
for i in range(tam):
	nome=str(input())
	nomes.append(nome)
nomes.sort()
print ("\n")
print (nomes[0])
```

```
print (nomes[1])
```

Em perl

Modo de compilar/executar o programa: chmod +x questao3.pl
Linha de comando para executar o programa: ./questao3.pl
Código-fonte:

```
#!/usr/bin/perl


print 'tamanho'

$tam = <STDIN>;

for ($i = 0; $i < $tam; ++$i)

        {

          $nome= <STDIN>;

             push(@nomes, $nome);

        }

my @sorted = sort{ lc($a) cmp lc($b) } @nomes;


print $nomes[0];

print $nomes[1];
```

QUESTÃO 4

Modo de compilar/executar o programa: chmod +x questao4.sh
Linha de comando para executar o programa: ./questao4.sh
Código-fonte:

```
#!/bin/bash

echo "Digite url"

read url

wget -r -A jpeg,jpg,png $url
```

QUESTÃO 5

(A) Imperativas

São linguagens que expressam sequências de comandos que realizam transformações sobre dados. É baseado na arquitetura de Von Neumann. É o primeiro paradigma a existir e até hoje é o dominante. Esse paradigma segue o conceito de um estado e de ações que manipulam esse estado, nele encontramos procedimentos que servem de mecanismos de estruturação. As vantagens desse paradigma são a eficiência, muito flexível e é bem estabelecido. A desvantagem é a difícil legibilidade.

Alguns exemplos de linguagem que baseiam-se no modelo imperativo: C, Java, Perl etc.

(B) Funcionais

Este paradigma trata a computação como uma avaliação de funções matemáticas. Dá ênfase em valores computados por funções. A desvantagem de usar esse tipo de paradigma é que parecem faltar diversas construções freqüentemente consideradas essenciais em linguagens imperativas. A vantagem é que a recursividade em programação funcional pode assumir várias formas e é em geral uma técnica mais poderosa que o uso de laços do paradigma imperativo.

Exemplos de linguagens funcionais são: Haskell e LISP.

(C) Lógico

Nesse paradigma programas são relações entre Entrada/Saída. Possui estilo declarativo, como o paradigma funcional. Inclui características imperativas, por questões de eficiência. Dá-se ênfase em axiomas lógicos. Possui a princípio todas as vantagens do paradigma funcional. A desvantagem é que as variáveis não possuem tipos.

Exemplos de linguagens lógicas : Prolog e Planner.

(D) Marcação / Híbrida

É representada por um conjunto de códigos empregados a dados e a textos, com a finalidade de adicionar informações específicas sobre esse dado ou texto. É um paradigma novo, não especificam computação e são usadas para definir o layout de informação em documentos Web.

Exemplos de marcação/programação híbrida (linguagens de marcação estendida para suportar alguma programação): JSTL e XSLT.

QUESTÃO 6

<expr> → <expo> * <expr> | <expo>

<expo>→ <base> ^ <expo> | <base>

<base>→ A | B | C

QUESTÃO 7

Desde o surgimento da primeira linguagem de programação de alto nível, Fortran, na década de 1950, uma grande variedade de linguagens de programação tem sido proposta, como consequência de domínios de aplicação distintos, avanços tecnológicos e interesses comerciais, dentre outros aspectos. Algumas linguagens compartilham características em comum e são ditas pertencerem a um mesmo paradigma: um modelo, padrão ou estilo de programação. A classificação de linguagens em paradigmas é uma consequência de decisões de projeto que impactam radicalmente a forma na qual uma aplicação real é modelada do ponto de vista computacional.

QUESTÃO 8

O analisador léxico (scanner) é a parte do compilador responsável por ler caracteres do programa fonte e transformá-los em uma representação conveniente para o analisador sintático. O analisador léxico lê o programa fonte caractere a caractere, agrupando os caracteres lidos para formar os símbolos básicos (tokens) da linguagem.

<program> ::= program p

<declaration_sequence>

begin

<statements_sequence>

end ;

<declaration_sequence>::=<expression.> |<declaration_sequence>

<expression>::= <var>{< | > | <= | >= | == | !=}<var> | <var>

<var>::= A|B|C|

<statements_sequence> ::= <staments> | <while>

<while> ::= while (<conditions>) <statements>

<statements> ::= [{]<statement> [}] | { <statement> <statement>*

<conditions> ::= [!]<condition { {&& | ||} [!]<condition>}*

<condition> ::= [!]expression {< | > | <= | >= | == | !=} [!]<expression>}*

<expression>::= <var>{< | > | <= | >= | == | !=}<var> | <var>

<var>::= A|B|C|

QUESTÃO 9

Analise semântica é a área de estudo de ciência da computação que se preocupa em especificar o significado (ou comportamento) de programas de computador e partes de hardware. Um dos objetivos é analisar um programa a partir somente da sintaxe da linguagem, sem necessariamente ter que executá-la. Ela descreve o significado das expressões, das instruções e das unidades de programas.

Semântica operacional tem como objetivo descrever como uma computação é processada. Descreve o significado de um programa pela execução de suas instruções em uma máquina, seja ela real ou simulada. As mudanças no estado da máquina (memória, registradores, etc.) definem o significado da instrução. Alguns problemas de se utilizar a semântica operacional: armazenamento no computador real é grande e complexo; não são usados na semântica operacional formal; usam linguagens intermediárias e interpretadores.

A semântica axiomática foi definida em conjunto com o desenvolvimento de um método para provar a exatidão dos programas que mostra a computação descrita por sua especificação, quando pode ser construída. Em uma prova, cada instrução de um programa tanto é precedida como seguida de uma expressão lógica (asserções) que especifica restrições a variáveis. Asserções que precedem uma instrução são chamadas de pré-condição. Asserções que seguem imediatamente uma instrução são chamadas de pós-condição. É baseada em Lógica Matemática. Possui abordagem mais abstrata. Não especifica diretamente o significado, mas sim o que pode ser provado sobre o programa.

Semântica denotacional é baseado na teoria de funções recursivas. O método mais abstrato de descrição da semântica. A ideia baseia-se no fato de que há maneiras rigorosas de manipular objetos matemáticos, mas não construções de linguagens de programação. O processo de construção da especificação denotacional para uma linguagem define um objeto matemático para cada entidade da linguagem. Define uma função que mapeia instâncias das entidades da linguagem em instâncias dos correspondentes objetos matemáticos.

QUESTÃO 10

a)

122*y -144>144
122*y>144+144
122*y>288
y>288 / 122

b)

y=5*x-5
y+5<45
y<40
5*x-5<40
5*x<45
x<9

c)
y>2
y+2>2
y>0
y=y-2
y-2>2
y>4

d)
para zero iterações
{i=N}
wp=(i=i+1, {i=N}) = {i+1=N}, ou {i=N-1}
ou seja,
{i<=N}

e)
wp=(sn=sn+a, sn=n*a):
n*a=sn+a
n=sn/a + a/a
n= sn/a +1
n-1=sn/a
a(n-1)=sn
{a=sn/(n-1)}

JAQUELINE MIRANDA NEVES