1) Conceitue e descreva as diferenças entre (i) linguagem de alto-nível e (ii) linguagem de baixo-nível. Descreva o papel e a importância do compilador no processo de criação de programas de computador.

Linguagem de alto nível são as linguagens que possuem uma estrutura e palavras-chave que são mais próximas da linguagem humana. Tornando os programas mais fáceis de serem lidos e escritos. Esta é a sua principal vantagem sobre as linguagens de nível mais baixo. Alguns exemplos de linguagens de alto nível são: JavaScript, Python, Pho, Ruby.

Já a Linguagem de baixo nível são linguagens cujas instruções correspondem quase que diretamente ao código de máquina que será enviado ao processador para execução. Assim, utiliza somente instruções do processador, para isso é necessário conhecer os registradores da máquina. Nesse sentido, as linguagens de baixo nível estão diretamente relacionadas com a arquitetura do computador. Um exemplo é a linguagem Assembly que trabalha diretamente com os registradores do processador, manipulando dados.

A compilação é um método de implementação que traduz programas de alto nível para linguagem de máquina, a qual pode ser executada diretamente no computador, e possui a vantagem de ter uma execução de programas muito mais rápida, uma vez que o processo de tradução estiver completo. A escolha do compilador deve levar em consideração os seguintes itens: A linguagem de alto nível que se está trabalhando; Os requisitos do problema (embarcado, tempo real, concorrente, etc; e a qualidade do código gerado. A linguagem que um compilador traduz é chamada de linguagem fonte e a compilação é composta por várias fases: Análise Léxica, Análise Sintática, Análise Semântica e Gerador de Código.

2) Faça um programa escrito na linguagem C, C++, Python e Perl que aceite como entrada um número inteiro n maior ou igual a 1 e retorne como saída o valor da série:

$$\frac{1^1}{2^2} + \frac{3^3}{4^4} * \frac{5^5}{6^6} + \frac{7^7}{8^8} * \ldots ? \frac{(2n-1)^{(2n-1)}}{(2n)^{(2n)}}$$

? = + se n é par

? = * se n é ímpar

Para a compilação e execução dos programas feitos em **C**, **C++** e **Perl** foi utilizado o Xcode versão 8.3.1

Já para a compilação e execução do programa feito em **Python** foi utilizado a IDLE versão 3.5.2 e o Python versão 3.5.2 também.

Valor de Entrada	Valor de Saída C, C++, Perl, Python
n = 1	0.250000
n = 2	0.355469
n = 3	0.023809
n = 4	0.072896

3) Faça um programa escrito na linguagem C, C++, Python e Perl que leia uma quantidade indeterminada de strings na entrada a partir de um arquivo e identifique quais são as duas strings que, respectivamente, são o primeiro e o segundo na ordenação lexicográfica. Exemplo:

Entrada: marcus ricardo romulo brauliro jorge marcelo ana fabrício **Saída**: ana, brauliro

Já para a compilação e execução do programa feito em **Python** foi utilizado a IDLE versão 3.5.2 e o Python versão 3.5.2 também.

5) Descreva as seguintes categorias de linguagens de programação e apresente o nome de duas linguagens de programação com seus respectivos exemplos.
(A) Imperativas: variáveis, atribuição e interação. Inclui programação orientada a objeto, scripting e visuais. Exemplos: C, Java, Perl, JavaScript, Visual BASIC, C++, Pascal.
(B) Funcionais: aplicação de funções sob parâmetros, enfatiza a avaliação de expressões, ao invés da execução de comandos. Exemplos: LISP, Scheme, Haskell, Ocaml.
(C) Lógicas: baseada em fatos e regras. Exemplo: Prolog.
(D) Marcação/Híbrida: linguagens de marcação estendida para suportar alguma programação. Exemplo JSTL, XSLT.

6) Escreva uma gramática no formato BNF e não ambígua para expressões envolvendo as variáveis A, B, C e os operadores * (multiplicação) e ^ (exponenciação). A precedência deve seguir as regras usuais da matemática.
<assing> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> * <term>
| <term>
<term> → <term> ^ <factor>
| <factor>
<factor> → (<expr>)
| <id>

7) Descreva o que é um paradigma de programação.

Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns. Paradigmas diferem nos conceitos e abstrações utilizadas para representar os elementos de um programa (como objetos, funções, variáveis, restrições, etc) e as etapas que compõem um cálculo (atribuição, avaliação, continuações, fluxos de dados, etc). A classificação de linguagens em paradigmas é uma consequência de decisões de projeto que tem impacto na forma segundo a qual uma aplicação real é modelada do ponto de vista computacional.

9) Defina análise semântica e descreva: Semântica Operacional; Semântica Axiomática; e Semântica Denotacional.

Análise Semântica: trata da análise do significado das expressões, das instruções e das unidades de programa. É importante pois fica igualmente encarregada de analisar a utilização dos identificadores e de ligar cada uma delas a sua declaração. Nesta situação verificar-se que o programa respeita as regras de visibilidade e de porte dos identificadores. Também é esperado que no processo da compilação verifique que cada expressão definida tenha um tipo adequado conforme as regras próprias da linguagem.

Semântica Operacional: a ideia é descrever o significado de uma sentença ou programa pela especificação dos efeitos e rodá-lo em uma máquina.

Semântica Axiomática: é baseada em lógica matemática, possui uma abordagem mais abstrata para a especificação de semântica. Ela especifica o que pode ser provado sobre o programa.

Semântica Denotacional: é o método mais rigoroso e mais conhecido para a descrição do significado de programas. Ela é solidamente baseada na teoria de funções recursivas.

10) Utilizando a semântica axiomática apresente a pré-condição para os seguintes programas.

PROGRAMA – A x = 122 * y - 144 **{x > 144}**	122*y - 144 > 144 122*y > 144 + 144 122*y > 288 y > 288/122 {y > 2.36}

PROGRAMA – B y = 5 * x - 5 x = y + 5 **{x < 45}**	y + 5 < 45 y < 40 5*x – 5 < 40 5*x < 45 {x < 9}

PROGRAMA – C
if (x < 200)
y = y + 2;
else
y = y – 2;
{y > 2}
	y = y + 2 {y > 2}, produzirá y > 0
	
	y = y - 2 {y > 2}, produzirá y > 4
	
	Logo: {y > 4}

PROGRAMA – D while i < N do i = i + 1 end **{i == N}**	A pré-condição deve garantir que a pós-condição seja atendia, para isso existem duas opções: 1) executar o laço de repetição “while” e; 2) não executar o laço de repetição “while”, resultando diretamente na pós-condição. Onde para que a opção 1) ocorra é necessário que {i < N} e para que ocorra a opção 2) basta que {i = N}, já que a opção {i > N} gera incoerência com a pós-condição. Portanto, temos que: {i <= N}

PROGRAMA – E
i = 1; sn = 0; n = 32767;
while (i <= n){
sn = sn + a;
i = i + 1;
}
{sn == n * a}
	A pré-condição deve garantir que a pós-condição seja atendia, entretanto temos os valores de i, sn e n já são dados na questão, o que nos mostra que o laço de repetição “while” será sempre executado pois a condição (i<= N) será sempre verdadeira. Logo, se testarmos o laço de repetição tomando para “a” tanto para valores negativos quanto positivos e incluindo zero, a pós-condição será confirmada.
	Portanto, temos que: {a ∈ R }