問2　コンパイラの処理内容に関する次の記述を読んで，設問1～4に答えよ。

コンパイラとは，プログラム言語で記述された原始プログラムを翻訳して目的プログラムを生成するためのソフトウェアである。コンパイラの処理過程において，構文解析は字句解析が出力する字句を読み込みながら構文木を生成し，その字句の列が文法で許されているかどうかを解析する。

設問1　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

2項演算子から成る式の構文は，2分木で表現される。2分木から目的プログラムを生成するには，2分木を深さ優先で探索しながら，帰り掛けに節の演算子を評価する。式の構文木と探索順序の例を図に示す。

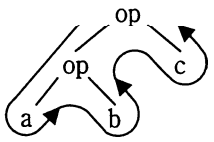


図　構文木と探索順序の例

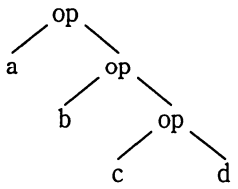
探索の結果，図の構文木の例は，aとbに対して演算opを施し，その結果とcに対して演算opを施すと解釈される。

括弧を含む式では，演算の優先度は，括弧内の優先度が高く，それ以外は左から右に式を評価する。このとき，次の式に対する構文木は［　　］である。

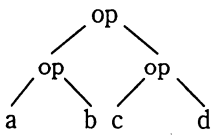
式：　a op (b op c) op d

解答群

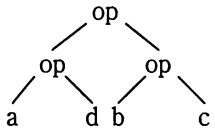
ア



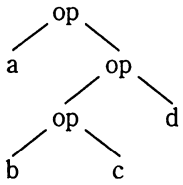
イ



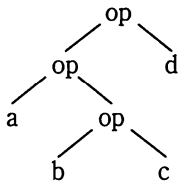
ウ



エ



オ



設問2　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

プログラム言語の文法は，構文規則で規定される。式の構文規則では式の構文を規定しているだけでなく，演算子の優先順位や結合規則も規定している。例として，優先順位が異なる演算子 op1，op2 と括弧を用いた式の構文規則を考える。

〔式の構文規則〕

(1)　式　→　項 | 式 op1 項

(2)　項　→　因子 | 項 op2 因子

(3)　因子 →　名前 | (式)

(4)　名前 →　a | b | c | d | e

“→”は，左側の構文要素が右側で定義されることを示す。

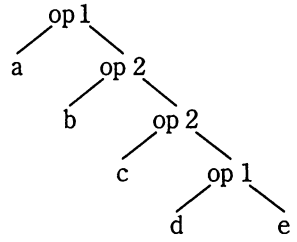
“|”は，“又は”を意味する。

深さ優先で探索すると仮定すれば，与えられた式に含まれる演算子op1とop2の演算順序は，〔式の構文規則〕から生成可能な構文木で表現できる。例えば，次の式に対して〔式の構文規則〕を適用した構文木は［　　］である。

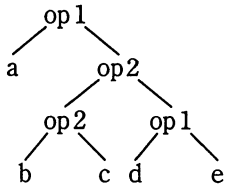
式：　a op1 b op2 c op2 (d op1 e)

解答群

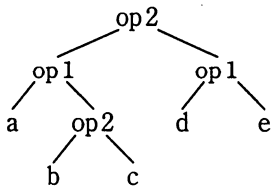
ア



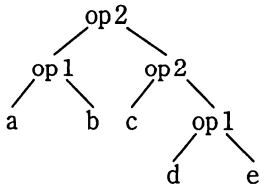
イ



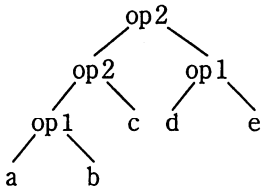
ウ



エ



オ



設問3　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

構文解析した結果は，構文木で表現する以外に，2分木と等価な表現の後置表記法（逆ポーランド表記法）で表現できる。後置表記法では，演算で使用する二つのオペランドを演算子の前に記述する。そして，後置表記法は，構文木を探索して得られる演算順序を表現したものであると考えることができる。例えば，設問1の図の例で，演算子opを加算＋とした場合，後置表記法ではａｂ＋ｃ＋となる。これは，aとbを加算し，その結果とcを加算することを表している。

後置表記法を用いて式ａ×ｂ＋ｃ×ｄ＋ｅを表現すると［　　］になる。ここで，加算＋と乗算×は，それぞれ設問2の演算子op1とop2に対応し，演算の優先順位や結合規則は，〔式の構文規則〕に従うものとする。

解答群

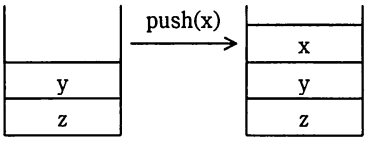
ア　ａｂｃｄ×＋ｅ＋　　イ　ａｂｃ×＋ｄ×ｅ＋

ウ　ａｂ×ｃｄ×＋ｅ＋　　エ　ａｂ×ｃ＋ｄ×ｅ＋

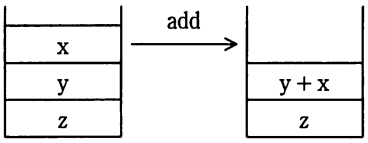
設問4　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

後置表記法で表現された式は，スタックを使って左から右に評価することができる。式 a + b + c × d を後置表記法に変換して評価する場合，スタックの操作順序は［　　］である。ここで，スタックを操作する命令として次の種類がある。また，加算 + と乗算 × は，それぞれ設問2の演算子 op1 と op2 に対応し，演算の優先順位や結合規則は，〔式の構文規則〕に従うものとする。

push(x)：スタックに x をプッシュする。



add：スタックからオペランドを二つポップして加算し，その結果をスタックにプッシュする。



mul：スタックからオペランドを二つポップして乗算し，その結果をスタックにプッシュする。

解答群

ア　push(a) → add → push(b) → add → push(c) → mul → push(d)

イ　push(a) → push(b) → add → push(c) → mul → push(d) → add

ウ　push(a) → push(b) → add → push(c) → push(d) → mul → add

エ　push(a) → push(b) → push(c) → push(d) → add → add → mul