問2　コンパイラの最適化（ソフトウェア）　(H24春·FE 午後問2)

【解答】

［設問1］　a－カ，b－キ
［設問2］　c－イ
［設問3］　d－イ，e－ウ

【解説】

コンパイラの最適化の方法，及び最適化と計算精度との関係に関する問題である。記述されている最適化の方法と具体的なプログラムの最適化過程を対応させていけば解答は難しくない。また，最適化実施の有無により浮動小数点数演算の結果が異なる例で，精度に関する理解度も問われているが，難易度の高いものではない。

コンパイラは，次の過程によって，プログラム言語で記述された原始プログラムを目的プログラムに翻訳する。コンパイラの最適化とは，原始プログラムを翻訳する過程で，プログラムの機能を変えることなく，プログラムの構造を変換することである。その目的は，問題の記述にあるように「プログラムの実行時間の短縮」である。また，ほかにも「プログラムのサイズの削減」という目的もある。

コンパイラの処理手順

原始プログラム→［字句解析→構文解析→意味解析→コード最適化→コード生成］→目的プログラム

最適化の方法の例として，七つの方法が示されている。実際には，プログラム言語やマシン環境によってその種類は幾つもあり複雑であるが，代表的なものが整理されて挙げられている。

・関数のインライン展開：関数を呼び出す箇所に，呼び出される関数のプログラムを展開する。関数を呼び出す場合には，制御が必要であり，それだけ手数が増える。その手数を減らすため，関数の内容をその場に展開する。それにより実行時間を短縮できる。ただし，何箇所も同じコードが展開されるとプログラムのサイズは大きくなってしまう。
・共通部分式の削除：同じ式が複数の箇所に存在し，それらの式で使用している変数の値が変更されず，式の値が変化しないとき，その式の値を作業用変数に格納する文（命令）を追加し，複数の箇所に存在する同じ式をその作業用変数で置き換える。1文追加されるが，複数箇所で同じ計算を何度も実行しなくて済むことにより実行時間の短縮になる。また，計算式が一つの変数になることによりプログラムサイズの縮小も期待できる。
・定数の畳込み：プログラム中の定数同士の計算式を，その計算結果で置き換える。前項の「共通部分式の削除」に似ているが，この場合は定数であるので，作業用変数に格納することなくダイレクトに定数に置き換える。
・定数伝播：変数を定数で置き換える。例えば，関数をインライン展開する際，実引数の定数を仮引数に代入するといった場合である。
・無用命令の削除：プログラムの実行結果に影響しない文を削除する。
・ループ内不変式の移動：ループ内で値の変化しない式があるとき，その式をループの外に移動する。一度計算しておけば値の変わらない式をループ内に残しておくと，繰り返し同じ計算を実行することになる。ループの外に移動すれば1回の実行で済む。
・ループのアンローリング：ループ内の繰返し処理を展開する。繰返しを実行するには，ループの制御が必要である。繰返しの回数をカウントし，判断しなければならない。もし，繰返しの回数が少ないのならば，ループ展開した方が制御の必要はなく，結果的に実行する文は少なくなって実行時間は短縮される。しかし，インライン展開同様，プログラムのサイズは大きくなってしまう。

［設問1］

・空欄a，b：問題では，次のプログラム1に対して，前述の最適化の方法を複数組み合わせて最適化した例について考える。

〔プログラム1〕
```
■ i: 0, i ≦ 1, 1
│ ・x[i] ← y[i] ＋ m ＋ n
│ ・v[i] ← w[i] ＋ m ＋ n
■
```

〔最適化の方法①を使った変換〕
```
■ i: 0, i ≦ 1, 1
│ ・t ← m ＋ n
│ ・x[i] ← y[i] ＋ t
│ ・v[i] ← w[i] ＋ t
■
```

プログラム1のループ内に変数の加算式m ＋ nが2箇所存在し，変数m，nは値が変わらないので，作業用変数tに格納する文が加えられ，m ＋ nの部分はtとしている。①は，最適化の方法「共通部分式の削除」である。

〔最適化の方法②を使った変換〕
```
・t ← m ＋ n
■ i: 0, i ≦ 1, 1
│ ・x[i] ← y[i] ＋ t
│ ・v[i] ← w[i] ＋ t
■
```

①で作業用変数tに格納した値はループ内で変わらないので，ループの外に出している。②は，最適化の方法「ループ内不変式の移動」である。したがって，空欄aは（カ）が正解である。

〔最適化の方法③を使った変換〕
```
・t ← m ＋ n
・x[0] ← y[0] ＋ t
・v[0] ← w[0] ＋ t
・x[1] ← y[1] ＋ t
・v[1] ← w[1] ＋ t
・i ← 2
```

②のループ部分を展開している。繰返しの回数は，iの初期値を0，増分1で変化させ，1以下の間繰り返すということから，i ＝ 0，i ＝ 1の2回だけである。③は，最適化の方法「ループのアンローリング」である。したがって，空欄bは（キ）が正解である。

［設問2］

・空欄c：ここでは，関数のインライン展開について考える。
元の文は，変数rにfunc(10)を代入するものである。

```
○整数型:r
   ⋮
・r ← func(10)
   ⋮
```

```
○整数型関数： func(整数型： p)
○整数型：w
▲ p ≧ 10
│ ・w ← p ＋ 5
├─
│ ・w ← p ＋ 15
▼
・return w
```

図の①では，別に定義されている関数funcを展開する。最適化の方法「関数のインライン展開」である。その際，実引数の値10を仮引数pに代入している。そのpの値はそのまま関数内のpの値として展開されている。最適化の方法「定数伝播」である。

```
① → ○整数型:r
        ⋮
     ▲ 10 ≧ 10
     │ ・#w ← 10 ＋ 5
     ├─
     │ ・#w ← 10 ＋ 15
     ▼
     ・r ← #w
        ⋮
```

次に，図の②では，p ＝ 10であることから，条件式p ≧ 10の判定結果がtrue（真）であること，そして，pを含んだ式が定数同士の計算式であることから，その結果に置き換えている。つまり，②は最適化の方法「定数の畳込み」を適用している。したがって，空欄cは（イ）が正解である。その後③では，条件式の判定結果がtrueであるので，関数func(10)の戻り値は15となる。最終的に，関数funcを呼び出してrに代入する文は，変数rに定数15を代入する文に変換されることになる。

```
② → ○整数型:r                ③ → ○整数型:r
        ⋮                              ⋮
     ▲ true                         ・r ← 15
     │ ・#w ← 15                       ⋮
     ├─
     │ ・#w ← 25
     ▼
     ・r ← #w
        ⋮
```

［設問3］

・空欄d，e：プログラムを変換し，プログラムの実行時間を短縮することが最適化の目的であった。しかし，大前提は「プログラムの機能を変えない」ということである。最適化をしないときと最適化をしたときとで実行結果が異なってしまっては，最適化とはいえない。設問3では，そのような例として，浮動小数点数の演算結果が異なってしまう現象を考える。

プログラム1に対して，最適化をしないときと表2の最適化をしたときとで，

y[0]＝307000000.0
y[1]＝305000000.0
m＝−303000000.0
n＝4.0

の値を与えて実行した結果は，次のとおりである。

最適化をしないとき：　x[0]＝4000004.0，x[1]＝2000004.0
最適化をしたとき　：　x[0]＝4000000.0，x[1]＝2000000.0

「同一優先順位の算術演算子は，左から順に演算する」ということから，プログラム1に対して，最適化をしないときと最適化をしたときとを比較すると，演算の順序が異なっている。y[i] ＋ m (i＝0, 1)を先に計算するか，m ＋ nを先に計算するかである。これは，設問1の解説でも述べたように，最適化の方法「共通部分式の削除」によって，共通に使われている式m ＋ nを1回行って，変数tに格納し，後は変数tに置き換えられたからである。したがって，空欄dは（イ）が正解である。

ここで注目する点は，mとnの値で絶対値が大きく異なっていることである。浮動小数点数演算では，演算前に絶対値の小さい数値の指数部を絶対値の大きい数値の指数部に揃える約束がある。すると，絶対値の小さい数値は表現できる桁から落ちて無視されてしまう現象が起こることがある。これを「情報落ち」と呼んでいる。演算結果も，最適化をしたときが4.0少なくなっており，この値は計算式m ＋ nのうち，絶対値の小さいnの値と同じである。演算結果の異なる原因は，この情報落ちが発生したことによると予想される。

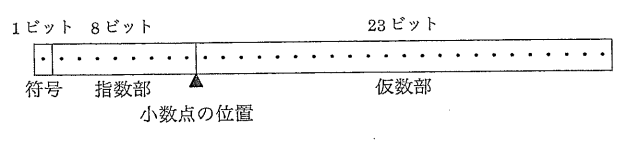
〔最適化をしないとき〕
```
・x[0] ← y[0] ＋ m ＋ n
・x[1] ← y[1] ＋ m ＋ n
```

〔最適化をしたとき〕
```
・t ← m ＋ n
・x[0] ← y[0] ＋ t
・x[1] ← y[1] ＋ t
```

実際に与えられた数値を代入して検証する前に，浮動小数点数の仕組みを確認しておく。浮動小数点数は数値の表現形式の一つで，

±(仮数)×(基数)^指数

と表す。また，コンピュータの中では「浮動小数点数の演算は単精度（仮数部は23ビット）で計算」ということであるから，一つの数値を32ビットで表す単精度では，符号1ビット，指数部8ビットになる。



問2　コンパイラの最適化（ソフトウェア）　(H24春·FE 午後問2)

【解答】

［設問1］　a－カ，b－キ
［設問2］　c－イ
［設問3］　d－イ，e－ウ

【解説】

コンパイラの最適化の方法，及び最適化と計算精度との関係に関する問題である。記述されている最適化の方法と具体的なプログラムの最適化過程を対応させていけば解答は難しくない。また，最適化実施の有無により浮動小数点数演算の結果が異なる例で，精度に関する理解度も問われているが，難易度の高いものではない。

コンパイラは，次の過程によって，プログラム言語で記述された原始プログラムを目的プログラムに翻訳する。コンパイラの最適化とは，原始プログラムを翻訳する過程で，プログラムの機能を変えることなく，プログラムの構造を変換することである。その目的は，問題の記述にあるように「プログラムの実行時間の短縮」である。また，ほかにも「プログラムのサイズの削減」という目的もある。

コンパイラの処理手順

原始プログラム→［字句解析→構文解析→意味解析→コード最適化→コード生成］→目的プログラム

最適化の方法の例として，七つの方法が示されている。実際には，プログラム言語やマシン環境によってその種類は幾つもあり複雑であるが，代表的なものが整理されて挙げられている。

・関数のインライン展開：関数を呼び出す箇所に，呼び出される関数のプログラムを展開する。関数を呼び出す場合には，制御が必要であり，それだけ手数が増える。その手数を減らすため，関数の内容をその場に展開する。それにより実行時間を短縮できる。ただし，何箇所も同じコードが展開されるとプログラムのサイズは大きくなってしまう。
・共通部分式の削除：同じ式が複数の箇所に存在し，それらの式で使用している変数の値が変更されず，式の値が変化しないとき，その式の値を作業用変数に格納する文（命令）を追加し，複数の箇所に存在する同じ式をその作業用変数で置き換える。1文追加されるが，複数箇所で同じ計算を何度も実行しなくて済むことにより実行時間の短縮になる。また，計算式が一つの変数になることによりプログラムサイズの縮小も期待できる。
・定数の畳込み：プログラム中の定数同士の計算式を，その計算結果で置き換える。前項の「共通部分式の削除」に似ているが，この場合は定数であるので，作業用変数に格納することなくダイレクトに定数に置き換える。
・定数伝播：変数を定数で置き換える。例えば，関数をインライン展開する際，実引数の定数を仮引数に代入するといった場合である。
・無用命令の削除：プログラムの実行結果に影響しない文を削除する。
・ループ内不変式の移動：ループ内で値の変化しない式があるとき，その式をループの外に移動する。一度計算しておけば値の変わらない式をループ内に残しておくと，繰り返し同じ計算を実行することになる。ループの外に移動すれば1回の実行で済む。
・ループのアンローリング：ループ内の繰返し処理を展開する。繰返しを実行するには，ループの制御が必要である。繰返しの回数をカウントし，判断しなければならない。もし，繰返しの回数が少ないのならば，ループ展開した方が制御の必要はなく，結果的に実行する文は少なくなって実行時間は短縮される。しかし，インライン展開同様，プログラムのサイズは大きくなってしまう。

［設問1］

・空欄a，b：問題では，次のプログラム1に対して，前述の最適化の方法を複数組み合わせて最適化した例について考える。

〔プログラム1〕
```
■ i: 0, i ≦ 1, 1
│ ・x[i] ← y[i] ＋ m ＋ n
│ ・v[i] ← w[i] ＋ m ＋ n
■
```

〔最適化の方法①を使った変換〕
```
■ i: 0, i ≦ 1, 1
│ ・t ← m ＋ n
│ ・x[i] ← y[i] ＋ t
│ ・v[i] ← w[i] ＋ t
■
```

プログラム1のループ内に変数の加算式m ＋ nが2箇所存在し，変数m，nは値が変わらないので，作業用変数tに格納する文が加えられ，m ＋ nの部分はtとしている。①は，最適化の方法「共通部分式の削除」である。

〔最適化の方法②を使った変換〕
```
・t ← m ＋ n
■ i: 0, i ≦ 1, 1
│ ・x[i] ← y[i] ＋ t
│ ・v[i] ← w[i] ＋ t
■
```

①で作業用変数tに格納した値はループ内で変わらないので，ループの外に出している。②は，最適化の方法「ループ内不変式の移動」である。したがって，空欄aは（カ）が正解である。

〔最適化の方法③を使った変換〕
```
・t ← m ＋ n
・x[0] ← y[0] ＋ t
・v[0] ← w[0] ＋ t
・x[1] ← y[1] ＋ t
・v[1] ← w[1] ＋ t
・i ← 2
```

②のループ部分を展開している。繰返しの回数は，iの初期値を0，増分1で変化させ，1以下の間繰り返すということから，i ＝ 0，i ＝ 1の2回だけである。③は，最適化の方法「ループのアンローリング」である。したがって，空欄bは（キ）が正解である。

［設問2］

・空欄c：ここでは，関数のインライン展開について考える。
元の文は，変数rにfunc(10)を代入するものである。

```
○整数型:r
  ⋮
・r ← func(10)
  ⋮
```

```
○整数型関数： func(整数型： p)
○整数型：w
▲ p ≧ 10
│ ・w ← p ＋ 5
├──
│ ・w ← p ＋ 15
▼
・return w
```

図の①では，別に定義されている関数funcを展開する。最適化の方法「関数のインライン展開」である。その際，実引数の値10を仮引数pに代入している。そのpの値はそのまま関数内のpの値として展開されている。最適化の方法「定数伝播」である。

①→
```
○整数型:r
  ⋮
▲ 10 ≧ 10
│ ・#w ← 10 ＋ 5
├──
│ ・#w ← 10 ＋ 15
▼
・r ← #w
  ⋮
```

次に，図の②では，p ＝ 10であることから，条件式p ≧ 10の判定結果がtrue（真）であること，そして，pを含んだ式が定数同士の計算式であることから，その結果に置き換えている。つまり，②は最適化の方法「定数の畳込み」を適用している。したがって，空欄cは（イ）が正解である。その後③では，条件式の判定結果がtrueであるので，関数func(10)の戻り値は15となる。最終的に，関数funcを呼び出してrに代入する文は，変数rに定数15を代入する文に変換されることになる。

②→
```
○整数型:r
  ⋮
▲ true
│ ・#w ← 15
├──
│ ・#w ← 25
▼
・r ← #w
  ⋮
```
③→
```
○整数型:r
  ⋮
・r ← 15
  ⋮
```

［設問3］

・空欄d，e：プログラムを変換し，プログラムの実行時間を短縮することが最適化の目的であった。しかし，大前提は「プログラムの機能を変えない」ということである。最適化をしないときと最適化をしたときとで実行結果が異なってしまっては，最適化とはいえない。設問3では，そのような例として，浮動小数点数の演算結果が異なってしまう現象を考える。

プログラム1に対して，最適化をしないときと表2の最適化をしたときとで，

y[0]＝307000000.0
y[1]＝305000000.0
m＝－303000000.0
n＝4.0

の値を与えて実行した結果は，次のとおりである。

最適化をしないとき：　x[0]＝4000004.0，x[1]＝2000004.0
最適化をしたとき　：　x[0]＝4000000.0，x[1]＝2000000.0

「同一優先順位の算術演算子は，左から順に演算する」ということから，プログラム1に対して，最適化をしないときと最適化をしたときとを比較すると，演算の順序が異なっている。y[i] ＋ m (i＝0, 1)を先に計算するか，m ＋ nを先に計算するかである。これは，設問1の解説でも述べたように，最適化の方法「共通部分式の削除」によって，共通に使われている式m ＋ nを1回行って，変数tに格納し，後は変数tに置き換えられたからである。したがって，空欄dは（イ）が正解である。

ここで注目する点は，mとnの値で絶対値が大きく異なっていることである。浮動小数点数演算では，演算前に絶対値の小さい数値の指数部を絶対値の大きい数値の指数部に揃える約束がある。すると，絶対値の小さい数値は表現できる桁から落ちて無視されてしまう現象が起こることがある。これを「情報落ち」と呼んでいる。演算結果も，最適化をしたときが4.0少なくなっており，この値は計算式m ＋ nのうち，絶対値の小さいnの値と同じである。演算結果の異なる原因は，この情報落ちが発生したことによると予想される。

〔最適化をしないとき〕
```
・x[0] ← y[0] ＋ m ＋ n
・x[1] ← y[1] ＋ m ＋ n
```

〔最適化をしたとき〕
```
・t ← m ＋ n
・x[0] ← y[0] ＋ t
・x[1] ← y[1] ＋ t
```

実際に与えられた数値を代入して検証する前に，浮動小数点数の仕組みを確認しておく。浮動小数点数は数値の表現形式の一つで，

±(仮数)×(基数)^指数

と表す。また，コンピュータの中では「浮動小数点数の演算は単精度（仮数部は23ビット）で計算」ということであるから，一つの数値を32ビットで表す単精度では，符号1ビット，指数部8ビットになる。