# 正　誤　表

平成 28 年 10 月 16 日実施

**基本情報技術者試験　午後　問題**

| ページ | 問題番号 | 行 | 誤 | 正 | 訂正の内容 |
|---|---|---|---|---|---|
| 7 | 1 | 下から2行目 | 3 要素との関連（一部）を表 3 にまとめた。 | 3 要素との関連を表 3 にまとめた。 | 下線部分を削除する。 |
| 8 | 1 | 上から1行目 | 表 3　情報セキュリティの 3 要素との関連（一部） | 表 3　情報セキュリティの 3 要素との関連 | 下線部分を削除する。 |

| ページ | 問題番号 | 行 | 誤 | 正 | 訂正の内容 |
|---|---|---|---|---|---|
| 40 | 9 | 下から1行目 | ③　目標作業期間は，今回の開発作業を開始する日から，… | ③　目標作業期間は，半年ごとの一連の開発作業を開始する日から，… | 下線部分を訂正する。 |
| 41 | 9 | (6) の上から2行目 | 遅延日数の合計wtは，次式で求められる。 | 遅延日数 wt は，次式で求められる。 | 下線部分を削除する。 |

次の問9から問13までの5問については，この中から1問を選択し，選択した問題については，答案用紙の選択欄の(選)をマークして解答してください。

なお，2問以上マークした場合には，はじめの1問について採点します。

問9　次のCプログラムの説明及びプログラムを読んで，設問1，2に答えよ。

C

〔プログラム1の説明〕

U社では，半年ごとに社内システムの見直しを行い，4月と10月に新たに開発又は改修するサブシステム（以下，対象サブシステムという）を決定し，開発又は改修作業（以下，開発作業という）を実施している。対象サブシステムは複数あり，それぞれに対して目標作業終了日が設定される。二つ以上の対象サブシステムの開発作業を同時に実施することはない。各対象サブシステムについて，開発リソースの制約があるので，必ずしも目標作業終了日までに開発作業を終了できるとは限らない。しかし，開発作業の終了が目標作業終了日よりも遅れる日数（以下，遅延日数という）の合計はできるだけ少なくしたい。そこで，対象サブシステムの開発作業順序を求める関数job_schedulingを作成した。

(1) 関数job_schedulingの仕様は次のとおりである。ここで，引数の値に誤りはないものとする。

機能：　対象サブシステムの開発作業順序を求める。

引数：　num_s　　対象サブシステム数

　　job　　対象サブシステム情報（JOB型の配列）

　　job_sch　開発作業順序（int型の配列）

(2) 構造体を使ったJOB型の定義は次のとおりである。

```
typedef struct {
    char pr_code[9];     /* 開発コード */
    int  job_term;       /* 開発作業日数 */
    int  target_term;    /* 目標作業期間 */
} JOB;
```

① 開発コードは，対象サブシステムごとに一意となる長さが8の文字列

② 開発作業日数は，開発作業に掛かる日数

③ 目標作業期間は，今回の開発作業を開始する日から，この対象サブシステムの

目標作業終了日までの日数

(3) 対象サブシステム情報 job には，次の①～③の順で，データが格納されている。

① 目標作業期間の昇順

② 目標作業期間が同じならば，開発作業日数の昇順

③ 目標作業期間も開発作業日数も同じならば，開発コードの昇順

(4) (5)の手順で，開発作業順序を求めると，開発作業順序 job_sch の要素 job_sch[i] には，i ＋ 1 番目（i ＝ 0，1，…，num_s － 1）に開発作業を実施する対象サブシステムの情報が格納された job[j] の添字の値 j（j ＝ 0，1，…，num_s － 1）が入る。

(5) 開発作業順序を求める手順は，次のとおりである。

① 開発作業順序 job_sch を，対象サブシステム情報 job の並び順に初期設定し，開発作業順序の添字 i に初期値 0 を設定する。

② i ＜ num_s － 1 である間，③～⑥の処理を行う。

③ i ＋ 1 番目に開発作業を実施する対象サブシステム job[job_sch[i]] と i ＋ 2 番目に開発作業を実施する対象サブシステム job[job_sch[i ＋ 1]] の開発作業を，順序どおりに実施した場合の二つの対象サブシステムの遅延日数の合計 wt_a と，順序を入れ替えて実施した場合の二つの対象サブシステムの遅延日数の合計 wt_b を求める。

④ 開発作業の実施順序を入れ替えることによって遅延日数の合計が減る場合（wt_a ＞ wt_b）には，開発作業の実施順序を入れ替える。

⑤ 開発作業の実施順序を入れ替えた場合，i が 0 でなければ，i を 1 減らす。

⑥ 入れ替えなかった場合，i を 1 増やす。

(6) i 番目の対象サブシステムの開発作業が終わるまでの開発作業日数の合計を ft としたとき，i ＋ 1 番目の対象サブシステムの開発作業が終わったときの遅延日数の合計 wt は，次式で求められる。

$$wt = \begin{cases} ft + job\_term - target\_term & (ft + job\_term > target\_term) \\ 0 & (ft + job\_term \leqq target\_term) \end{cases}$$

ここで，job_term と target_term は，i ＋ 1 番目に開発作業を実施する対象サブシステムの開発作業日数と目標作業期間である。

〔プログラム1〕

```
typedef struct {
    char pr_code[9];    /* 開発コード */
    int  job_term;      /* 開発作業日数 */
    int  target_term;   /* 目標作業期間 */
} JOB;

void job_scheduling(int, JOB[], int[]);

void job_scheduling(int num_s, JOB job[num_s], int job_sch[num_s]) {
    int ft, ft_a, ft_b, wt_a, wt_b, job_no, i, j;

    for (i = 0; i < num_s; i++) {
        job_sch[i] = i;
    }

    ft = 0;
    i  = 0;
    while (i < num_s - 1) {
        ft_a = ft;
        ft_b = ft;
        wt_a = 0;
        wt_b = 0;
        for (j = 0; j < 2; j++) { /* 遅延日数の合計 wt_a, wt_b を求める */
            ft_a += job[job_sch[i + j]].job_term;
            if (ft_a > job[job_sch[i + j]].target_term) {
                wt_a += ft_a - job[job_sch[i + j]].target_term;
            }
            ft_b += job[job_sch[ a ]].job_term;
            if (ft_b > job[job_sch[ a ]].target_term) {
                wt_b += ft_b - job[job_sch[ a ]].target_term;
            }
        }

        if (wt_a > wt_b) {
            job_no         = job_sch[i];
            job_sch[i]     = job_sch[ b ];
            job_sch[ b ] = job_no;
            if (i > 0) {
                ft -= job[job_sch[ c ]].job_term;
            }
```

```
        } else {
            ft [ d ] job[job_sch[i++]].job_term;
        }
    }
}
```

設問1　プログラム1中の［　　］に入れる正しい答えを，解答群の中から選べ。

aに関する解答群

ア　i + j　　イ　i + j - 1　　ウ　i - j　　エ　i - j + 1

bに関する解答群

ア　i + 1　　イ　i - 1　　ウ　job_no

エ　job_no + 1　　オ　job_no - 1

cに関する解答群

ア　i + 1　　イ　i - 1　　ウ　i++

エ　i--　　オ　++i　　カ　--i

dに関する解答群

ア　=　　イ　+=　　ウ　-=

設問2　関数 job_scheduling を実行したときに得られる対象サブシステムの開発作業順序などを出力する関数 print_schedule を作成した。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。

(1) 関数 print_schedule の仕様は次のとおりである。ここで，引数の値に誤りはないものとする。

機能： 対象サブシステムの開発作業順序などを出力する。

引数： num_s 対象サブシステム数

job 対象サブシステム情報（JOB 型の配列）

job_sch 開発作業順序（int 型の配列）

(2) 図 1 に示す五つの対象サブシステムの対象サブシステム情報に対して関数 job_scheduling を実行した。このときに得られた開発作業順序を，図 2 に示す。

| | pr_code | job_term | target_term |
|---|---|---|---|
| job[0] | "APL12339" | 25 | 27 |
| job[1] | "GGL08001" | 27 | 29 |
| job[2] | "MSO16101" | 21 | 30 |
| job[3] | "CAN03022" | 28 | 77 |
| job[4] | "ORA14031" | 12 | 93 |

図 1 対象サブシステム情報

| | |
|---|---|
| job_sch[0] | 0 |
| job_sch[1] | 2 |
| job_sch[2] | 1 |
| job_sch[3] | 4 |
| job_sch[4] | 3 |

図 2 開発作業順序

(3) 続けて，関数 print_schedule を実行した。その出力結果の 3 行目を図 3 に示す。

| | | | |
|---|---|---|---|
| 3 | GGL08001 | e | f |

注記 枠線は，実際には出力されない。

図 3 関数 print_schedule の出力結果の 3 行目

〔プログラム2〕

```
#include <stdio.h>

typedef struct {
    char pr_code[9];    /* 開発コード */
    int  job_term;      /* 開発作業日数 */
    int  target_term;   /* 目標作業期間 */
} JOB;

void print_schedule(int, JOB[], int[]);

void print_schedule(int num_s, JOB job[num_s], int job_sch[num_s]) {
    int ft, wt, wt_sum, i;

    ft      = 0;
    wt_sum  = 0;
    for (i = 0; i < num_s; i++) {
        ft += job[job_sch[i]].job_term;
        if (ft > job[job_sch[i]].target_term) {
            wt = ft - job[job_sch[i]].target_term;
            wt_sum += wt;
        } else {
            wt = 0;
        }
        printf("%3d  %10s  %10d  %10d\n",
               i + 1, job[job_sch[i]].pr_code, wt, wt_sum);
    }
}
```

e, fに関する解答群

| | | | |
|---|---|---|---|
| ア 0 | イ 27 | ウ 44 | エ 46 |
| オ 57 | カ 60 | キ 73 | ク 86 |