UESTC1005(HN)

Introductory Programming Tutorial Sample Solutions

01-Nov.-2025

NOTE AT THE FRONT

The following are the sample solutions to the exercises of this tutorial. They are provided for your reference, and you should NOT regard them as the only correct implementations.

STARTER/APPETIZER

Leap Year

```
bool is_leap_year(int year) {
    return ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0); // the 4-year, 100-year, and 400-year criteria, respectively.
}

int count_leap_years(int start_year, int end_year) {
        int count = 0;

    for(int year = start_year; year <= end_year; year++) {
        if(is_leap_year(year)){
            count++;
        }
    }
    return count;
}
```

You are also encouraged to check for non-negativity of inputs. After all, a year should always be a positive integer.

MAIN

Dish I – Difference of Squares

```c
#include <math.h>

unsigned int sum_of_squares(unsigned int number) {
    unsigned int sum = 0;
    for (unsigned int i = 1; i <= number; i++) {
        sum += pow(i, 2);
    }
    return sum;
}

unsigned int square_of_sum(unsigned int number) {
    unsigned int sum = 0;
    for (unsigned int i = 1; i <= number; i++) {
        sum += i;
    }
    return pow(sum, 2);
}

unsigned int difference_of_squares(unsigned int number) {
    return square_of_sum(number) - sum_of_squares(number);
}
```

You may choose not to utilize the `pow()` function.

Dish II – Resistor Colour Trio

```c
#include <stdio.h>

int resistor_color_trio(char bands[]) {
        const char color_values[] = {'k', 'n', 'r', 'o', 'y', 'g', 'b', 'v', 'e', 'w'};
        /* You can initialise these with other values.
        But why do we do so here? */
    int first_digit = -1;
    int second_digit = -1;
    int multiplier = -1;

    for(int i = 0; i < 10; i++) {// iterate through color_values[]
        if(bands[0] == color_values[i]) {
            first_digit = i;
        }
        if(bands[1] == color_values[i]) {
            second_digit = i;
        }
        if(bands[2] == color_values[i]) {
            multiplier = i;
        }
    }
    int resistance = (first_digit * 10 + second_digit) * pow(10, multiplier);

    if (resistance >= 1000 && (resistance % 1000 == 0)) {
        printf("%d kiloohms\n", resistance / 1000);
    } else {
        printf("%d ohms\n", resistance);
        }

    return resistance;
}
```

*Can you think of another way to verify the first/second digits, as well as the multiplier, in addition to the above solution?

DESSERT

Dish I – D&D Characters

```c
#include <stdlib.h>
#include <time.h>
#include <math.h>

static int roll_d6(void) {
        return (rand() % 6) + 1;
}

int ability(void) {
/* To generate values for each ability */
        int d0 = roll_d6(), d1 = roll_d6(), d2 = roll_d6(), d3 = roll_d6();
    int sum = d0 + d1 + d2 + d3;
    int minv = d0;
    if (d1 < minv) minv = d1;
    if (d2 < minv) minv = d2;
    if (d3 < minv) minv = d3;
    return sum - minv;
}

int modifier_floor(int score) {
    int diff = score - 10;
    if (diff >= 0) return diff / 2;

    return -(((-diff) + 1) / 2); // for the negative case
}

int* make_dnd_character() {
    srand((unsigned)time(NULL));
    int* out = malloc(7 * sizeof(int));
    if (!out) return NULL;

    out[0] = ability(); // STR
    out[1] = ability(); // DEX
    out[2] = ability(); // CON
    out[3] = ability(); // INT
    out[4] = ability(); // WIS
    out[5] = ability(); // CHA
    out[6] = 10 + modifier_floor(out[2]); // HP = 10 + CON modifier
    return out;
}
```

The above is for the single-player case. To include additional players, you can initialise a 2D array in the make_dnd_character() function. The overall idea still holds for the struct-based solution, as you have gone through the lectures; please adapt accordingly.

Dish II – Allergies

```
#include <stdlib.h>

static const int W[8] = {1, 2, 4, 8, 16, 32, 64, 128};

bool is_allergic_to(int score, int weight) {
/* Return 1 if 'weight' (must be one of W) is present in 'score', else 0. */
   for (int i = 0; i < 8; ++i) {
      if (W[i] == weight) {
         return (score & weight) ? true : false;  // bitmask check
      }
   }
   return false;
}

int* list_allergy_id(int score) {
   /* First pass: count how many IDs are present (for exact allocation). */
   int count = 0;
   for (int id = 0; id < 8; id++) {
      if (is_allergic_to(score, 1 << id)) {
         count++;
      }
   }
   int *ids = (int*)malloc((count) * sizeof *ids);
   if (!ids) return NULL;

   /* Second pass: fill the IDs in ascending order */
   int n = 0;
   for (int id = 0; id < 8; id++) {
      if (is_allergic_to(score, 1 << id)) {
         ids[n++] = id + 1; // Store IDs starting from 1
      }
   }

   return ids;
}
```