Structuur van Computerprogrammas II: Week 6

Elisa Gonzalez Boix Maarten Vandercammen Robbe De Greef

Multi-dimensional arrays

1. (**Dodona**) Schrijf een functie `void flatten(int multi[][2], int single[], int length)` die drie argumenten neemt: een tweedimensionale array, waar de binnenste array uit 2 integers bestaat, een gewone, eendimensionale array, en de lengte van de tweedimensionale array. De functie plaatst alle elementen uit de tweedimensionele array dan op hun overeenkomstige plaats in de eendimensionale. Als je `flatten` oproept op de tweedimensionale array { {1, 2}, {3, 4}, {5, 6}, {7, 8}} zou de eendimensionale daarna moeten overeenkomen met {1, 2, 3, 4, 5, 6, 7, 8 }.

Pointer arithmetic

2. (**Dodona**) Schrijf een functie `char *substring(char *str1, char *str2)` die, gegeven twee strings, nagaat of de tweede string een substring is van de eerste. Als dat zo is, geeft de functie een pointer terug naar de plaats in de eerste string vanwaar de substring start. Als dat niet zo is, geeft de functie de speciale pointerwaarde `NULL` uit <`stdlib.h`> terug.

Structs

3. Maak een struct `Point` dat een x- en een y-coördinaat bevat. Maak daarna een struct `Rectangle` dat een `Point` field bevat (dat dan zijn linkerbovenhoek voorstelt) en verder ook een breedte en een hoogte heeft. Schrijf ook functies `set_width`, `set_height` en `set_point` om deze drie eigenschappen van een gegeven rechthoek te veranderen door een nieuw, gegeven argument.

4. (**Dodona**) Implementeer een functie `void increment_day(struct date *d)` dat als argument een pointer naar een `struct date` met drie members (`year`, `month` en `day`) krijgt. `increment_day` zorgt ervoor dat de `date` waarnaar het argument wijst een dag verder wordt gezet. Als je bv. een variabele `d` hebt die gedefinieerd is als `struct date d = {.year = 2020, .month = 10, .day = 31}`, en je voert `increment_day(&d)` uit, zou `d`

daarna moeten overeenkomen met {`.year = 2020, .month = 11, .day = 1`}. Je hoeft geen rekening te houden met schrikkeljaren.

De definitie van een `struct date` wordt op Dodona al gegeven. Als je de oefening op je eigen computer wil uitvoeren, kan je de definitie van de struct in het bestand `oefeningen.c` vinden.

5. (**Dodona**) Stel dat je punten (met gehele coördinaten) wil voorstellen in een rechthoek met breedte `7` en hoogte `31`. Je zou het geheugenverbruik bij het voorstellen van deze punten kunnen minimaliseren door zo'n punten te *encoderen* a.d.h.v. zogenaamde *bit masks* als getallen met een grootte van 1 byte. In `oefeningen.c` op Canvas vind je de definitie van een `struct Small_Point` dat zo'n punten voorstelt. Op Dodona wordt deze definitie ook al gegeven.

   Schrijf een functie `unsigned char encode(struct Small_Point point)` die een punt omzet naar een getal (een `unsigned char`), en een functie `struct Small_Point decode(unsigned char encoded)`, die een geëncodeerd getal terug omzet naar een punt. Je kan hierbij gebruik maken van bitwise-operaties (zie slide 11 uit hoofdstuk 2) om het x- en y-coördinaat samen te voegen tot 1 getal. Meer informatie over hoe je zo'n masks kan gebruiken om deze functies te schrijven, vind je op `https://stackoverflow.com/a/142335`

6. (**Dodona**) In `oefeningen.c` op Canvas vind je de definitie van een `struct Chess_Piece`: een voorstelling van een schaakstuk met een kleur, type, en een x- en y-coördinaat.

   Schrijf opnieuw een functie `uint16_t encode_piece(struct Chess_Piece piece)` om zo'n schaakstuk te encoderen als een getal van 2 byte (een `uint16_t`, uit de `<stdint.h>` header file), en een functie `struct Chess_Piece decode_piece(uint16_t encoded)` om zo'n getal te decoderen.

   Als je deze oefening op Dodona maakt, wordt er verwacht dat je de velden van de struct in deze volgorde (van least naar most significant bit, t.t.z, van rechts naar links) bijhoudt: het y-coördinaat, het x-coördinaat, het type, en de kleur.

Extra

7. (**Dodona**) Schrijf een functie `struct Timestamp calc_difference(struct Timestamp t1, struct Timestamp t2)` die, gegeven twee zulke structs die tijdstippen (met members voor het aantal uren, minuten, en seconden), het verschil tussen de twee tijdstippen teruggeeft in de vorm van zo'n struct. Op Dodona wordt de definitie van een `struct Timestamp` al gegeven. Je vindt deze definitie ook in `oefeningen.c`.

8. Je kan matrices voorstellen met behulp van twee-dimensionale arrays, zoals aangegeven in de figuur hieronder. Schrijf nu een functie `void`

```
//                          K1  K2  K3  K4
double matrix_1[][4] = {{1,  2,  3,  4}, // rij 1
                        {5,  6,  7,  8}, // rij 2
                        {9,  10, 11, 12} // rij 3
//                           K1  K2  K3  K4
double matrix1[][4] = {{1,  2,  3,  4}, // rij 1
                        {5,  6,  7,  8}, // rij 2
                        {9,  10, 11, 12} // rij 3
};

//                           K1  K2  K3  K4  K5
double matrix2[][5] = {{20, 19, 18, 17, 16}, // rij 1
                        {15, 14, 13, 12, 11}, // rij 2
                        {10,  9,  8,  7,  6}, // rij 3
                        { 5,  4,  3,  2,  1}  // rij 4
};
double result[3][5]; // Resultaat komt hier in
matrix_multiply(matrix1, matrix2, result);
```

`matrix_multiply(double matrix1[][4], double matrix2[][5], double result[][5])` om een 3×4 matrix van doubles te vermenigvuldigen met een 4×5 matrix en het resultaat op te slaan in een derde, 4×5, matrix. In `oefeningen.c` vind je een functie `print_matrix` waarmee je het resultaat van de vermenigvuldiging, de 3×5 matrix, kan uitprinten.