A szerszámosláda

2. Struktúrák - Struct

2.1 Adatok

Egy számítógép feladata az információfeldolgozás. Az információk tulajdonképpen egy feladat adatai. Ezekből kiindulva, ezekkel műveleteket végezve, vagyis átalakítva, eljutunk a feladat megoldásáig.

Figure 2.1: Egyszerű és összetett adatszerkezetek

Az adatok lehetnek:

- **konstansok** - értéke nem változhat a megoldás folyamatában

változók - értéke változhat a megoldás folyamatában

A változók lehetnek:

egyszerű változók - csak egyetlen értéket tárolhatnak

összetett változók - több adattagot is tárolhatnak

A C nyelvben az alapértelmezett változók "egyszerű" változóknak számítanak. Példa egyszerű változó típusokra:

- int (egész szám)
- float (lebegőpontos szám)
- double (dupla precíziójú lebegőpontos szám)
- char (karakter)

Példa összetett változó típusokra:

- tömbök (például int myArray[10])
- struktúrák (struct)
- pointer-ek (például int *myPointer)

2.2 Struktúra definiálás. Struktúra változók.

Definíció 2.2.1 A struct (szerkezet, vagy rekord) egy adattípus, amely több változó tárolására alkalmas egyetlen változóban. A tárolt adatok különböző típusúak lehetnek, például egész számok, valós számok, karakterek stb. Ennek a típusnak a definiálása mindenképp kell tartalmazza a struct kulcsszót. A struct szó után következhet egy úgynevezett címke, amely segítségével változókat definiálhatunk.

2.2.1 Struktúra definiálás címkével

Általánosan:

```
struct structName {
   type1 member1;
   type2 member2;
   ...
};
```

Ebben az esetben a címke: structName.

Példa:

```
struct Point {
   int x;
   int y;
};
```

A fenti példában definiáltunk egy Point címkejű struct típust. Azonban amikor egy struktúra típus meghatározására kerül sor, akkor nem kerül lefoglalásra memóriarész. Ahhoz, hogy ez megtörténjen, ilyen típusú változókat kell létrehoznunk.

Struktúra változók definiálása

Általánosan:

1. Változat

```
struct structName {
   type1 member1;
   type2 member2;
};
int main() {
struct structName variable1, variable2;
...
}
```

2. Változat

```
struct structName {
   type1 member1;
   type2 member2;
}variable1,variable2;
```

Példa:

```
struct Point {
  int x;
  int y;
};
int main() {
  struct Point point1, point2, p[20];
  ...
}
```

vagy

```
struct Point {
  int x;
  int y;
}point1, point2, p[20];
```

Mindkét esetben a:

- point1 és point2 két Point típusú változó
- a p[] egy tömb, amely legtöbb 20, Point típusú elemet tud tárolni

2.2.2 Struktúra definiálás címke nélkül

Általánosan:

```
struct {
   type1 member1;
   type2 member2;
}variable1,variable2;
int main() {
```

```
variable1.member1 = 1;
...
```

Ezesetben, amint az alábbi programrészben is látható, a változókat nem tudjuk a főprogramban definiálni.

```
struct {
   type1 member1;
   type2 member2;
}
int main() {
struct variable1, variable2;//hibauzenet
...
}
```

A struct elemeit "tag"-nek, tagoknak, mezőknekvagy elemeknek nevezik. A struct típusú változó tagjainak az elérése különböző programozási nyelveken eltérő szintakszissal történhet. Általában a struct változó neve után következik a: - pont (.) operátor melyet a hozzáférendő mező neve követ vagy - mutató (->) operátor, ha a változó egy pointer melyet a hozzáférendő mező neve követ

Általánosan:

```
variable1.member1
variable1.member2
```

Példa:

```
point1.x
point1.y
```

2.2.3 Struktúra definiálás típusnévvel

A C nyelvben a typedef kulcsszó segítségével is lehet a struktúra típust új nevűre deklarálni. Gyakran használják struct típusú változók deklarálásánál, mert így rövidebben és egyszerűbben lehet hivatkozni rá.

Általánosan:

```
typedef struct typeName {
   type1 member1;
   type2 member2;
   ...
}typeName;
```

Példák:

1. Változat

```
#include <stdio.h>
typedef struct Convert {
```

```
    int ron;
    float euro;
} Convert;

int main() {
  Convert sum;
  ...
  }
```

2. Változat

```
#include <stdio.h>
typedef struct {
    int ron;
    float euro;
} Convert;

int main() {
  Convert sum;
  ...
  }
```

3. Változat

```
#include <stdio.h>
typedef struct Change{
    int ron;
    float euro;
} Convert;

int main() {
 Convert sum;
 ...
 }
```

Gyakori hibák:

```
#include <stdio.h>
typedef struct Convert{
    int ron;
    float euro;
};

int main() {
 Convert sum;//hibauzenet
 ...
 }
```

vagy

```
#include <stdio.h>
typedef struct Change{
```

```
    int ron;
    float euro;
}Convert;

int main() {
    Change sum;//hibauzenet
    ...
    }
```

A fenti esetekben egy új típusnévet definiáltunk (Convert), amely segítségével változókat is deklarálhatunk.

2.3 A struktúra adattagjainak kezdőértékadása, inicializálása

Ez többféle módon is lehetséges:

1. A struktúra példány létrehozásakor megadjuk az adattagok kezdeti értékeit:
   Példa:

```
  struct Circle {
    int x;
    int y;
    float r ;
    char type;
};
struct Circle circle = {10, 5, 3.14, 'a'};
```

2. A struktúra példány létrehozása után egyenként állíthatjuk be az adattagok értékeit:
   Példa:

```
struct Circle circle;
circle.x = 10;
circle.y = 5;
circle.r = 3.14;
circle.type = 'a';
```

   A struktúra deklarálása és a példány létrehozása között C nyelvben NEM inicializálható az adattagok értéke: Példa:

```
 struct Circle {
    int x = 10;//hibauzenet
    int y = 5;//hibauzenet
    float r = 3.14; //hibauzenet
    char type = 'a';//hibauzenet
};
struct Circle circle;
```

2.4 Struct típusú pointerek

A C nyelvben a struct típusokat pointerként is deklarálni lehet. A pointer deklarálásának alakja a következő:

```
struct Point {
    int x;
    int y;
} p1;
struct Point *p2 = &p1;
```

A fenti példában létrehoztunk egy struct Point típusú változót, amit p1-nek neveztünk el, majd létrehoztunk egy struct Point típusú pointert, amit p2-nek neveztünk el és a p1 változóra mutat (Fontos: ha a p2 ponter nem kapja meg a p1 változó címét, a p2-nek helyet kell foglalnunk dinamikusan). A -> operátor segítségével elérhetjük a struct változóban lévő tagjait. Ennek alapján az adattagok értékadása a következőképpen alakul:
p1.x = 1;
p1.y = 2;
p2->x = 1;
p2->y = 2;

2.5 Egymásba ágyazott struktúrák

A C nyelvben lehetőség van egymásba ágyazott (nested) struct típusok létrehozására. Az egymásba ágyazott struktúrákban egy adott struct típus tagjaként lehet egy másik struct típust használni.

```
struct Point {
    int x;
    int y;
};

struct Rectangle {
    struct Point topLeft;
    struct Point bottomRight;
};
```

Ez azt jelenti, hogy létrehoztunk egy struct Point típust ami x és y koordinátát tárol, majd létrehoztunk egy struct Rectangle típust, aminek tagjai két struct Point típusú változó. Az így létrejött Rectangle struct-ban tárolhatóak a négyzet topLeft és bottomRight koordinátái. Az egymásba ágyazott strukturák segítségével lehet komplex adatszerkezeteket létrehozni, amelyekben többféle adattípus van összefogva.

2.6 Struktúra függvényparaméter és visszatérési érték

A C nyelvben egy struct típust is használhatunk függvény paraméterként vagy visszatérési értékként. Példa függvény paraméterként:

```
struct Point {
    int x;
    int y;
```

```
};
    void printPoint(struct Point p) {
    printf("x: %d, y: %d\n", p.x, p.y);
}

int main() {
    struct Point p = {1, 2};
    printPoint(p);
    return 0;
}
```

A paraméterként átadott struct változó adattagjainak értékeit változtatjuk, ez nem látszik a főprogramban. Vagyis:

```
struct Point {
    int x;
    int y;
};
    void printPoint(struct Point p) {
        p.x=3;
        p.y=4;
    printf("x: %d, y: %d\n", p.x, p.y);
}

int main() {
    struct Point p = {1, 2};
    printPoint(p);
    printf("x: %d, y: %d\n", p.x, p.y);
    return 0;
}
```

A példa esetén a kiírt eredmény: x: 3, y: 4 x: 1, y: 2 Ennek magyarázata az, hogy ha egy struct típusú változót paraméterként adunk át egy függvénynek, akkor a függvény a paraméterben kapott változó másolatával dolgozik. Ha a függvénnyel végzett műveletek során a paraméterben kapott változó adattagjainak értékeit változtatjuk, akkor ezek az értékek nem látszanak a főprogramban. Erre a problémára két megoldás is van:

1. A függvény visszatérési értékének használata: a függvény a módosított változót visszaadja, és a főprogramban a visszaadott érték segítségével módosíthatjuk a változót.
2. A függvény paraméterének pointerként való átadása: a függvénnyel egy adattagot módosítunk a paraméterben kapott változóra mutató pointer segítségével.

Példa 1:

```
struct Point {
    int x;
    int y;
};
struct Point modifyPoint(struct Point p) {
    p.x = 3;
    p.y = 4;
    return p;
```

```
}

int main() {
    struct Point p = {1, 2};
    p = modifyPoint(p);
    printf("x: %d, y: %d\n", p.x, p.y);
    return 0;
}
```

Példa 2:

```
struct Point {
    int x;
    int y;
};
void modifyPoint(struct Point *p) {
    p->x = 3;
    p->y = 4;
}

int main() {
    struct Point p = {1, 2};
    modifyPoint(&p);
    printf("x: %d, y: %d\n", p.x, p.y);
    return 0;
}
```

Az első példa esetén a függvény visszatérési értékét használtuk, hogy a függvény által módosított változót visszaadjuk. A második példa esetén a függvénynek pointerként átadjuk a változót, így a függvény a paraméterben kapott változóra mutató pointer segítségével módosítja az adattagok értékét.

2.7 Felsorolás (enum) típus

A felsorolás egy olyan adattípus, amely lehetővé teszi a programozó számára, hogy nevezetes értékeket rendeljen hozzá egy változóhoz. A felsorolás értékei csak a meghatározott értékek közül lehetnek, így könnyebben elérhetővé válik a programozó számára a hibakeresés és a kód olvashatósága. Például egy programban a hét napjait lehetne felsorolni, és egy változót "nap" névvel ellátni, amely csak a hét napjai közül lehet értéke. A felsorolásokat általában a nyelv saját szintaktikájával definiálják, de a felsorolásokat gyakran használják a konstansok-ként, vagy a választható értékek listája-ként.

Előnyök:

- Átláthatóság: A felsorolások segítségével könnyen megadhatjuk a várható értékeket egy adott változónak, ami javítja a kód olvashatóságát és megakadályozza a helytelen értékek bevitelét.
- Memóriahatékonyság: a felsorolások kevés memóriát foglalnak, mivel az értékeik számok, így a kód hatékonyabb lesz
- Memóriahatékonyság: a felsorolások kevés memóriát foglalnak, mivel az értékeik számok, így a kód hatékonyabb lesz

- Könnyen kezelhetőség: a számokkal való műveletek gyorsabbak és hatékonyabbak, mint a szöveges értékekkel való műveletek
- Egyértelműség: könnyen megadhatjuk, hogy mely értékek használhatók egy adott változónál, ami csökkenti a programhibák számát és javítja a kód minőségét

Hátrányok:

- Konstans értékek hiánya: A felsorolások értékei nem állandók, így nehéz velük matematikai műveleteket végezni.
- Bővíthetőség korlátozottsága: A felsorolás értékei nem bővíthetők, így ha új értékre van szükség, a felsorolást újra kell definiálni.
- Összetett értékek hiánya: A felsorolások csak egyszerű értékeket tartalmazhatnak, nem lehet őket összetett adatstruktúrákkal kombinálni.
- Konverziós problémák: A felsorolások konvertálása más adattípusokká.

Példa 1:

```c
#include <stdio.h>
enum Days { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
int main() {
    enum Days today = Monday;
    scanf("%s",&today);//hiba!!!!!
    printf("%i",today);
    return 0;
}
```

Az eredmény: 0.

Példa 2:

```c
#include <stdio.h>
enum Days { Monday=100, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
int main() {
    enum Days today = Thursday;
    printf("%i",today);
    return 0;
}
```

Az eredmény: 103. Az enum változó konkrét értékének kiíratása a következőképpen történhet:

```c
#include <stdio.h>
enum Gender {Male, Female};
int main() {
    enum Gender g=Female;
    printf("Gender: %s\n", (g == 1) ? ("Female") : ("Male"));
    return 0;
}
```

Az eredmény: Gender: Female vagy

```c
#include <stdio.h>
enum Days {
    red, blue, green
```

```
};
char *printColors(enum Days color) {
    switch (color) {
        case 0:
            return "red";
        case 1:
            return "blue";
        case 2:
            return "green";
    }
}
int main() {
    enum Days color = green;
    printf("%The color is s.", printColors(color));
    return 0;
}
```

Az eredmény: The color is green.

2.8 Konstansok

A C nyelvben a konstansokat a const kulcsszóval deklarálhatjuk. A konstansok értékét deklaráláskor lehet megadni.

Példa:

```
const float pi = 3.14;
```

A konstansok, ahogy a neve is jelzi, olyan értékek, amelyek a program futása során nem változnak meg. A C nyelv az alábbi konstans típusokat támogatja:

- Számérték típus: Az egész (int), valós (float, double), Például: 42, 3.14, 0.1.
- Karakter típus: egy karakterből álló konstans, amelyet idézőjelek között kell megadni. Például: 'A', 'b', '5'.
- Sztring típus: Egy vagy több karakterből álló konstans, amelyet idézőjelek között kell megadni. Például: "Good morning!", "green".
- Szimbolikus típus: Olyan értékek, amelyeknek neve van, és amelyeket a #define direktíva segítségével definiálhatunk. Például: #define PI 3.14159. Ez esetben a preprocessor kicseréli a konstans összes előfordulását a megadott értékre. Ez esetben azonban nincs típusellenőrzés, és nincs értékadás, így, ha hibásan adunk meg egy értéket, a fordító nem fogja észre venni. Ezeket a konstansokat nem lehet direkt módon kiíratni.

A konstansokat a programban használva garantálható, hogy az értékeik nem változnak meg, ami biztonságosabb és könnyebben karbantartható kódot eredményez.

Példa:

```
#include <stdio.h>
#define pi 3.14;

int main() {
    printf("%.2f",pi);
```

```
    float s=pi;
    printf("%.2f",s);
    int e=pi;
    printf("%i",e); //helytelen eredmeny pirossal
    return 0;
}
```

2.9 Megoldott feladatok

Követelmény: Adott egy bemeneti állomány, amely tartalmazza n (0 <= n <= 250) diák:

1. vezetéknevét, keresztnevét
2. évfolyam számát (1,2,3,4)
3. nemét csoportját, számmal kifejezve (0-Informatics, 1-Engineer, 2-Computer Science)
4. laborgyakorlati jegyét (40% a végleges jegynek)
5. elméleti vizsgajegyét (40% a végleges vizsgajegynek)
6. gyakorlati vizsgajegyét (60% a végleges vizsgajegynek)
7. elméleti tesztkérdések eredményét (max. 3 pont=30 tesztkérdés helyes válasza), ami az elméleti vizsgajegyet emeli

Deklarálj egy olyan típusú változót, amelybe elmentheted a fenti adatokat. Számold ki minden diák vizsgajegyét valamit a végleges jegyét. A megoldást a következő függvényekkel add meg:

functions.h

```
#include <stdlib.h>
#include <stdio.h>
#ifndef ELMELET1_FUNCTIONS_H
#define ELMELET1_FUNCTIONS_H
enum Specialization {Computer_Science,Engineer,Informatics};
typedef struct{
    char name[50];
    int  year;
    int gender;
    enum Specialization specialization;
    float markPractice;
    float testResult;
    float markTheoryExam;
    float markPracticeExam;
    float examResult;
    float finalResult;
}Student_t;
char*getSpecialization(enum Specialization specialization);
void allocateMemoryForAllStudents(Student_t **dpStudents, int numberOfStudents);
void readOneStudentDetails(Student_t *pStudent);
void printOneStudent(Student_t student);
void calculateExamResultForOneStudent(Student_t *dpStudent);
void calculateExamResultForAllStudents(Student_t **dpStudents, int
    numberOfStudents);
void calculateFinalResultForOneStudent(Student_t *dpStudent);
```

```
void calculateFinalResultForAllStudents(Student_t **dpStudents, int
    numberOfStudents);
void readAllStudents(Student_t **dpStudents, int *numberOfStudents,const char
    *input);
void printAllStudents(Student_t *pStudents, int numberOfStudents, const char
    *destination);
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.23)
project(Elmelet1 C)
set(CMAKE_C_STANDARD 23)
add_executable(Elmelet1 src/main.c src/functions.c headers/functions.h)
include_directories(Elmelet1 headers)
```

functions.c

```
#include "../headers/functions.h"

char *getSpecialization(enum Specialization specialization) {
    switch (specialization) {
        case 0:
            return "Computer Science";
        case 1:
            return "Engineer";
        case 2:
            return "Informatics";
        default:
            return "Error";
    }
}

void allocateMemoryForAllStudents(Student_t **dpStudents, int numberOfStudents) {
    (*dpStudents) = (Student_t *) malloc(numberOfStudents * sizeof(Student_t));
    if (!(*dpStudents)) {
        printf("Memory allocation");
        exit - 1;
    }
}

void readOneStudentDetails(Student_t *pStudent) {
    scanf("%[^\n]\n", pStudent->name);
    scanf("%i", &pStudent->year);
    scanf("%i", &pStudent->gender);
    scanf("%i", &pStudent->specialization);
    scanf("%f", &pStudent->markPractice);
    scanf("%f", &pStudent->testResult);
    scanf("%f", &pStudent->markPracticeExam);
    scanf("%f\n", &pStudent->markTheoryExam);
}
```

```
void calculateExamResultForOneStudent(Student_t *dpStudent) {
    dpStudent->examResult = (dpStudent->markTheoryExam + dpStudent->testResult) *
        0.4 +
                            dpStudent->markPracticeExam * 0.6;
}
void calculateExamResultForAllStudents(Student_t **dpStudents, int
    numberOfStudents) {
    for (int i = 0; i < numberOfStudents; i++) {

        calculateExamResultForOneStudent(&(*dpStudents)[i]) ;
    }
}
void calculateFinalResultForOneStudent(Student_t *dpStudent) {
    dpStudent->finalResult = dpStudent->markPractice * 0.4 +
        dpStudent->examResult * 0.6;
}
void calculateFinalResultForAllStudents(Student_t **dpStudents, int
    numberOfStudents) {
    for (int i = 0; i < numberOfStudents; i++) {
        calculateFinalResultForOneStudent(&(*dpStudents)[i]);
    }
}

void printOneStudent(Student_t student) {
    printf("%s ", student.name);
    printf("%i ", student.year);
    printf("%s ",(student.gender)?("male"):("female"));
    printf("%s ", getSpecialization(student.specialization));
    printf("%.2f ", student.markPractice);
    printf("%.2f ", student.testResult);
    printf("%.2f ", student.markPracticeExam);
    printf("%.2f ", student.markTheoryExam);
    printf("%.2f ", student.examResult);
    if(student.markPractice<5 || student.markTheoryExam<5 ||
        student.markPracticeExam<5)
        printf("unclassifiable\n");
        else
    printf("%.2f \n", student.finalResult);
}

void readAllStudents(Student_t **dpStudents, int *numberOfStudents, const char
    *input) {
    if (!(freopen(input, "r", stdin))) {
        printf("File not found");
        exit(-2);
    };
    scanf("%i\n", numberOfStudents);
    allocateMemoryForAllStudents(&(*dpStudents), *numberOfStudents);
    for (int i = 0; i < *numberOfStudents; i++) {
        readOneStudentDetails(&(*dpStudents)[i]);
    }
    freopen("CON","r",stdin);
```

```
}

void printAllStudents(Student_t *dpStudents, int numberOfStudents, const char
    *destination) {
    if (!(freopen(destination, "w", stdout))) {
        printf("File not found");
        exit(-2);
    };
    for (int i = 0; i < numberOfStudents; i++) {
        printOneStudent(dpStudents[i]); }
    freopen("CON","r",stdout);
}
```

main.c

```
#include <stdio.h>
#include "functions.h"
int main() {
    /*Peldanyositas
    Student_t student1={"Nagy Bertalan", 3,1, 0 , 6, 0, 6, 7};
    calculateExamResultForOneStudent(&student1);
    calculateFinalResultForOneStudent(&student1);
    printOneStudent(student1);*/
Student_t *pStudent;
int numberOfStudents;
    readAllStudents(&pStudent,&numberOfStudents,"date.in");
       calculateExamResultForAllStudents(&pStudent,numberOfStudents);
       calculateFinalResultForAllStudents(&pStudent,numberOfStudents);
    printAllStudents(pStudent,numberOfStudents,"CON");
    return 0;
}
```

2.10 Javasolt kérdések

1. Milyen adattípusokat tárolhat egy struct típusú változó?
2. Hogyan adhatunk értéket egy struct mezőjének C nyelven?
3. Hogyan lehet hozzáférni az egyes mezőkhöz egy struct típusú változó esetén?
4. Lehet-e több struct típusú változót létrehozni egy adott struct típusból?
5. Mi az előnye a struct típusnak a programozásban?
6. Adott az alábbi deklaráció:

```
struct s {
int x;
float y;
};
struct s my_struct; }
```

   Hogyan lehet értéket adni az x és a y mezőknek?
7. Milyen típusú adatokat lehet eltárolni az x és a y mezőkben a fenti deklaráció esetén?
8. Lehet-e új mezőket hozzáadni a s struct típushoz? Ha igen adj egy példát erre.
9. Adott az alábbi deklaráció:

```
struct Date {
  int day;
  int month;
  int year;
};

struct Person {
  char name[50];
  int age;
  struct Date birthdate;
};
```

10. Mi a "tag" jelentése struktúrák kapcsán?
11. Az alábbi programrészlet nyomán hogyan tudunk hivatkozni egy Person típusú változó nevének első betűjére? Írj kódrészletet.

```
struct Person {
   char name[50];
   int age;
};
```

12. Az előző kérdésben megadott Person stukrúra alapján írj kódrészletet, amely helyet foglal dinamikusan a memóriában egy személynek.
13. Mi lesz a következő programrészlet eredménye?

```
#include<stdio.h>
int main()
{
   struct simp
   {
      int i = 6;
```

```
      char city[] = "chennai";
   };
   struct simp s1;
   printf("%d",s1.city);
   printf("%d", s1.i);
   return 0;
}
```

14. Mi lesz a következő programrészlet eredménye?

```
#include<stdio.h>
int main()
{
   enum numbers{num1, num2 = 0, num3, num4, num5, num6}n;
   printf("%d\n", sizeof(n));
}
```

Adott az alábbi struktúra szerkezet:

```
enum Specialization {Computer_Science,Engineer,Informatics};
typedef struct BirthDate{
int year,month,day;}BirthDate;
typedef struct{
char name[50];
int  year;
int gender;
BirthDate birthDate;
enum Specialization specialization;
float markPractice;
float testResult;
float markTheoryExam;
float markPracticeExam;
float examResult;
float finalResult;}Student;
```

15. Milyen típusú adatokat tárol a **name** mező?
16. Mi a **gender** mező értéktartománya?
17. Hogyan lehet inicializálni egy **BirthDate** típusú változót, amelyet a **Student** struktúra **birthDate** mezője tárol?
18. Milyen típusú adatokat tárol a **markPractice, testResult, markTheoryExam, markPracticeExam, examResult és finalResult** mezők?
19. Hogyan lehet egy **Student** típusú változó **finalResult** mezőjét kiszámolni a **markPractice, testResult, markTheoryExam, markPracticeExam** és **examResult** mezőkből?
20. Milyen értékeket vehet fel az **enum Specialization** típusú változó, amelyet a **Student** struktúra **specialization** mezője tárol?