Programozási Nyelvek: C++

Gyakorló feladatkönyv

Umann Kristóf

```
#include "CppStudent.h"

int main()
{
    CppStudent *reader = new CppStudent();
    reader->readBook();
    while(!reader->doesUnderstand())
    {
        reader->exercise();
        reader->readBook();
    }
    delete reader;
}
```

2018. május 5.

Tartalomjegyzék

1. Előszó

Ez a jegyzet az ELTE Informatikai Kar hallgatóinak készült a *Programozási Nyelvek C++* című tárgyhoz. Ez a dokumentum a gyakorlatokon történő számonkéréshez és vizsgához szolgál gyakorló feladatokkal.

Ez a dokumentum segédanyagként szolgál egy meglevő jegyzethez, mely szintén e projektnek a része!

A jegyzet első számú forrása Horváth Gábor 2015/2016/2 és 2016/2017/1 félévében tartott gyakorlatai, Pataki Norbert 2015/2016/1 előadásai, valamint a gyakorlaton megjelent helyettesítő tanárok órái: Porkoláb Zoltán, Brunner Tibor.

Külön köszönet jár Horváth Gábornak, aki a jegyzet javításában segített és aktívan segít, és mindenki másnak, aki az esetleges hibák észrevétele után szóltak.

A jegyzet teljes mértékben nyílt forráskódú, amennyiben esetleges hibába, pontatlanságba botlana, vagy szeretne segíteni az jegyzet fejlesztésében, az alábbi linken megteheti:

`https://github.com/Szelethus/ELTE-IK-CPP`.

(Felhívnánk rá a figyelmet, hogy a szerkesztés jelen pillanatában a jegyzet nem teljes terjedelmében lektorált.)

2. Preprocesszor

2.0.1. Feladat. Legyen adott ez a C++ fájl:

01_01pp_main.cpp

```
#include "01_01pp.h"
#include <iostream>

int main() { std::cout << add(5, 6) << std::endl; }
```

Kimenet: `11`

Implementáljuk a `01_01pp.h` header fájlt mely tartalmazza az `add` függvény deklarációját, mely két egész számot vár paraméterül, és az összegüket adja vissza visszatérési értékeként! A függvényt definiáljuk a `01_01pp.cpp` fájlban, majd fordítsuk le a programot és ellenőrizzük hogy helyesen működik-e!

2.0.2. Feladat. Legyen adott ez a C++ fájl:

01_02pp_main.cpp

```
#include <iostream>

// *

int main() { std::cout << 5 * ADD(1, 2) << std::endl; }
```

Kimenet: `15`

Implementáljunk a `*`-al jelölt sorba egy preprocesszor függvényt, mely az összeadást valósítja meg! Figyeljünk a műveletek asszociativitására is!

3. Mutatók

3.0.1. Feladat. Legyen adott ez a C++ fájl:

02_01ptr_main.cpp

```
#include <iostream>

int main() {
  int t[] = {1, 2, 3, 4};

  int *ptr;
  for (/* ... */) {
```

```
    std::cout << /* t egyik eleme */ << std::endl;
  }
}
```

Iteráljunk végig `t` tömb elemein újabb változó létrehozása nélkül! Hogyan változna a megoldás, ha `t` méretét megváltoztatjuk?

3.0.2. Feladat. Legyen adott ez a C++ fájl:

02_02ptr_main.cpp

```
#include <iostream>

void changePointer(/* ... */) {
  // ...
}

int main() {
  int c = 5, d = 6;
  int *ptr = &c;
  changePointer(&ptr, &d);
  std::cout << ptr << '\n' << &d << std::endl;
}
```

Implementáljuk a `changePointer` függvényt, mely egy `int`re mutatót egy, a paraméterként átadott `int` objektum címére állít! Megoldásunk helyességét azzal ellenőrizhetjük, hogy `ptr` és `&d` értékét kiíratjuk – amennyiben ezek azonosak, az azt jelenti hogy `ptr` `d`-re mutat.

3.0.3. Feladat. Miben változna az előző feladat megoldása, ha `c` és `d` konstansok lennének? Implementáljuk a `02_03ptr_main.cpp` fájlba!

3.0.4. Feladat. Legyen adott ez a C++ fájl:

02_04ptr_main.cpp

```
#include <iostream>

int findIf(/* ... */) {
  // ...
}

bool isEven(int i) { return i % 2 == 0; }

bool isEqualToSeven(int i) { return i == 7; }

bool isNegative(int i) { return i < 0; }

int main() {
  int t[] = {1, 3, 4, 6, 7, 8, 10};

  std::cout << findIf(/* ... */, isEven) << ' '
            << findIf(/* ... */, isEqualToSeven) << ' '
            << findIf(/* ... */, isNegative) << std::endl;
}
```

Kimenet: `2 4 -1`

Valósítsuk meg a `findIf` függvényt, mely egy tetszőleges `int`-eket tartalmazó tömböt és egy predikátum függvényt (olyan függvény, melynek `bool` a visszatérési értéke és egy paramétere van, ez esetben `int` típusú) kap paraméterül, és amennyiben a tömb tartalmaz olyan elemet amelyre a predikátumfüggvény igazat ad, adja vissza a tömbbéli indexét, különben térjen vissza `-1`-el!

Azt, hogy az Olvasó hogyan szeretné a `findIf` függvény implementálni (beleértve azt is, hogy milyen paramétereket várjon) nincs megkötve, azzal a kivétellel, hogy a predikátumfüggvény függvény legyen az utolsó paraméter.

3.0.5. Feladat. Hogyan tudnánk a fenti feladatokat referenciákkal megoldani?

4. Láthatóság, élettartam, sztringkezelés, függvénytúlterhelés

4.0.1. Feladat. Legyen adott ez a C++ fájl:

03_01str_main.cpp

```
#include <iostream>

void insertSpace(const char *orig, char *dest) {
  // ...
}

int main() {
  const char *str = "Sziamizujs?";
  char buf[256];

  insertSpace(str, buf);

  std::cout << buf << std::endl;
}
```

Kimenet: `Szia mizujs?`

Valósítsuk meg az `insertSpace` függvényt, mely minden `'a'` karakter után beszúr egy space-t a paraméterként kapott strigbe! Feltételezhetjük, hogy `dest` legalább akkora tömbre mutat, mint `orig`.

Hogyan változna a megoldás, ha `char*` és `char[]` helyett `std::string` objektumokkal dolgoznánk?

4.0.2. Feladat. Egészítsünk ki egy korábbi feladatot!

03_01static_main.cpp

```
#include <iostream>

int findIf(/* ... */) {
  // ...
}

bool isSecondEven(int i) {
  // ...
}

int main() {
  int t[] = {1, 3, 4, 6, 7, 8, 10};

  std::cout << findIf(/* ... */, isSecondEven) << std::endl;
}
```

Kimenet: `3`

A korábban megvalósított `findIf` függvényhez írjunk egy újabb predikátumfüggvényt, mely egy statikus változó segítségével akkor ad igazat, ha a paraméterként kapott `int` a második páros szám!

Soroljunk fel legalább két indokot, miért célszerűtlen ez a megoldás valós alkalmazásra!

4.0.3. Feladat. Legyen adott ez a C++ fájl:

03_01overload_main.cpp

```
#include <iostream>

struct IntWrapper {
```

```
  int i;
};

// *

int main() {
  int i = 10;
  IntWrapper wrap;
  wrap.i = 5;

  std::cout << add(i, wrap) << ' ' << add(wrap, i) << ' ' << add(wrap, wrap)
            << ' ' << add(i, i) << std::endl;
}
```

Kimenet: `15 15 10 20`

Valósítsuk meg az `add` függvényt, mely két tetszőleges szám összegét adja vissza! Függvénytúlterhelés segítségével érjük el azt, hogy ez a két paraméter mind `int` mind `IntWrapper` típusú is lehessen!

4.0.4. Feladat. Legyen adott ez a C++ fájl:

03_02overload_main.cpp

```
#include <iostream>

struct IntWrapper {
  int i;
};

// *

int main() {
  IntWrapper wrap1, wrap2;
  wrap1.i = 10;
  wrap2.i = 5;

  std::cout << wrap1 * wrap2 << ' ' << wrap1 / wrap2 << ' ' << wrap1 + wrap2
            << ' ' << wrap1 - wrap2 << std::endl;
}
```

Kimenet: `50 2 15 5`

Operátor túlterheléssel valósítsuk meg a szorzás, osztás, összeadás és a kivonás műveletét! Minden operátor két `IntWrapper` típusú paramétert várjon, és a visszatérési értéke is egy `IntWrapper` típusú változó legyen! Gondoljuk meg, érték vagy referencia szerinti átvétel lenne-e hatékonyabb!

A kiíratáshoz terheljük túl a right shift operátort (`<<`) a jegyzetben található módon!

5. Memóriakezelés

5.0.1. Feladat. Legyen adott ez a C++ fájl:

05_01heap_main.cpp

```
#include <iostream>

void allocAll(int **t, int size) {
    // ...
}

void fillAll(int **t, int size) {
    // ...
```

```
}

void printAll(int **t, int size) {
// ...
}

void freeAll(int **t, int size) {
    // ...
}

int main() {
    int* t[20];
    allocAll(t, sizeof(t) / sizeof(t[0]));
    fillAll(t, sizeof(t)/sizeof(t[0]));
    printAll(t, sizeof(t)/sizeof(t[0]));
    freeAll(t, sizeof(t)/sizeof(t[0]));
}
```

Kimenet: `0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19`

Implementáljuk az `allocAll` függvényt mely végigiterál egy tetszőleges `int` pointereket tároló tömbön, és minden eleméhez allokál a heapről memóriát! Ezután a `fillAll` függvénnyel a tömb összes elemét változtassuk meg a mutatott értékét a tömbbéli indexére, a `printAll` függvény a tömb összes eleme által mutatott értéket írassa ki, majd a `freeAll` függvénnyel szabadítsuk fel a memóriát!

5.0.2. Feladat. Módosítsuk az előző feladatot úgy, a `t` tömb maga is heapen legyen allokálva!