

Katona József – Kővári Attila

GENERIKUS PROGRAMOZÁS
Osztálysablonok,
Általános felépítésű függvények,
Függvénynevek túlterhelése és
Függvénytípusablonok

Gyakorlatorientált szoftverfejlesztés

C++ nyelven

Visual Studio Community
fejlesztőkörnyezetben

```
#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;
    return 0;
}
```


GENERIKUS PROGRAMOZÁS
Osztálysablonok,
Általános felépítésű függvények,
Függvénynevek túlterhelése és
Függvénysablonok

Gyakorlatorientált szoftverfejlesztés C++ nyelven
Visual Studio Community fejlesztőkörnyezetben

@Katona József – Kővári Attila
Lektorálta: Kratzmajer Zoltán

Publio Kiadó
Felelős kiadó: Publio Kiadó Kft.
2016

Minden jog fenntartva!

DOI: 10.18395/generikus.programozas.2016

Tartalomjegyzék

Előszó.....	5
Generikus programozás.....	6
Osztálysablonok.....	6
Minta példa Visual Studio Community fejlesztőkörnyezetben	7
Statikus tervezés.....	8
A projekt létrehozása.....	8
A feladathoz kapcsolódó magyarázatok	23
Függvénynevek túlterhelése, Függvénytípusok	24
Minta példa „általánosan alkalmazott” függvények használatára Visual Studio Community fejlesztőkörnyezetben	25
Statikus tervezés.....	26
A projekt létrehozása.....	26
A feladathoz kapcsolódó magyarázatok	43
Minta példa függvénynevek túlterhelésére Visual Studio Community fejlesztőkörnyezetben.....	45
Statikus tervezés.....	47
A projekt létrehozása.....	48
A feladathoz kapcsolódó magyarázatok	71
Minta példa függvénytípusok használatára Visual Studio Community fejlesztőkörnyezetben.....	74
Statikus tervezés.....	76
A projekt létrehozása.....	76
A feladathoz kapcsolódó magyarázatok	96
Gyakorló példa.....	98
Osztály diagram	99
Megoldás	100
Felhasznált irodalom.....	107

Előszó

A C++ programozási nyelv egy általános célú, magas szintű nyelv. A nyelv elterjedtségét és létjogosultságát támassza alá, hogy szinte minden operációs rendszer alá létezik C++ fordító. A C++ nyelv elsajátításához szükséges a C nyelv ismerete is, mivel a C++ általánosságban a C nyelv szintaxisára és koncepcióira épül, annak kiterjesztése.

E könyv a C++ nyelven történő objektumorientált szoftverfejlesztést megismertető sorozat harmadik kötete, mely sorozat gyakorlati példákon keresztül segíti a nyelv elsajátítását. Bízunk benne, hogy azok, akik végigolvassák a könyvet, és kidolgozzák a példákat, alkalmazás szintjén megismerik a C++ nyelvi elemei által adott lehetőségeket, a nyelv logikáját, az átláthatóbb szoftver készítést elősegítő funkcióját és mélyebben elsajátítják az objektumorientált szemlélet alapjait, mely a hatékonyan szoftverfejlesztéshez elengedhetetlen. A bemutatott ismeretek gyakorlati példákkal illusztráltak, melyek a megértést és az alkalmazás elsajátítását is nagymértékben segítik.

Jelen kötet a generikus programozásban használt osztálysablon, általános felépítésű függvényt, függvénynév túlterhelést, függvénysablon és ezen fogalmak kapcsolatát ismerteti, továbbá egy projekten keresztül lépésről-lépésre bemutatja ezeknek a nyelvi eszközöknek a használatát és jelentőségét. A mintapélda a könyvsorozat első kötetében bemutatott Visual Studio Community fejlesztőkörnyezetben kerül megvalósításra, melyben a bemutatott mintapélda implementálható és összeállítható, a futtatható kód előállítható.

Köszönettel tartozunk a kézirat lektorának, Kratzmajer Zoltánnak szakmai tanácsaiért és módszertani javaslataiért.

A Szerzők

Generikus programozás

A generikus programozási paradigma során használt sablonok (**templates**) és a hozzájuk rendelt típusok, mint formális paraméterek, lehetőséget kínálnak a programozók számára, hogy egy logikailag egymással kapcsolatban álló osztályok és függvények csoportját hozzák létre. A könyvsorozat első és második kötetében egy adott függvény paraméter értékeit adott – jellemzően primitív (**integer, double, char, string, bool** stb) – típussal használtuk. A generikus programozás bármilyen más paradigmával (strukturált, objektumorientált) használható, célja, hogy adattípustól függetlenül képesek legyünk újra felhasználni kódrészleteket. A sablonok lehetőséget kínálnak arra, hogy általános típussal dolgozzunk, ahelyett, hogy úgynevezett meghatározott típusokat használnánk. Az ilyen jellegű programozási technika alkalmazásánál a fordító (nem a futtató rendszer) feladata, hogy egy sablon helyére megfelelő típussal helyettesítse be a használni kívánt változót, mindazonáltal fordítási időben szabadon paraméterezhető.

Az osztályok és függvények akkor válnak teljessé, ha azok hívásra kerülnek. A keretrendszer fejlesztők gyakran használnak sablonokat, ugyanis, ha például egy függvényt fel szeretnénk használni több, eltérő típussal, akkor nem szükséges minden egyes típus esetén új metódust implementálnunk, mivel a sablonok használatával elég azt csak egyszer megírunk és a hívásakor átadni neki azt a típust, amivel dolgozni szeretnénk.

Összefoglalva a generikus programozás megközelítésének lényege, hogy a sablont és a hozzá tartozó szemantikát a programozó határozza meg, minden egyéb a fordítóprogram felelőssége. Az osztályokat, függvényeket nemcsak értékekkel, hanem egy vagy több típussal is paraméterezünk. Generikus programozás növeli az újrafelhasználás mértékét, hiszen típusoktól független algoritmusokat lehet a segítségével megvalósítani. Szokás parametrikus polimorfizmusnak is nevezni. A C++ nyelven függvény- és az osztálysablonok (**templates**) készítésével valósítható meg a többféle típussal történő paraméterezés, mint például egy osztálysablon esetén:

```
template <class T> Osztálynév<T>::Függvénynév(T paraméter).
```

Osztálysablonok

A típussablonnal paraméterezett osztály (**generic class**), lehetővé teszi, hogy más osztályok definiálásához a paraméterezett osztályt felhasználjuk, ezáltal egy adott osztálydefiniáció minden típus esetén alkalmazható lesz. Tehát az osztály példányosítása során átadjuk azt a típust, amellyel használni szeretnénk az osztályunkat, majd ha más típussal szeretnénk tovább dolgozni, akkor nincs más teendő, mint új objektumpéldány létrehozása és a megfelelő típus átadása.

A fentebb megismert osztálysablon fogalmak gyakorlati elsajátításához a következőkben leírt minta feladat tervezését és implementálását végezzük el. A

típussablonnal paraméterezett osztály (**generic class**), lehetővé teszi, hogy más osztályok definiálásához a paraméterezett osztályt felhasználjuk, ezáltal egy adott osztálydefiníció minden típus esetén alkalmazható lesz. Tehát az osztály példányosítása során átadjuk azt a típust, amellyel szeretnénk használni az osztályunkat, majd ha más típussal szeretnénk tovább dolgozni, akkor nincs más teendő, mint új objektumpéldány létrehozása és a megfelelő típus átadása.

A fentebb megismert osztállysablon fogalmak gyakorlati elsajátításához a következőkben leírt minta feladat tervezését és implementálását végezzük el.

Minta példa Visual Studio Community fejlesztőkörnyezetben

Írjunk egy Ikozaéder osztályt (*Ikozaeder*), amely számításokat végez a poliéderen. Vegyük figyelembe, hogy az „ikozaéder” szó alatt szabályos ikozaédert értünk, amely az öt platóni vagy szabályos test egyike, konvex, szabályos poliéder és húsz háromszöglap alkotja, amelyekből a csúcson öt-öt találkozik. Az osztály adatai a következők:

- a , a szabályos ikozaéder él hossza.

Ügyeljünk az egységbezárás elvére, tehát az adat(ok) **private** elérésűek legyenek!

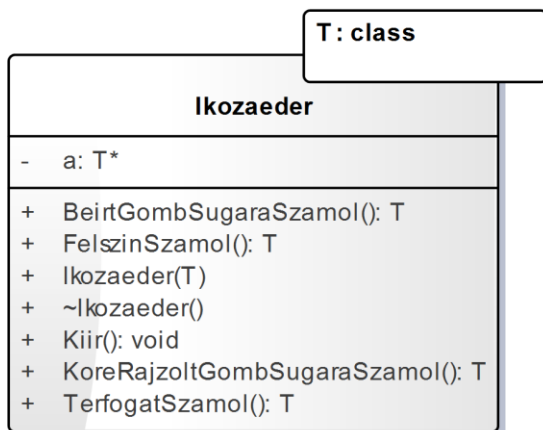
Valósítsuk meg az osztály metódusait:

- Írjuk meg az osztály konstruktorát, mely kötelező jelleggel paraméterül várja a szabályos ikozaéder él hosszát. A konstruktor megvalósításnál alapértelmezett paraméteres konstruktort használjunk.
- Írjunk egy *FelszinSzamol()* függvényt, amely az alábbi képlet segítségével: $A = 5\sqrt{3}a^2$ kiszámolja a szabályos ikozaéder felszínét (A).
- Írjunk egy *TerfogatSzamol()* függvényt, amely az alábbi képlet segítségével: $V = \frac{5}{12}(3 + \sqrt{5})a^3$ kiszámolja a szabályos ikozaéder térfogatát (V).
- Írjunk egy *KoreRajzoltGombSugaraSzamol()* függvényt, amely az alábbi képlet segítségével: $r_u = \frac{a}{4}\sqrt{10 + 2\sqrt{5}} \cdot a$ kiszámolja a szabályos ikozaéder köré rajzolt gömb sugarát (r_u).
- Írjunk egy *BeirtGombSugaraSzamol()* függvényt, amely az alábbi képlet segítségével: $r_i = \frac{a}{12}\sqrt{3}(3 + \sqrt{5}) \cdot a$ kiszámolja a szabályos ikozaéderbe beírt gömb sugarát (r_i).
- Írjunk egy egyszerű *Kiir()* függvényt, amely a fentebbi függvények által kiszámolt adatokat megjeleníti.
- Ne feledkezzünk meg az objektumpéldány által lefoglalt erőforrás felszabadításáról, a destruktor megírásáról sem.

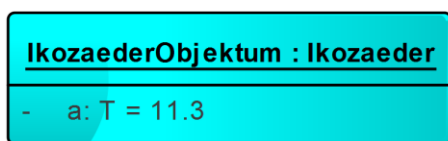
Az osztály implementálását követően, dinamikus módszerrel (a memória dinamikus területén [**heap**]) készítsünk egy objektumpéldányt. Az

objektumpéldány segítségével a feladat leírásnak megfelelően hívjuk meg az osztály egyes tagfüggvényeit. Generikus programozás lévén, próbáljuk meg az objektumpéldányt több primitív típussal (**double**, **float**, **short** és **integer**) is létrehozni. Figyeljük meg a változást! Amennyiben nincs tovább szükség az objektumpéldányra, gondoskodjunk annak megszüntetéséről. Mielőtt a feladatnak nekilátnánk, tervezzük meg a statikus modelleket.

Statikus tervezés



1. ábra: Osztály diagram



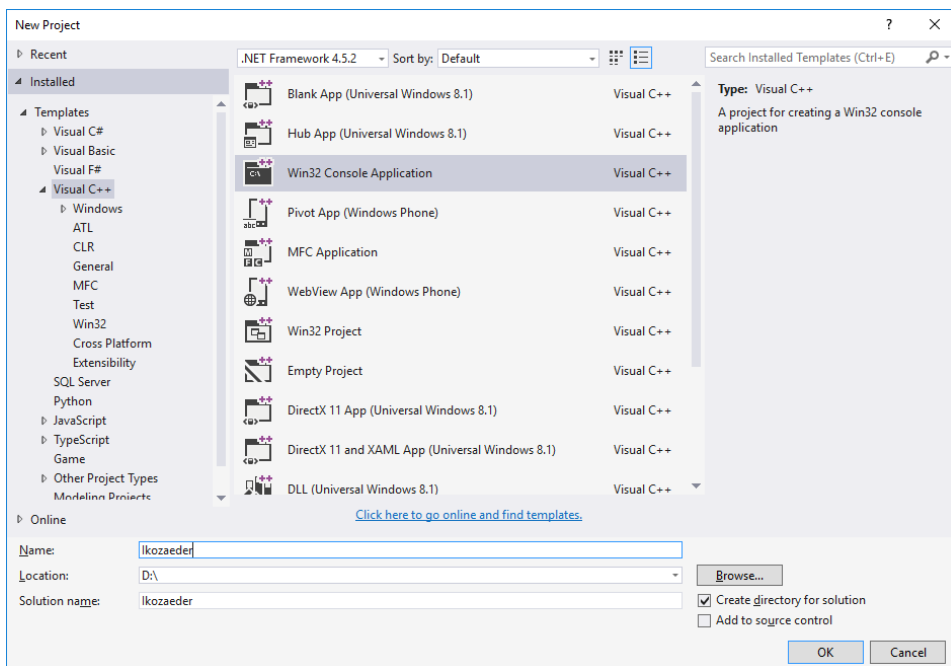
2. ábra: Objektum diagram

A projekt létrehozása

A projekt megvalósítása során konzolalkalmazást készítünk, ahol a Visual Studio, szabványos C++-nak megfelelő szintaktikát alkalmaz. A projekt három fájlban kerül implementálásra. Az alkalmazás belépési pontja a **main** függvény, amely a három fájl közül az egyikben fog helyet foglalni. Az előzőekben megfogalmazott mintapélda kerül implementálásra és összeállításra, továbbá Windows operációs rendszeren futtatható kóddá történő lefordításra Visual Studio

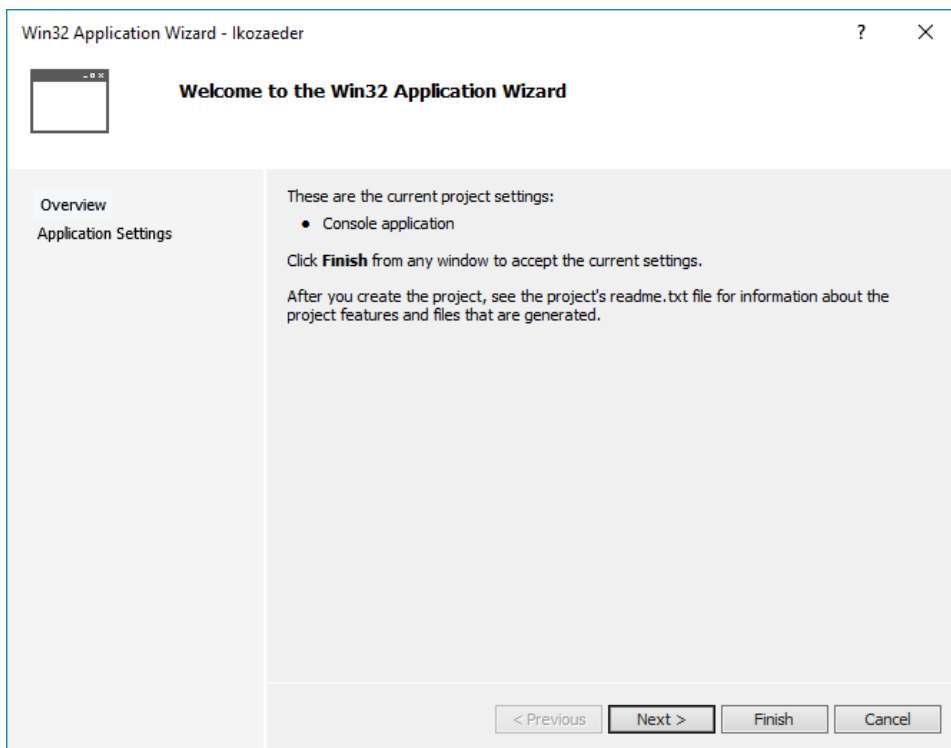
Community fejlesztőkörnyezet felhasználásával. A lefordított 32 bites alkalmazás konzolban kerül futtatásra.

Új projekt a **File** menü **New** menüpontjában hozható létre, a **Project** almenüt kiválasztva, vagy akár a **Ctrl+Shift+N** gyorsindító gombkombinációval. Új projekt kiválasztása után megjelenő ablakban a **Visual C++ Win32** sablont (**Template**) kell kiválasztani és a **Win32 Console Application**-t, mely az alábbi ábrán látható.



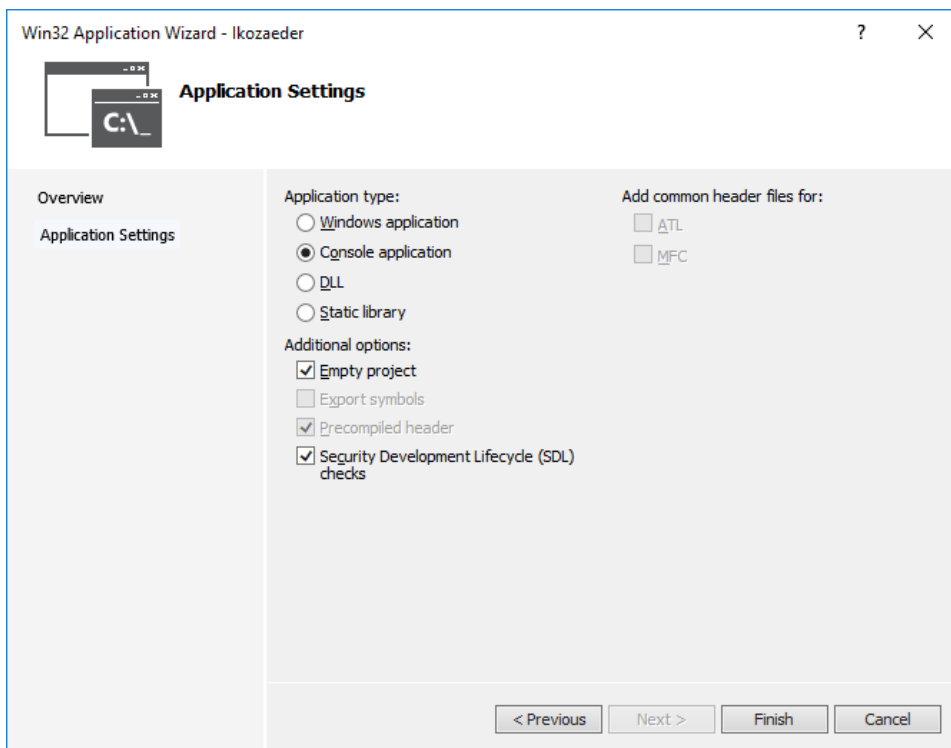
3. ábra: Win32 Console Application projekt létrehozása

Az ablak alsó részén a projekt nevének (**Name**) a példának megfelelően az **Ikozaeder** név került megadásra. Ebben a részben lehet módosítani a Projekthez kapcsolódó fájlok mentési helyét is (**Location**). Egy összetett program általában több projektből tevődik össze, ezeket egységes rendszerre a megoldás (**Solution**) fogja össze. Az OK gombra kattintást követően elindul az alkalmazás varázsló, mely a következő ábrán látható.



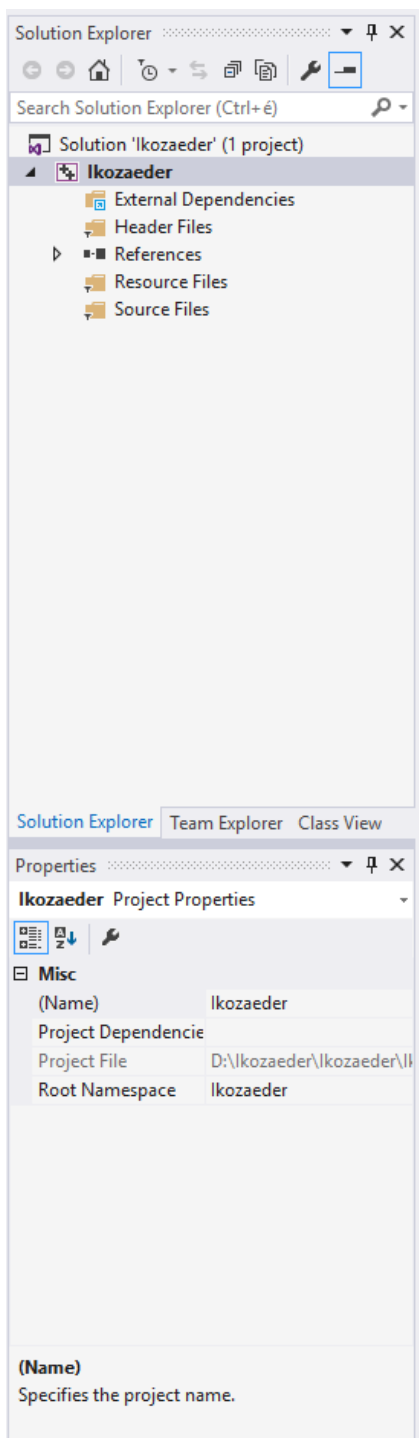
4. ábra: Win32 konzol alkalmazás varázsló

A **Next** gombra kattintva a beállítások ablak jelenik meg, itt megadhatjuk az alkalmazás típusát (**Console Application**), és néhány kiegészítő opciót. Mivel egyszerű alkalmazást készítünk, ezért célszerű az üres projekt (**Empty project**) opció kiválasztása, ahogy az alábbi képen látszódik.



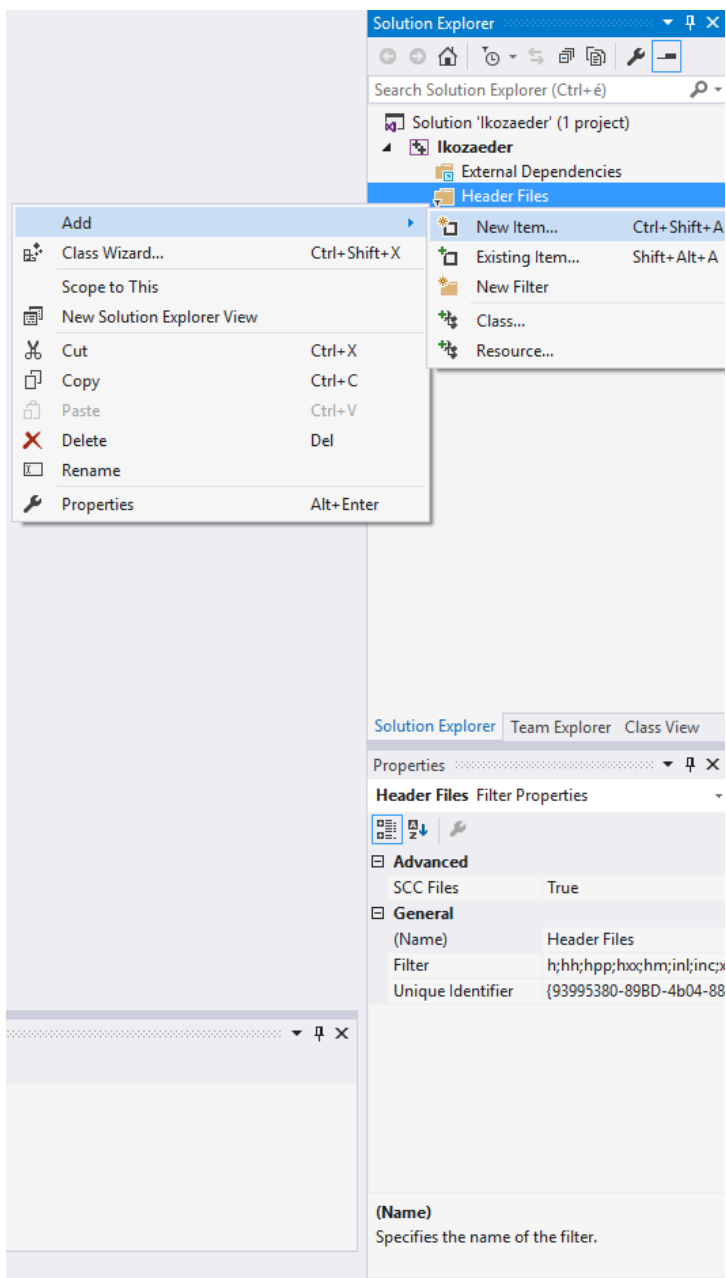
5. ábra: Alkalmazás beállításai

A **Finish** megnyomását követően a képernyő jobb szélén, a megoldás ablaka (**Solution Explorer**), alatta a beállítások ablak jelenik meg, ahogy a következő ábrán látható.



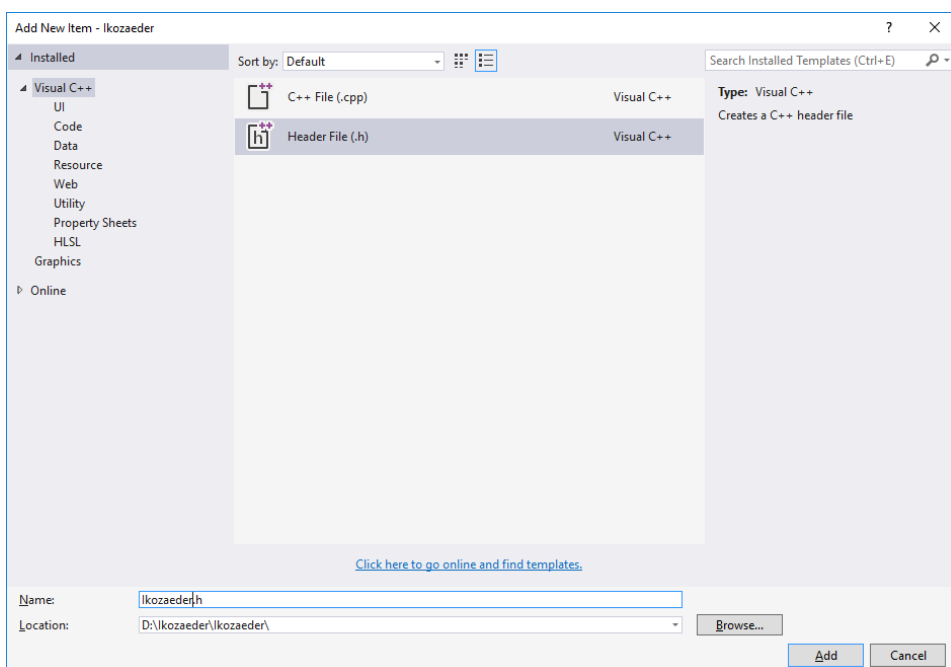
6. ábra: Megoldás és beállítások ablak

A program kódjának forrás és **header** fájllai (**Source Files, Header Files**) a megoldás ablakban, a kiválasztott fájl típusra az egér jobb gombjával kattintva, adható hozzá, az alábbi ábrának megfelelően vagy a **Shift+A** billentyűkombináció megnyomásával.



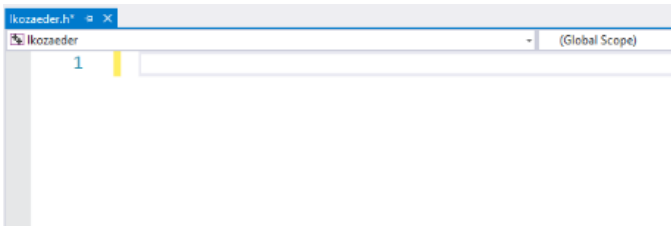
7. ábra: Új header fájl hozzáadása a projekthez

Új header fájl (**Header File**) választásával és a név (**Name**) az **Ikozaeder.h** megadásával hozható létre az **Ikozaeder.h** fájl, az alábbi ábrának megfelelően.



8. ábra: Új header fájl hozzáadása a projekthez, név megadása

Az új fájl hozzáadásával a képernyő középső részén megjelenik a forráskód szerkesztő az alábbi ábrának megfelelően.



9. ábra: Forráskód szerkesztő

Ikozaeder.h

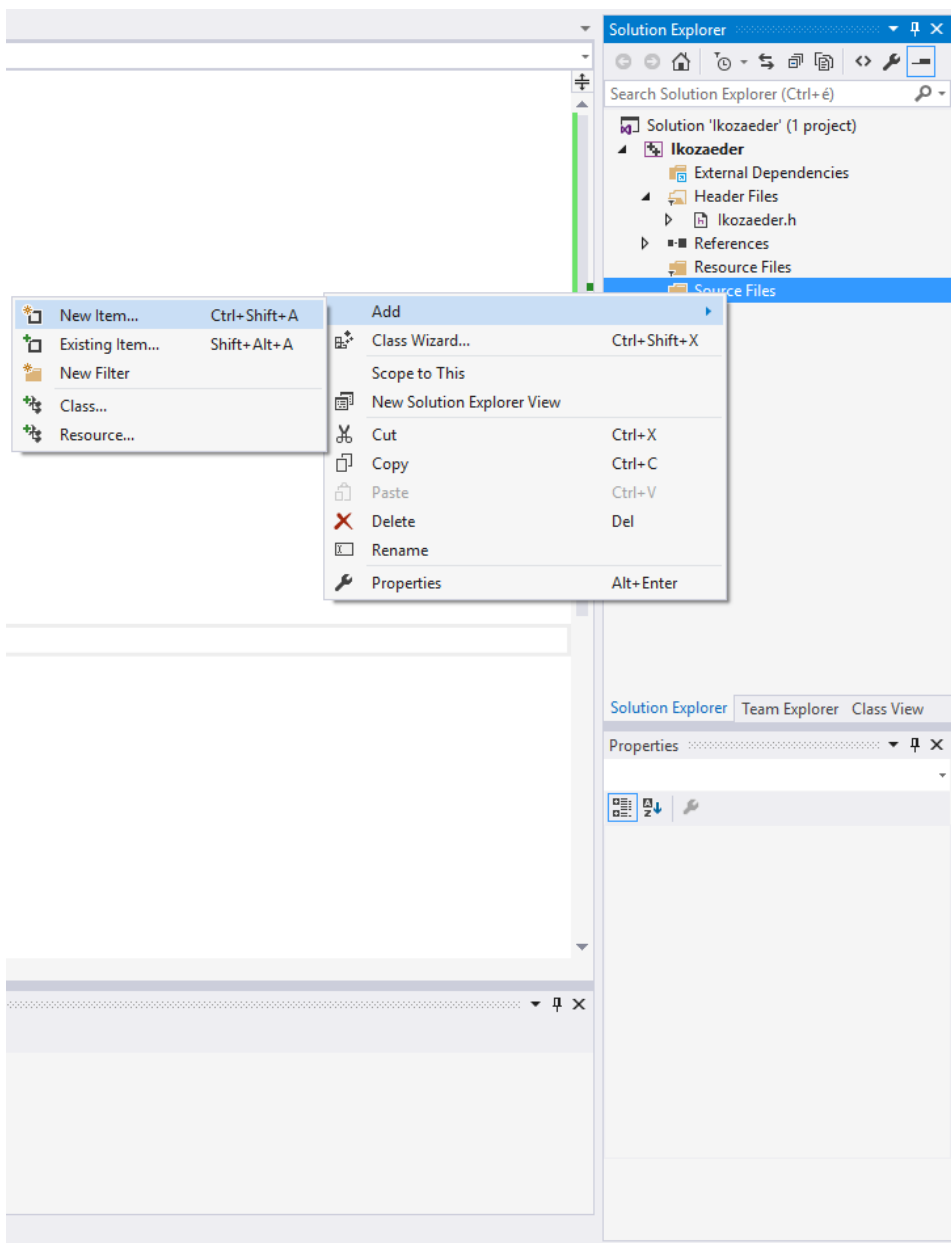
```
#ifndef IKOZAEDER_H
#define IKOZAEDER_H

template <class T> class Ikozaeder
{
private:
    T *a;

public:
    Ikozaeder(T a);
    double FelszinSzamol();
    double TerfogatSzamol();
    double KoreRajzoltGombSugaraSzamol();
    double BeirtGombSugaraSzamol();
    void Kiir();
    ~Ikozaeder(void);
};

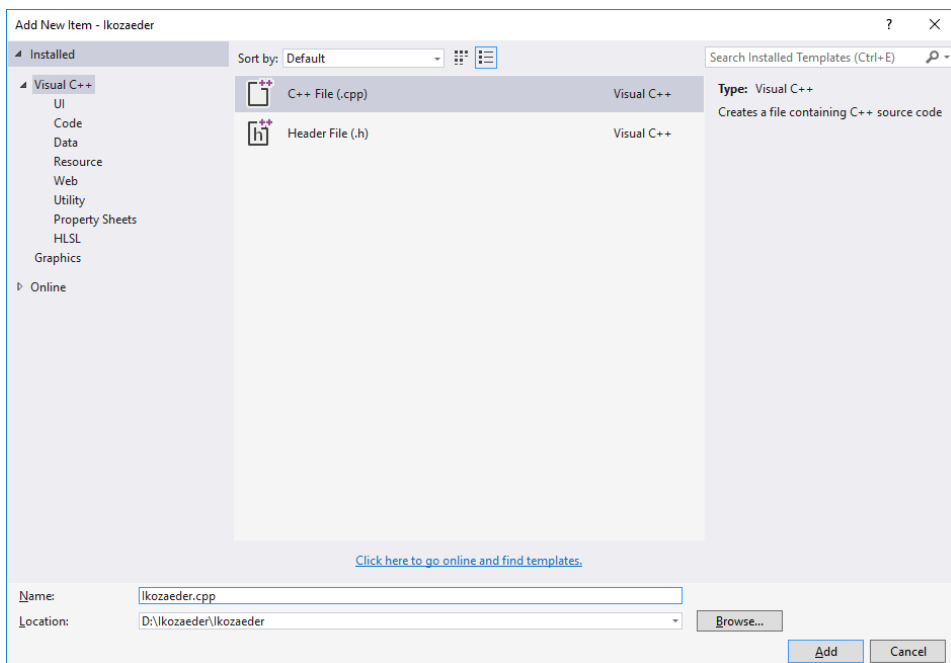
#endif
```

A deklarált osztályok megvalósításait az **Ikozaeder.cpp** forrásfájl tartalmazza, melyeket a **header** fájlokhoz hasonlóan, de forrásfájl (**Source File**) hozzáadásával hozhatunk létre.



10. ábra: Új forrásfájl hozzáadása a projekthez

Új C++ fájl (**C++ File**) választásával és a név (**Name**) az **Ikozaeder.cpp** megadásával hozható létre.



11. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

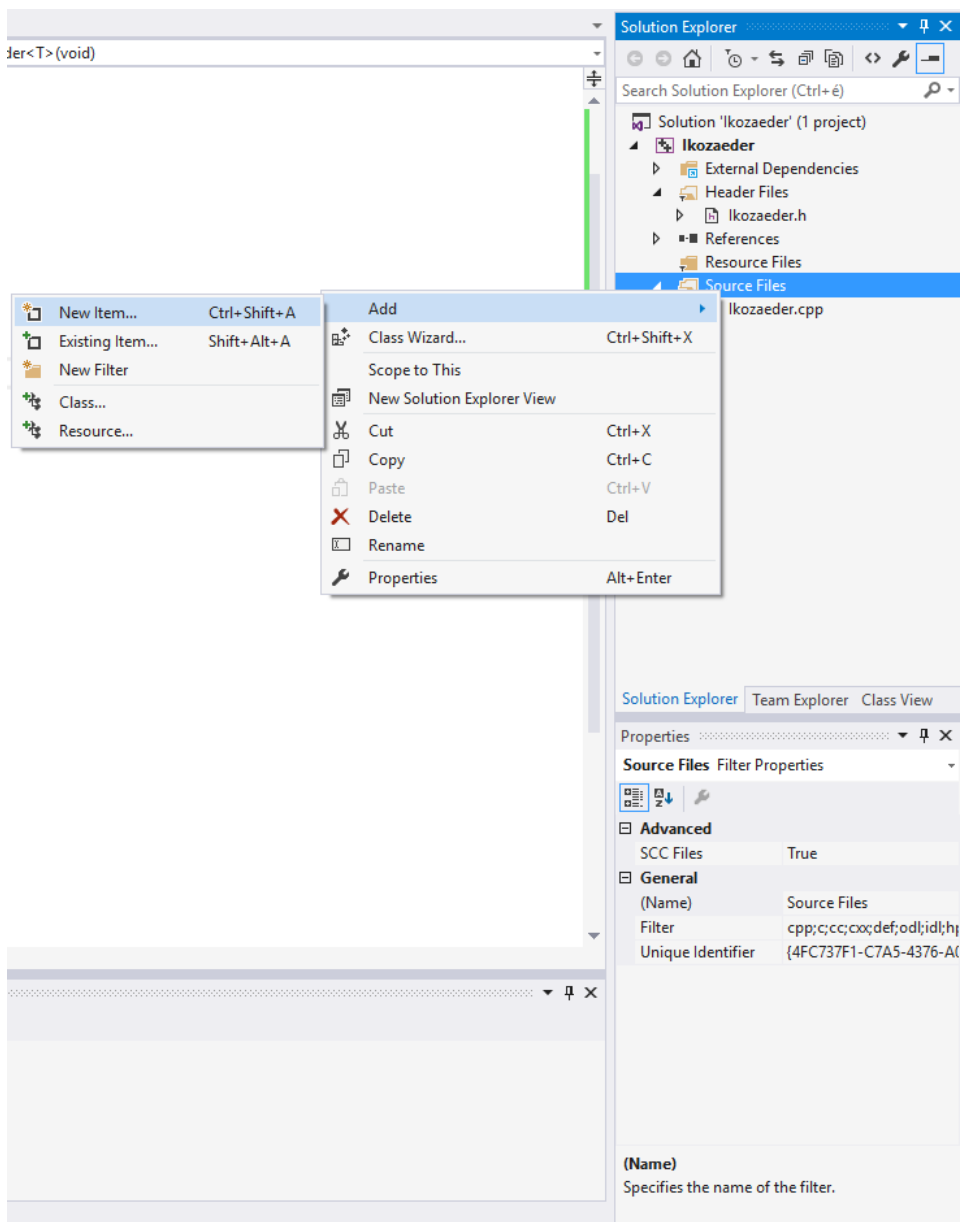
Ikozaeder.cpp

```
#include <iostream>
#include <string>
#include "Ikozaeder.h"

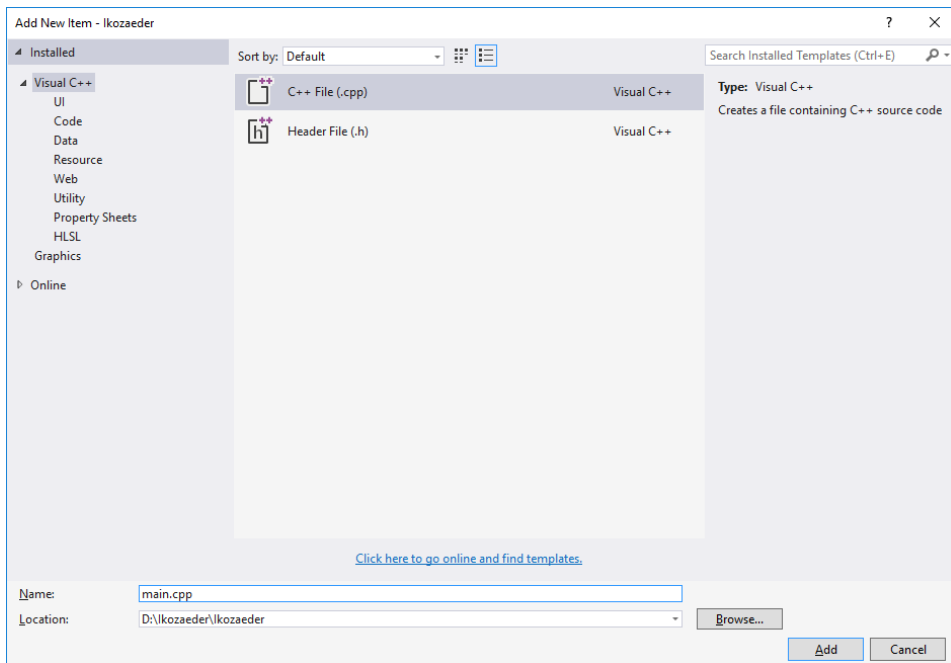
using namespace std;

template <class T> Ikozaeder<T>::Ikozaeder(T a)
{
    this->a = new T(a);
}
template <class T> double Ikozaeder<T>::FelszinSzamol()
{
    return 5.0 * sqrt(3.0) * pow(*a, 2.0);
}
template <class T> double Ikozaeder<T>::TerfogatSzamol()
{
    return 5.0 / 12.0 * (3.0 + sqrt(5)) * pow(*a, 3.0);
}
template <class T> double Ikozaeder<T>::KoreRajzoltGombSugaraSzamol()
{
    return *a / 4.0 * sqrt(10 + 2 * sqrt(5.0)) **a;
}
template <class T> double Ikozaeder<T>::BeirtGombSugaraSzamol()
{
    return *a / 12.0 * sqrt(3.0) * (3.0 + sqrt(5.0)) **a;
}
template <class T> void Ikozaeder<T>::Kiir()
{
    cout << "A(z) " << *a << " cm elhosszusagu, szabalyos ikozaeder  
felszine: " << FelszinSzamol() << " cm^2" << endl;
    cout << "A(z) " << *a << " cm elhosszusagu, szabalyos ikozaeder  
terfogata: " << TerfogatSzamol() << " cm^3" << endl;
    cout << "A(z) " << *a << " cm elhosszusagu, szabalyos ikozaeder kore  
rajzolt gomb sugara: " << KoreRajzoltGombSugaraSzamol() << " cm" <<  
endl;
    cout << "A(z) " << *a << " cm elhosszusagu, szabalyos ikozaederbe beirt  
gomb sugara: " << BeirtGombSugaraSzamol() << " cm" << endl;
}
template <class T> Ikozaeder<T>::~Ikozaeder(void)
{
    delete a;
}
```

A main függvény fájlja a C++ fájl (**C++ File**) választásával és a név (**Name**) a **main.cpp** megadásával hozható létre.



12. ábra: Új forrásfájl hozzáadása a projekthez



13. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

main.cpp

```
#include <iostream>
#include <conio.h>
#include "Ikozaeder.cpp"

int main()
{
    cout << "Objektumpeldany double tipussal" << endl;
    Ikozaeder<double> *IkozaederObjektumDuplaPontossaguLebegopontosTipussal
    = new Ikozaeder<double>(10.1);
    IkozaederObjektumDuplaPontossaguLebegopontosTipussal->Kiir();
    delete IkozaederObjektumDuplaPontossaguLebegopontosTipussal;

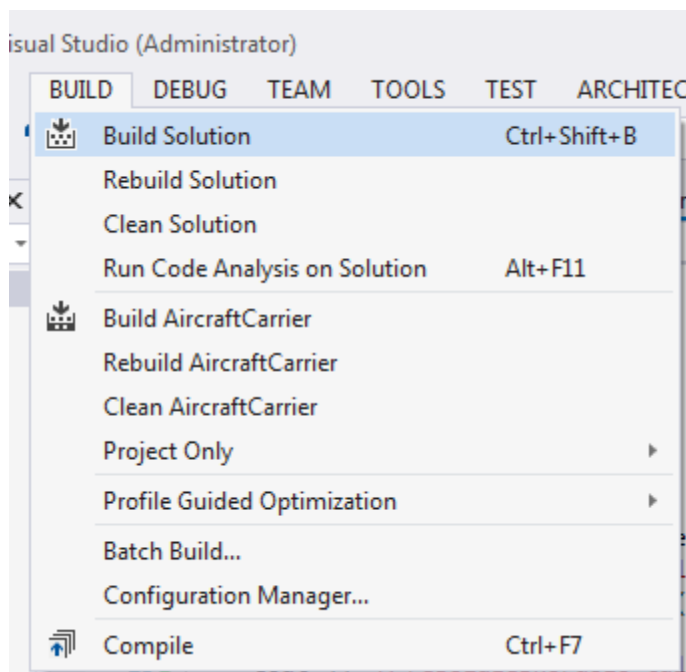
    cout << endl;
    cout << "Objektumpeldany float tipussal" << endl;
    Ikozaeder<float> *IkozaederObjektumValosTipussal = new
    Ikozaeder<float>(13.5);
    IkozaederObjektumValosTipussal->Kiir();
    delete IkozaederObjektumValosTipussal;

    cout << endl;
    cout << "Objektumpeldany short tipussal" << endl;
    Ikozaeder<short> *IkozaederObjektumRovidEgeszTipussal = new
    Ikozaeder<short>(7);
    IkozaederObjektumRovidEgeszTipussal->Kiir();
    delete IkozaederObjektumRovidEgeszTipussal;

    cout << endl;
    cout << "Objektumpeldany integer tipussal" << endl;
    Ikozaeder<int> *IkozaederObjektumEgeszTipussal = new Ikozaeder<int>(25);
    IkozaederObjektumEgeszTipussal->Kiir();
    delete IkozaederObjektumEgeszTipussal;

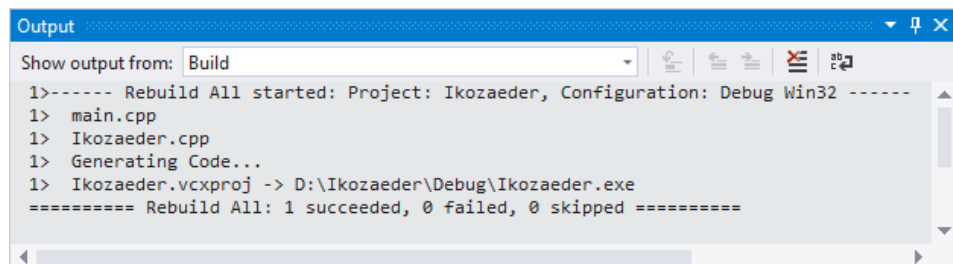
    _getch();
    return 0;
}
```

A forrásfájlok hozzáadása után a program forráskódja lefordítható, futtatható kód előállítását a **BUILD** menüpont **Build Solution** almenüpontjának kiválasztásával, vagy a **Ctrl+Shift+B** gyorsbillentyű kombináció megnyomásával, az alábbi ábrának megfelelően.



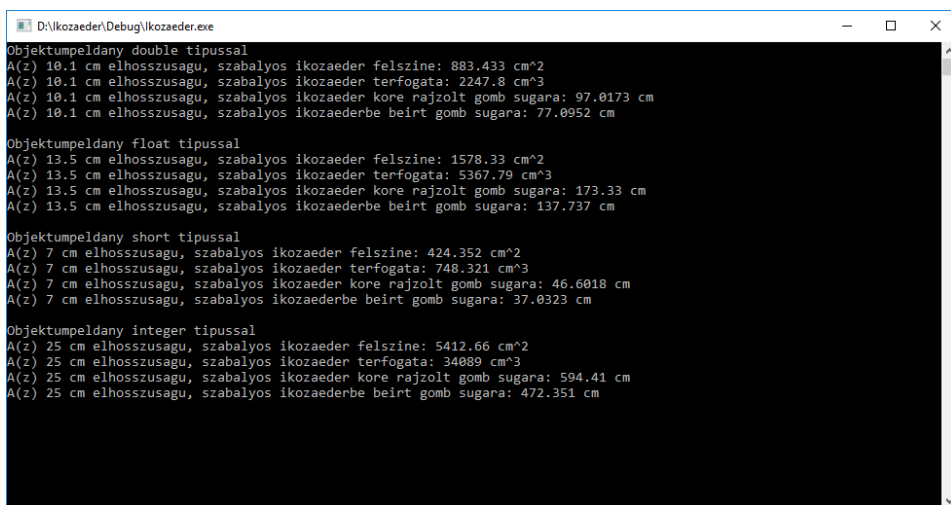
14. ábra: Program fordítása

Sikeres fordítás a kimenet (**Output**) ablakban látható.



15. ábra: Kimenet ablak

A Start-ra kattintva a program futásának eredménye konzolban jelenik meg, amely a következő ábrán látható.



```
D:\Ikozaeder\Debug\Ikozaeder.exe
Objektpeldany double tipussal
A(z) 10.1 cm elhosszusagu, szabalyos ikozaeder felszine: 883.433 cm^2
A(z) 10.1 cm elhosszusagu, szabalyos ikozaeder terfogata: 2247.8 cm^3
A(z) 10.1 cm elhosszusagu, szabalyos ikozaeder kore rajzolt gomb sugara: 97.0173 cm
A(z) 10.1 cm elhosszusagu, szabalyos ikozaederbe beirt gomb sugara: 77.0952 cm

Objektpeldany float tipussal
A(z) 13.5 cm elhosszusagu, szabalyos ikozaeder felszine: 1578.33 cm^2
A(z) 13.5 cm elhosszusagu, szabalyos ikozaeder terfogata: 5367.79 cm^3
A(z) 13.5 cm elhosszusagu, szabalyos ikozaeder kore rajzolt gomb sugara: 173.33 cm
A(z) 13.5 cm elhosszusagu, szabalyos ikozaederbe beirt gomb sugara: 137.737 cm

Objektpeldany short tipussal
A(z) 7 cm elhosszusagu, szabalyos ikozaeder felszine: 424.352 cm^2
A(z) 7 cm elhosszusagu, szabalyos ikozaeder terfogata: 748.321 cm^3
A(z) 7 cm elhosszusagu, szabalyos ikozaeder kore rajzolt gomb sugara: 46.6018 cm
A(z) 7 cm elhosszusagu, szabalyos ikozaederbe beirt gomb sugara: 37.0323 cm

Objektpeldany integer tipussal
A(z) 25 cm elhosszusagu, szabalyos ikozaeder felszine: 5412.66 cm^2
A(z) 25 cm elhosszusagu, szabalyos ikozaeder terfogata: 34089 cm^3
A(z) 25 cm elhosszusagu, szabalyos ikozaeder kore rajzolt gomb sugara: 594.41 cm
A(z) 25 cm elhosszusagu, szabalyos ikozaederbe beirt gomb sugara: 472.351 cm
```

16. ábra: A program eredménye

A feladathoz kapcsolódó magyarázatok

Az osztály deklarációját az (**Ikozaeder.h**) állományban adjuk meg. A szükséges paramétereket osztály(ok), adattag(ok) és tagfüggvény(ek) elé kell beszúrni, valamint a paraméterek típusát is jelölni kell. A feladat megvalósítása során az osztály deklarációban egy magán (**private**) jellegű adattag-mutató kerül megfogalmazásra, amely az ikozaéder él hosszúság értékét tartalmazza. A konstruktor paraméteres formában jelenik meg, amely egy darab *T* típusú adatot vár paraméterül (él hossz érték). A *FelszinSzamol()*, *TerfogatSzamol()*, *KoreRajzoltGombSugaraSzamol()* és *BeirtGombSugaraSzamol()* tagfüggvények **double** típusú számolt adatokkal térnek vissza, paraméterlistájuk üres. A *Kiir()* tagfüggvény visszatérési típusa **void**, amely jelentése szerint a függvény nem tér vissza adott típusú értékkel, továbbá megfogalmazásra került a(z) *~Ikozaeder(void)* nevezetű, üres paraméterlistával rendelkező destruktork, amely az erőforrások felszabadításáért lesz felelős.

Az (**Ikozaeder.cpp**) állomány tartalmazza az osztály megvalósítását, tagfüggvényeinek implementálását. A tagfüggvények neve előtt a **::** hatókör operátor kerül felhasználásra azon célból, hogy egyértelműsítésre kerüljön melyik osztály tagfüggvényét szeretnénk kifejteni. A konstruktor felelőssége az osztály adattag-mutatójának definiálása, memória helyének lefoglalása, amely a **new** operátor segítségével történik. Az osztály implementációja során adott számolást végző tagfüggvények felelőssége a rájuk szabott műveletek kiszámítása és a számított érték visszaadása, a számolás során a gyökvonás implementálásához az

std névtérben található **sqrt()** nevezetű függvényt használjuk, amely megfelelő működéséhez egy **double** típusú számértéket vár paraméterül, majd annak négyzetgyök értékével tér vissza **double** típusként. A hatványértékek meghatározásához szintén az **std** névtérben fellelhető **pow()** függvény került felhasználásra, amely paraméterül vár egy hatványalapot, illetve egy kitevőértéket, majd **double** típusként adja vissza a számított értéket.

A konzolon történő megjelenítésért felelős függvény feladata a szöveges tartalom megjelenítése, ahol kiírásra kerül az ikozaéder számított felszíne, térfogata, köré rajzolt gömb sugara, illetve a beírható gömb sugara. Az osztály implementáció végén a destruktork **delete** operátor segítségével többek között az adattag-mutató számára lefoglalt memória területet szabadítja fel.

A **(main.cpp)** állomány tartalmazza a **main()** függvényt, a program belépési pontját. A főfüggvényben történő dinamikus példányosítása után hívjuk meg a szükséges függvényeket. A példányosítás során átadásra kerülnek azon típusok (**double**, **float**, **short** és **int**), amelyekkel a műveletek elvégzésre kerülnek, majd gondoskodunk a lefoglalt erőforrások felszabadításáról, a destruktork hívásának segítségével.

Függvénynevek túlterhelése, Függvénytípusok

A műsorozat első két kötetben használt „konvencionális” függvények esetében a futtató környezet a paraméterértékeket a megfelelő (formális) helyre helyettesítette.

Eddig, ha adott volt például egy rendezési feladat, amelynek célja volt különböző típusú adatok rendezése, akkor minden rendezési algoritmus esetén új függvényt kellett implementálnunk. Ilyen esetekben jól megfigyelhető volt, hogy lényegében a függvény paraméterlistáján kívül semmi nem változott, sem a törzs, adott esetekben még a függvény neve is maradhatott volna változatlanul, mivel egyértelműen utalt a feladatára.

A fentebbi „probléma” áthidalására a C++ több ízben is lehetőséget kínál. Az egyik ilyen – bizonyos szintig – elegánsabb megoldás a függvénynevek túlterhelése, ahol is kizárólag a függvény paraméterlistája (paraméterszignatúrája) változik, az azonosítója és törzse változatlan marad. Ilyen esetekben a fordítóprogram nem a függvény azonosítója alapján hív meg egy adott függvényt, hanem annak, paraméterlistájának függvényeként. A másik sokkal hatékonyabb lehetőség az úgynevezett függvénytípusok használata, ahol is elég a függvényt egyszer megírni és a továbbiakban az bármilyen paramétertípussal el fog boldogulni. Azt azonban figyelembe kell venni, hogy a fordítóprogram az ilyen típusú függvényeket kizárólag akkor készíti el (fordítási vagy, akár futási időben), ha már egyértelműsíthető a konkrét paramétertípus.

Fontos megjegyezni, hogy a függvénytípusok minden esetben átírhatók függvénytúlterheléssé, illetve az osztálysablonok típusokkal nem paraméterezett osztályokká. A következő példák jól prezentálják a sablonok előnyeit.

Minta példa „általánosan alkalmazott” függvények használatára Visual Studio Community fejlesztőkörnyezetben

Írjunk egy **Sort** nevezetű osztályt, amely néhány rendezési algoritmust implementál. A rendezési műveleteket egy száz elemű, egész típusú, 1 és 100 közötti véletlen számokkal feltöltött tömbön végezzük el, úgy, hogy közben a rendezési algoritmus hatékonyságát is prezentáljuk a rendezési művelethez szükséges idővel. A képernyőn jelenítsük meg a kiinduló és végállapotokat, valamint a rendezéshez szükséges időtartamot mikromásodpercben. Az osztály adatai a következők:

- **SIZE** nevezetű statikus, konstans, egész típusú érték, amelyet 100-al inicializálunk.

Ügyeljünk az egységbezárás elvére, tehát az adattag(ok) **private** elérhetőségűek legyenek!

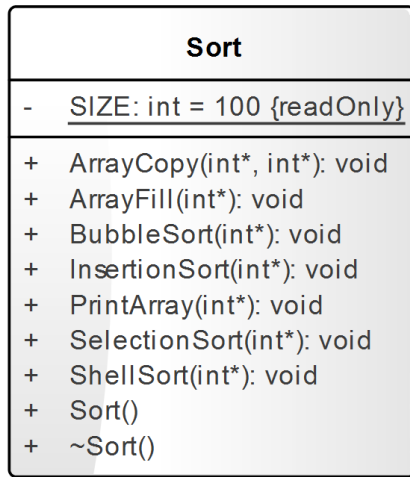
Valósítsuk meg az osztály metódusait:

- Írjuk meg az osztály explicit, alapértelmezett konstruktorát.
- Írjunk egy *ArrayFill(int *numbers)* függvényt, amelynek felelőssége egy 100 elemű tömb, véletlen számokkal (1 és 100 között) történő feltöltése.
- Írjunk egy *ArrayCopy(const int *numbers, int *copiedNumbers)* függvényt, amely felelőssége az eredeti tömb átmásolása egy átmeneti tömbbe. Ennek célja, hogy a rendezési algoritmusok hatékonyságát ne torzítsa, mivel, ha mindig újra inicializálnánk a rendezendő tömböt, akkor folyamatosan eltérő értékeket tartalmazna, ezáltal megnövelve vagy csökkentve a rendezéshez szükséges időtartamot, így szükséges minden egyes rendezési algoritmusnál ugyanazokkal a tömb értékekkel dolgozni.
- Írjunk egy *BubbleSort(int *copiedNumbers)* függvényt, amely a buborékos rendezés algoritmusát valósítja meg.
- Írjunk egy *InsertionSort(int *copiedNumbers)* függvényt, amely a beszűrő rendezés algoritmusát valósítja meg.
- Írjunk egy *SelectionSort(int *copiedNumbers)* függvényt, amely a közvetlen kiválasztásos rendezés algoritmusát valósítja meg.
- Írjunk egy *ShellSort(int *copiedNumbers)* függvényt, amely a Shell-rendezés algoritmusát valósítja meg.
- Írjunk egy *PrintArray(int *copiedNumbers)* függvényt, amelynek felelőssége a paraméterül kapott tömb értékeinek megjelenítése a képernyőn, adott formátumban.
- Ne feledkezzünk meg az objektumpéldány által lefoglalt erőforrás felszabadításáról, a destruktork megírásáról sem.

Az osztály implementálását követően dinamikus módszerrel (a memória dinamikus területén [**heap**]) készítsünk egy objektumpéldányt. Az objektumpéldány segítségével a feladat leírásnak megfelelően hívjuk meg az

osztály egyes tagfüggvényeit. A megvalósítás során, ha tudjuk, hogy a függvénynek átadott paraméter a függvény futása során nem változik, használjunk **const** jelzőt. Amennyiben nincs tovább szükség az objektumpéldányra, gondoskodjunk annak megszüntetéséről, elkerülve ezzel a memóriaszivargást. Mielőtt a feladatnak neki látnánk, tervezzük meg a statikus osztály modellt.

Statikus tervezés

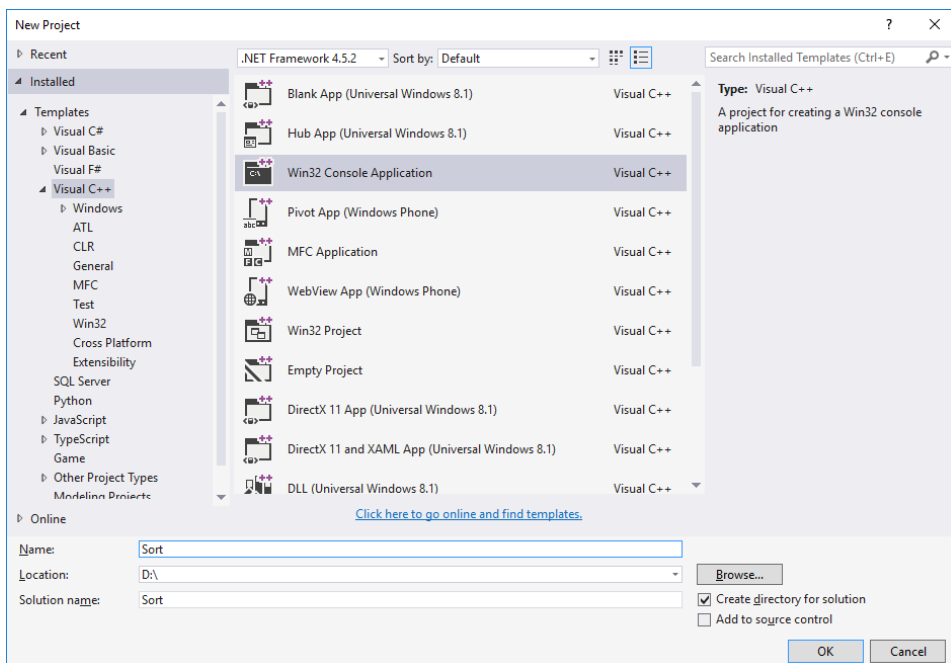


17. ábra: Osztály diagram

A projekt létrehozása

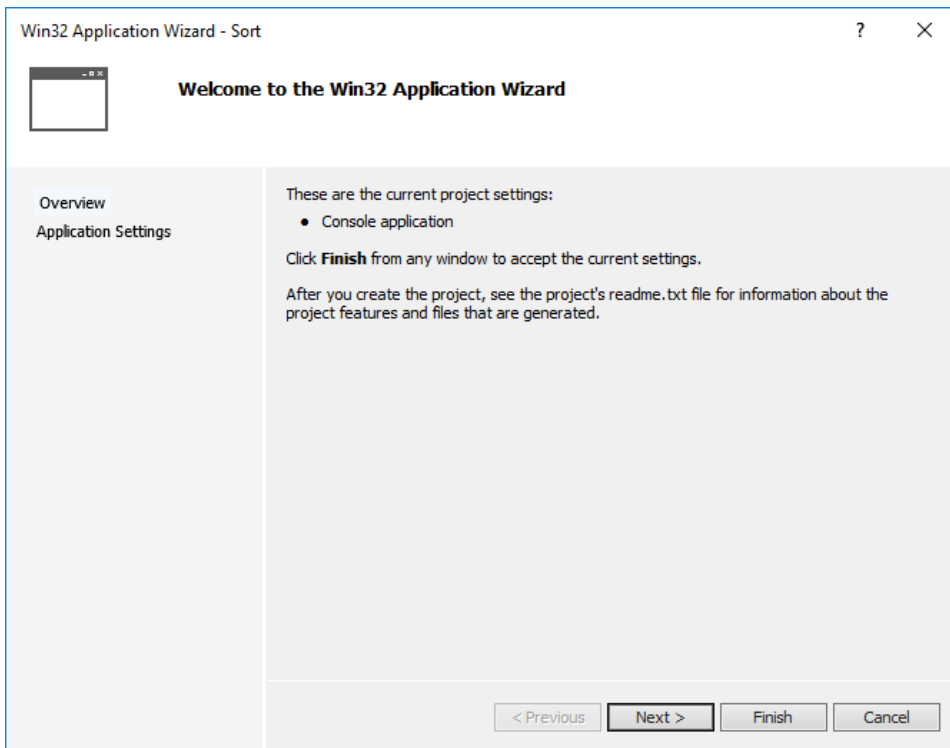
A projekt megvalósítása során konzolalkalmazást készítünk, ahol a Visual Studio, szabványos C++-nak megfelelő szintaktikát alkalmaz. A projekt három fájlban kerül implementálásra. Az alkalmazás belépési pontja a **main** függvény, amely a három fájl közül az egyikben fog helyet foglalni. Az előzőekben megfogalmazott minta példa kerül implementálásra és összeállításra, továbbá Windows operációs rendszeren futtatható kóddá történő lefordításra **Visual Studio Community** fejlesztőkörnyezet felhasználásával. A lefordított 32 bites alkalmazás konzolban kerül futtatásra.

Új projekt a **File** menü **New** menüpontjában hozható létre, a **Project** almenüt kiválasztva, vagy akár a **Ctrl+Shift+N** gyorsindító gombkombinációval. Új projekt kiválasztása után megjelenő ablakban a közül a **Visual C++ Win32** sablont (**Template**) kell kiválasztani és a **Win32 Console Application**-t, mely az alábbi ábrán látható.



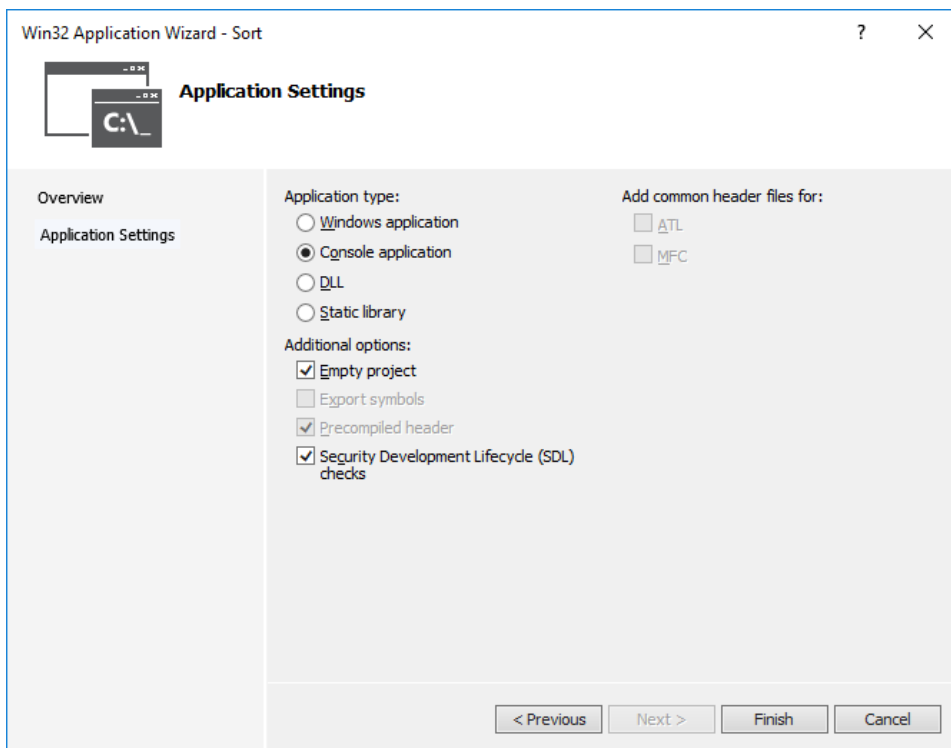
18. ábra: Win32 Console Application projekt létrehozása

Az ablak alsó részén a projekt nevének (**Name**) a példának megfelelően az **Sort** név került megadásra. Ebben a részben lehet módosítani a Projekthez kapcsolódó fájlok mentési helyét is (**Location**). Egy összetett program általában több projektből tevődik össze, ezeket egységes rendszerre a megoldás (**Solution**) fogja össze. Az OK gombra kattintást követően elindul az alkalmazás varázsló, mely a következő ábrán látható.



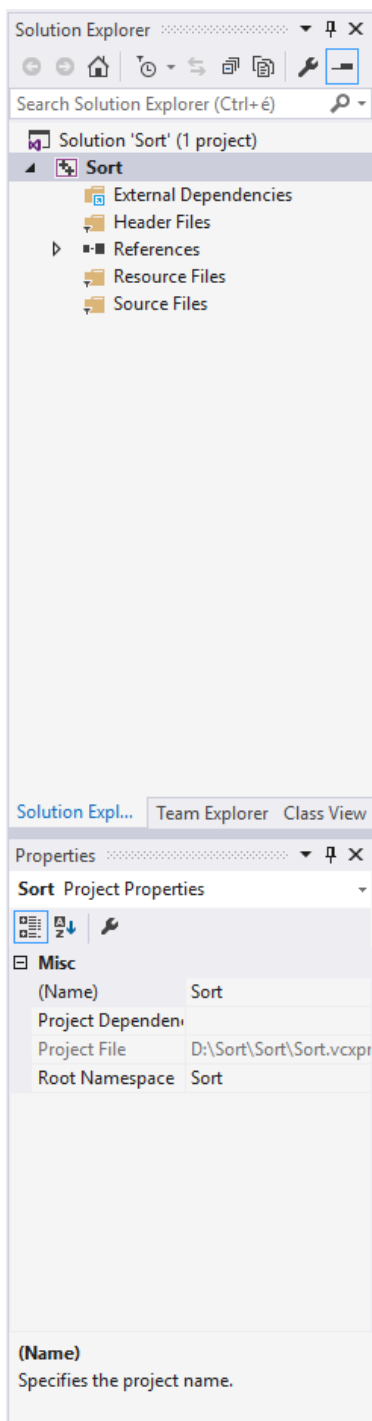
19. ábra: Win32 konzol alkalmazás varázsló

A **Next** gombra kattintva a beállítások ablak jelenik meg, itt megadhatjuk az alkalmazás típusát (**Console Application**), és néhány kiegészítő opciót. Mivel egyszerű alkalmazást készítünk, ezért célszerű az üres projekt (**Empty project**) opció kiválasztása, ahogy az alábbi képen látszódik.



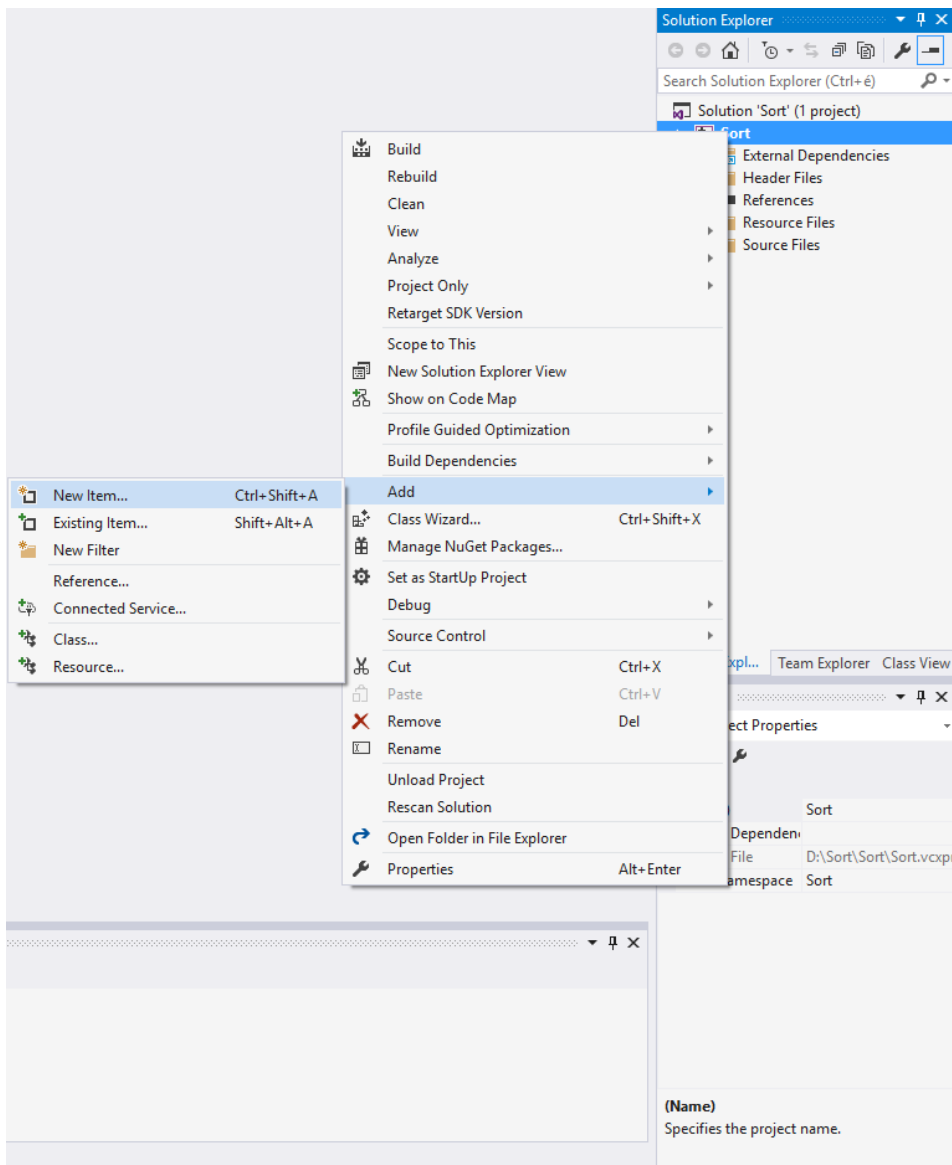
20. ábra: Alkalmazás beállításai

A **Finish** megnyomását követően a képernyő jobb szélén, a megoldás ablaka (**Solution Explorer**), alatta a beállítások ablak jelenik meg, ahogy a következő ábrán látható.



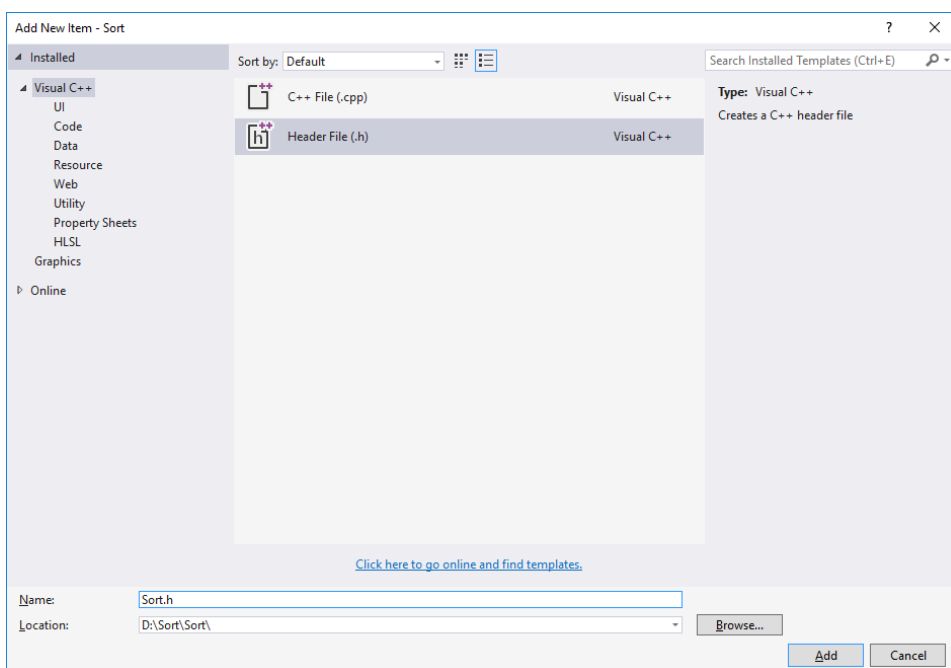
21. ábra: Megoldás és beállítások ablak

A program kódjának forrás és **header** fájljait (**Source Files, Header Files**) a megoldás ablakban a kiválasztott fájl típusra az egér jobb gombjával kattintva adható hozzá az alábbi ábrának megfelelően vagy a **Shift+A** billentyűkombináció megnyomásával.



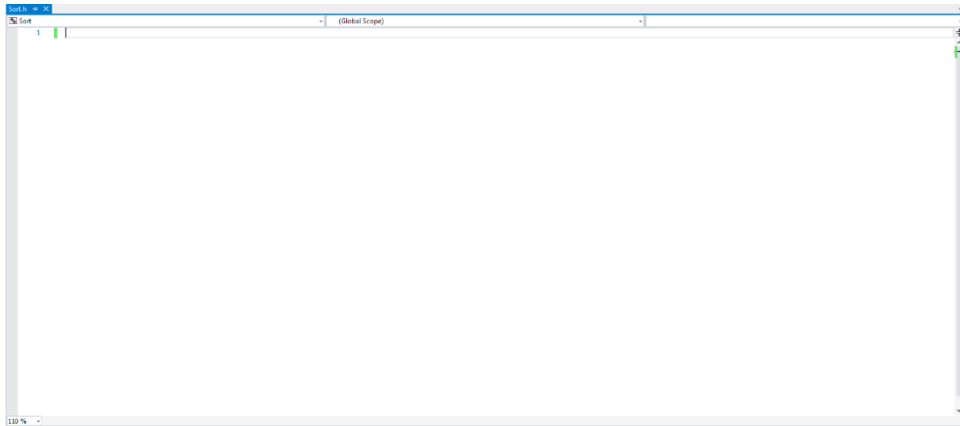
22. ábra: Új header fájl hozzáadása a projekthez

Új **header** fájl (**Header File**) választásával és a név (**Name**) az **Sort.h** megadásával hozható létre a **Sort.h** fájl, az alábbi ábrának megfelelően.



23. ábra: Új header fájl hozzáadása a projekthez, név megadása

Az új fájl hozzáadásával a képernyő középső részén megjelenik a forráskód szerkesztő az alábbi ábrának megfelelően.



24. ábra: Forráskód szerkesztő

Sort.h

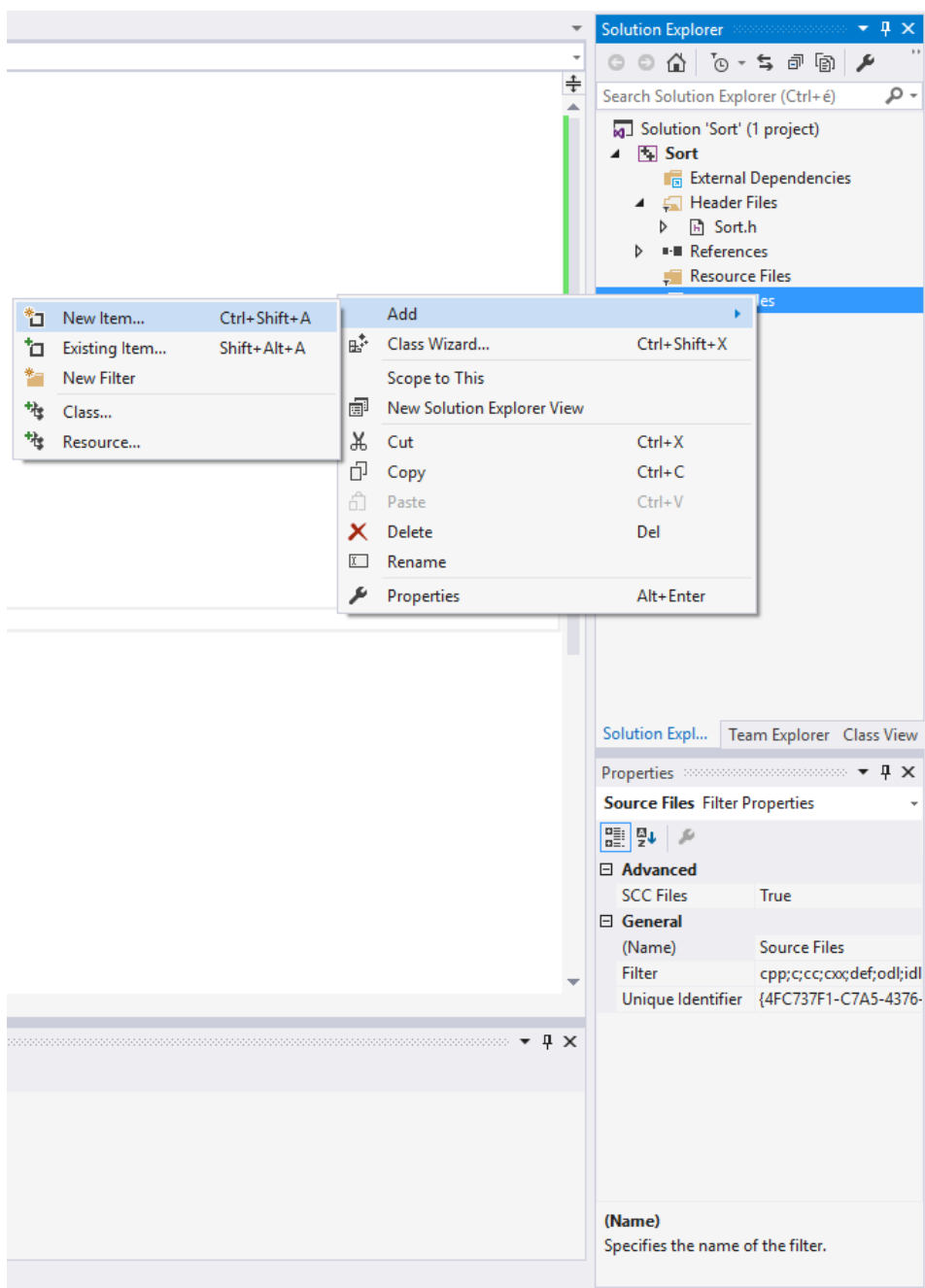
```
#ifndef SORT_H
#define SORT_H

class Sort
{
    private:
        static const int SIZE = 100;

    public:
        Sort() { }
        void ArrayFill(int *numbers);
        void ArrayCopy(const int *numbers, int *copiedNumbers) { for (int
i = 0; i < SIZE; ++i) copiedNumbers[i] = numbers[i]; }
        void BubbleSort(int *copiedNumbers);
        void SelectionSort(int *copiedNumbers);
        void InsertionSort(int *copiedNumbers);
        void ShellSort(int *copiedNumbers);
        void PrintArray(const int *copiedNumbers);
        ~Sort() { }
};

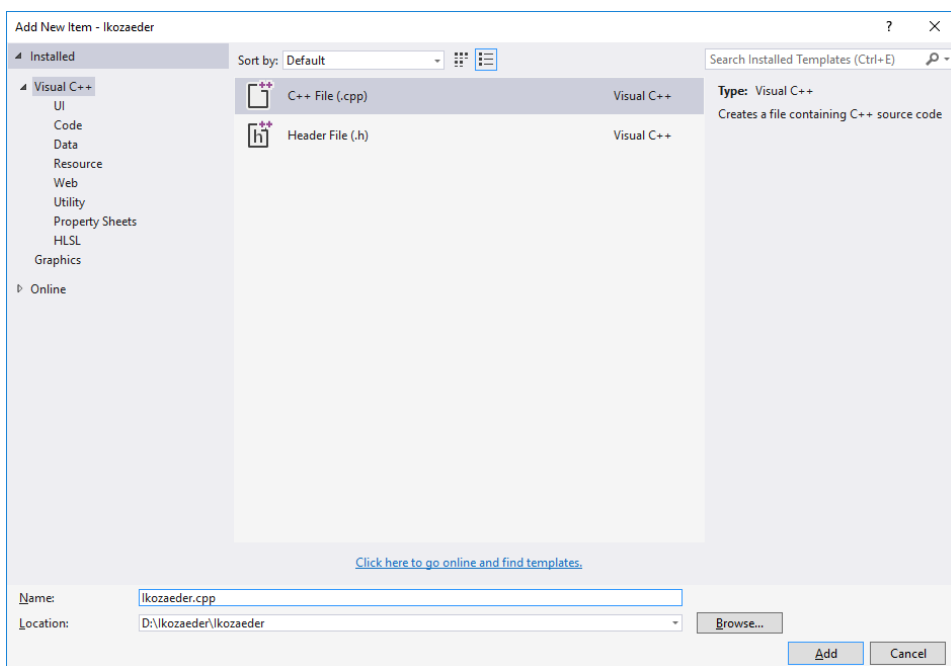
#endif
```

A deklarált osztályok megvalósításait a **Sort.cpp** forrásfájl tartalmazza, melyeket a **header** fájlokhoz hasonlóan, de a forrásfájl (**Source File**) hozzáadásával hozhatunk létre.



25. ábra: Új forrásfájl hozzáadása a projekthez

Új C++ fájl (**C++ File**) választásával és a név (**Name**) a **Sort.cpp** megadásával hozható létre.



26. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

Sort.cpp

```
#include <iostream>
#include <ctime>
#include <chrono>
#include "Sort.h"

using namespace std;

void Sort::BubbleSort(int *copiedNumbers)
{
    cout << "Buborekrendezes" << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(copiedNumbers);

    int temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int i = 0; i < SIZE; i++)
    {
        for (int j = SIZE - 1; j > i; j--)
        {
            if (copiedNumbers[j] < copiedNumbers[j - 1])
            {
                temp = copiedNumbers[j - 1];
                copiedNumbers[j - 1] = copiedNumbers[j];
                copiedNumbers[j] = temp;
            }
        }
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(copiedNumbers);
    cout << "Rendezesi ido: " <<
        chrono::duration_cast<chrono::microseconds>(end - start).count() << "
        mikromasodperc [us]";
    cout << endl; cout << endl;
}

void Sort::InsertionSort(int *copiedNumbers)
{
    cout << "Beszuro rendezes: " << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(copiedNumbers);

    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int j = 1; j < SIZE; j++)
    {
        int key = copiedNumbers[j];
        int i = j - 1;
        while (i > -1 && copiedNumbers[i] > key)
        {
            copiedNumbers[i + 1] = copiedNumbers[i];
            i = i - 1;
        }
        copiedNumbers[i + 1] = key;
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(copiedNumbers);
    cout << "Rendezesi ido: " <<
```

```

        chrono::duration_cast<chrono::microseconds>(end - start).count() << "
        mikromasodperc [us]";
        cout << endl; cout << endl;
    }
    void Sort::SelectionSort(int *copiedNumbers)
    {
        cout << "Kozvetlen kivalaszto rendezes" << endl;
        cout << "===== " << endl;
        cout << "Kiindulo allapot =>"; PrintArray(copiedNumbers);

        int position, temp;

        chrono::steady_clock::time_point start = chrono::steady_clock::now();
        for (int i = 0; i < SIZE - 1; i++)
        {
            position = i;
            for (int j = i + 1; j < SIZE; j++)
            {
                if (copiedNumbers[j] < copiedNumbers[position])
                    position = j;
            }

            if (position != i)
            {
                temp = copiedNumbers[i];
                copiedNumbers[i] = copiedNumbers[position];
                copiedNumbers[position] = temp;
            }
        }
        chrono::steady_clock::time_point end = chrono::steady_clock::now();
        cout << "Vegallapot => "; PrintArray(copiedNumbers);
        cout << "Rendezesi ido: " <<
        chrono::duration_cast<chrono::microseconds>(end - start).count() << "
        mikromasodperc [us]";
        cout << endl; cout << endl;
    }
    void Sort::ShellSort(int *copiedNumbers)
    {
        cout << "Shell-rendezes" << endl;
        cout << "===== " << endl;
        cout << "Kiindulo allapot =>"; PrintArray(copiedNumbers);

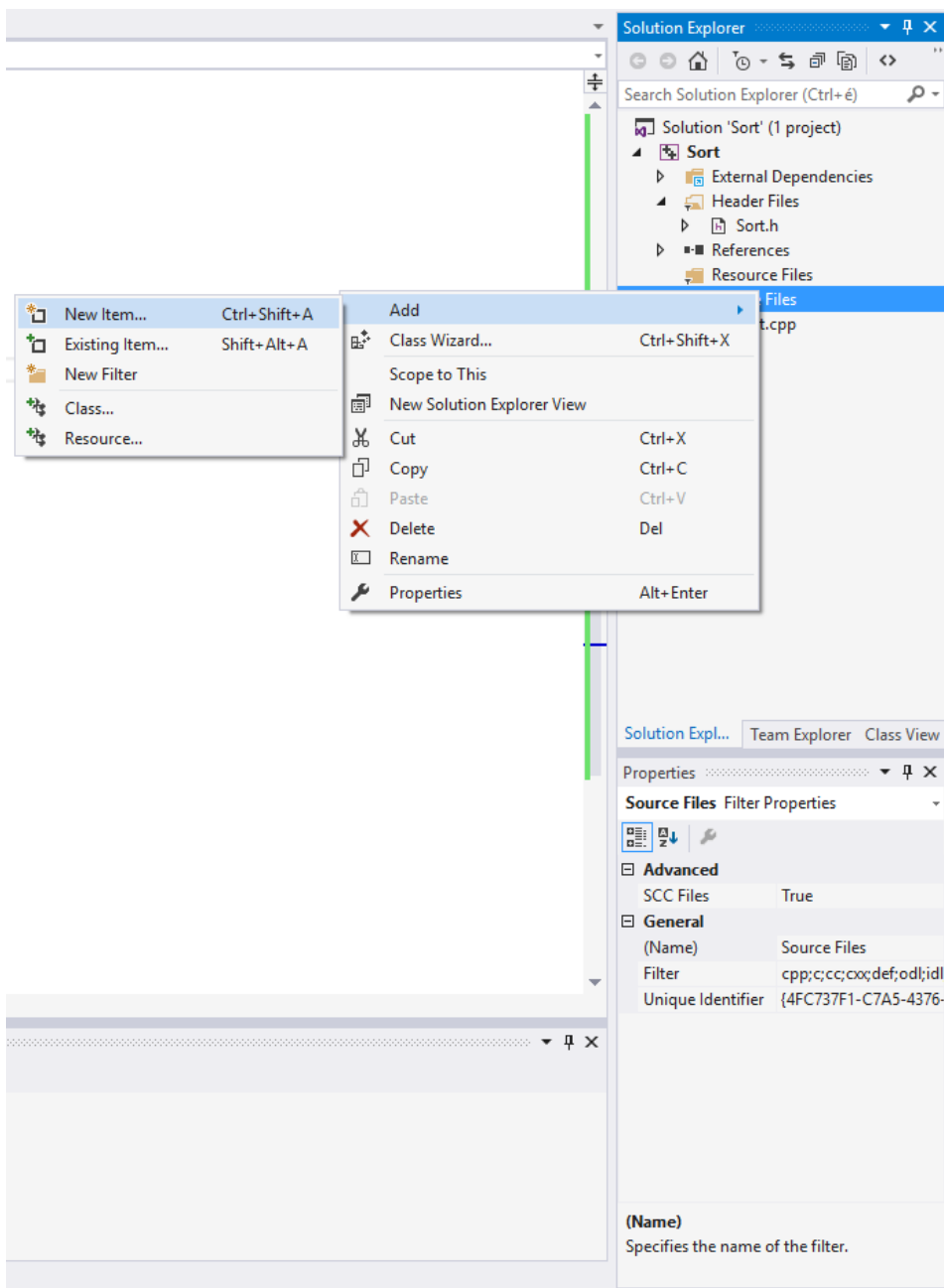
        int temp;
        chrono::steady_clock::time_point start = chrono::steady_clock::now();

        for (int gap = SIZE / 2; gap > 0; gap /= 2)
            for (int i = gap; i < SIZE; i++)
                for (int j = i - gap; j >= 0 &&
                    copiedNumbers[j]>copiedNumbers[j + gap]; j -= gap) {
                    temp = copiedNumbers[j];
                    copiedNumbers[j] = copiedNumbers[j + gap];
                    copiedNumbers[j + gap] = temp;
                }

        chrono::steady_clock::time_point end = chrono::steady_clock::now();
        cout << "Vegallapot => "; PrintArray(copiedNumbers);
        cout << "Rendezesi ido: " <<
        chrono::duration_cast<chrono::microseconds>(end - start).count() << "
        mikromasodperc [us]";
        cout << endl; cout << endl;
    }

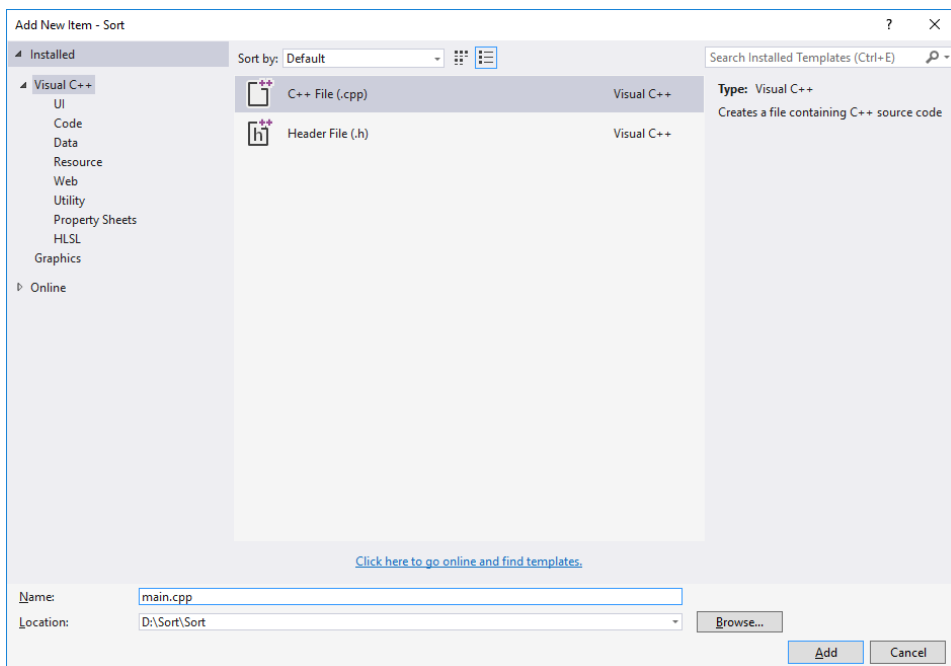
```

```
}  
void Sort::PrintArray(const int *copiedNumbers)  
{  
    cout << " { ";  
    for (int i = 0; i < SIZE; ++i)  
        cout << copiedNumbers[i] << " ";  
    cout << " } " << endl;  
}  
void Sort::ArrayFill(int *numbers)  
{  
    srand((unsigned)time(0));  
    for (int i = 0; i < SIZE; ++i)  
        numbers[i] = (rand() % 100) + 1;  
}
```



27. ábra: Új forrásfájl hozzáadása a projekthez

C++ fájl (**C++ File**) választásával és a név (**Name**) a **main.cpp** megadásával hozható létre.



28. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

main.cpp

```
#include <iostream>
#include <conio.h>
#include <ctime>
#include "Sort.h"

using namespace std;

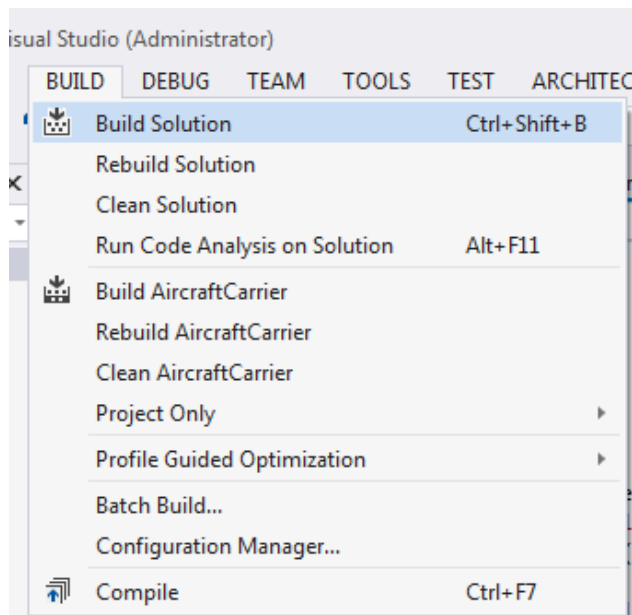
int main()
{
    const int SIZE(100);
    int *numbers = new int[SIZE];
    int *copiedNumbers = new int[SIZE];
    Sort *objectOfSortArray = new Sort();

    objectOfSortArray->ArrayFill(numbers);
    objectOfSortArray->ArrayCopy(numbers, copiedNumbers);
    objectOfSortArray->BubbleSort(copiedNumbers);
    objectOfSortArray->ArrayCopy(numbers, copiedNumbers);
    objectOfSortArray->InsertionSort(copiedNumbers);
    objectOfSortArray->ArrayCopy(numbers, copiedNumbers);
    objectOfSortArray->SelectionSort(copiedNumbers);
    objectOfSortArray->ArrayCopy(numbers, copiedNumbers);
    objectOfSortArray->ShellSort(copiedNumbers);

    delete objectOfSortArray;
    delete[] copiedNumbers;
    delete[] numbers;

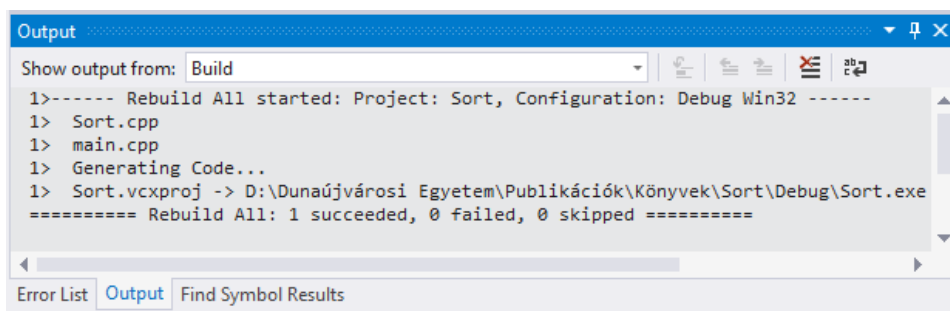
    _getch();
    return 0;
}
```

A forrásfájlok hozzáadása után a program forráskódja lefordítható, futtatható kód előállítását a **BUILD** menüpont **Build Solution** almenüpontjának kiválasztásával, vagy a **Ctrl+Shift+B** gyorsbillentyű kombináció megnyomásával tehető meg az alábbi ábrának megfelelően.



29. ábra: Program fordítása

Sikeres fordítás a kimenet (**Output**) ablakban látható.



30. ábra: Kimenet ablak

A program futásának eredménye konzolban jelenik meg, amely a következő ábrán látható.

```

D:\Sort\Debug\Sort.exe
Buborekrendezés
=====
Kiindulo állapot => { 54 6 32 49 67 15 70 10 53 5 7 90 52 53 97 63 25 45 71 39 67 72 43 50 66 33 86 63 33 80 85 7 16 77
58 94 96 11 43 10 78 52 30 21 49 57 30 78 36 88 33 27 75 68 11 54 36 93 38 77 25 9 38 15 93 67 62 15 64 81 11 31 27 43 8
5 5 35 78 33 65 36 33 78 92 53 55 54 11 54 33 74 66 79 51 64 62 11 42 29 81 }
Vegallapot => { 5 5 6 7 7 9 10 10 11 11 11 11 15 15 15 16 21 25 25 27 27 29 30 30 31 32 33 33 33 33 33 35 36
36 36 38 38 39 42 43 43 43 45 49 49 50 51 52 52 53 53 53 54 54 54 54 55 57 58 62 62 63 63 64 64 65 66 66 67 67 67 68 70
71 72 74 75 77 77 78 78 78 79 80 81 81 85 85 86 88 88 90 92 93 93 94 96 97 }
Rendezesi ido: 26 mikromasodperc [us]

Beszuro rendezes:
=====
Kiindulo állapot => { 54 6 32 49 67 15 70 10 53 5 7 90 52 53 97 63 25 45 71 39 67 72 43 50 66 33 86 63 33 80 85 7 16 77
58 94 96 11 43 10 78 52 30 21 49 57 30 78 36 88 33 27 75 68 11 54 36 93 38 77 25 9 38 15 93 67 62 15 64 81 11 31 27 43 8
5 5 35 78 33 65 36 33 78 92 53 55 54 11 54 33 74 66 79 51 64 62 11 42 29 81 }
Vegallapot => { 5 5 6 7 7 9 10 10 11 11 11 11 15 15 15 16 21 25 25 27 27 29 30 30 31 32 33 33 33 33 33 35 36
36 36 38 38 39 42 43 43 43 45 49 49 50 51 52 52 53 53 53 54 54 54 54 55 57 58 62 62 63 63 64 64 65 66 66 67 67 67 68 70
71 72 74 75 77 77 78 78 78 79 80 81 81 85 85 86 88 88 90 92 93 93 94 96 97 }
Rendezesi ido: 9 mikromasodperc [us]

Kozvetlen kivlasztos rendezes
=====
Kiindulo állapot => { 54 6 32 49 67 15 70 10 53 5 7 90 52 53 97 63 25 45 71 39 67 72 43 50 66 33 86 63 33 80 85 7 16 77
58 94 96 11 43 10 78 52 30 21 49 57 30 78 36 88 33 27 75 68 11 54 36 93 38 77 25 9 38 15 93 67 62 15 64 81 11 31 27 43 8
5 5 35 78 33 65 36 33 78 92 53 55 54 11 54 33 74 66 79 51 64 62 11 42 29 81 }
Vegallapot => { 5 5 6 7 7 9 10 10 11 11 11 11 15 15 15 16 21 25 25 27 27 29 30 30 31 32 33 33 33 33 33 35 36
36 36 38 38 39 42 43 43 43 45 49 49 50 51 52 52 53 53 53 54 54 54 54 55 57 58 62 62 63 63 64 64 65 66 66 67 67 67 68 70
71 72 74 75 77 77 78 78 78 79 80 81 81 85 85 86 88 88 90 92 93 93 94 96 97 }
Rendezesi ido: 18 mikromasodperc [us]

Shell-rendezés
=====
Kiindulo állapot => { 54 6 32 49 67 15 70 10 53 5 7 90 52 53 97 63 25 45 71 39 67 72 43 50 66 33 86 63 33 80 85 7 16 77
58 94 96 11 43 10 78 52 30 21 49 57 30 78 36 88 33 27 75 68 11 54 36 93 38 77 25 9 38 15 93 67 62 15 64 81 11 31 27 43 8
5 5 35 78 33 65 36 33 78 92 53 55 54 11 54 33 74 66 79 51 64 62 11 42 29 81 }
Vegallapot => { 5 5 6 7 7 9 10 10 11 11 11 11 15 15 15 16 21 25 25 27 27 29 30 30 31 32 33 33 33 33 33 35 36
36 36 38 38 39 42 43 43 43 45 49 49 50 51 52 52 53 53 53 54 54 54 54 55 57 58 62 62 63 63 64 64 65 66 66 67 67 67 68 70
71 72 74 75 77 77 78 78 78 79 80 81 81 85 85 86 88 88 90 92 93 93 94 96 97 }
Rendezesi ido: 9 mikromasodperc [us]

```

31. ábra: A program eredménye

A feladathoz kapcsolódó magyarázatok

Az osztály deklarációját a (**Sort.h**) állományban adjuk meg. Az osztály egy magán (**private**) láthatóságú konstans adattaggal rendelkezik, amely a jelenlegi feladat esetében, eltárolja a rendezni kívánt tömbök méretét. Konstansok használata különböző tömb méretek meghatározásánál bevett szokás, mivel ha a méretérték valamilyen okból kifolyólag változtatásra kerül, elég csak az állandó értékét átírni és nem kell a különböző műveletek (tömb feltöltés, rendezés vagy épp megjelenítés) végrehajtásáért felelős egységeket, részeket nyomon követni és módosítani. C++-ban, egy osztályon belül konstans megadása az alábbi formában történhet:

```
static const int SIZE = 100;
```

A **static** kulcsszó jelöli azt, hogy ezt a konstanst megosztva használják az osztályok példányai, így az közvetlenül az osztályhoz tartozik. Fontos szabály, hogy amíg a statikus osztály adattagokat az osztályon kívül inicializáljuk, addig az állandókat az osztályon belül is elláthatjuk kezdőértékkel. A **const** kulcsszó

jelöli, hogy ebben az esetben egy állandóról van szó, majd következik a konstans típusának megadása, ezt követi a konstans neve, amelyet a kód könnyebb olvashatóság végett (jobban elkülönítve a környezettől) nagybetűvel került megfogalmazásra, végül a konstans kezdőértékkel történő ellátása következik, ebben az esetben a 100 számértékkel. Az adattag megfogalmazását követően létrehozásra kerül egy explicit, alapértelmezett, paraméter nélküli konstruktor. Az *ArrayFill(int *numbers)*, *BubbleSort(int *copiedNumbers)*, *SelectionSort(int *copiedNumbers)*, *InsertionSort(int *copiedNumbers)*, *ShellSort(int *copiedNumbers)*, *PrintArray(const int *copiedNumbers)* függvények visszatérési típusaik **void**, amely szó jelentése szerint a függvények nem térnek vissza különböző típusú értékekkel. A fentebbi függvények paraméterlistájukban is megegyeznek, egy egész típusú dinamikus (memória) helyfoglalású tömböt várnak. Az *ArrayCopy(const int *numbers, int *copiedNumbers)* tagfüggvény **inline** módon került megvalósításra, a függvény felelőssége az eredeti tömb egy másolatának létrehozása. A deklaráció végén megfogalmazásra került a(z) *~Sort()* nevezetű, üres paraméterlistával rendelkező destruktorktor, amely az erőforrások felszabadításáért lesz felelős.

A **(Sort.cpp)** állomány tartalmazza az osztály megvalósítását, tagfüggvényeinek implementálását. A tagfüggvények neve előtt a **::** hatókör operátor kerül felhasználásra azon célból, hogy egyértelműsítésre kerüljön melyik osztály tagfüggvényét szeretnénk kifejezni. Az *ArrayFill(int numbers)* függvény feladata a paraméterül kapott egész típusú tömb véletlen számokkal történő feltöltése 1 és 100 között. Az osztály implementációja során adott rendezési algoritmust végző tagfüggvények felelőssége a rájuk szabott rendezési műveletek implementálása. Ezek a tagfüggvények minden esetben paraméterül kapják a még rendezetlen tömböt, majd megjelenítik azt a konzolon, mint kiinduló állapot, majd közvetlenül a rendezési algoritmusok előtt elindítjuk az időmérést, amelyet a rendezés végeztével leállítunk. Ezt követően megjelenítjük a már rendezett tömb – a *PrintArray()* függvény segítségével – értékeket, valamint azt, hogy mennyi időbe került a rendezés. Az idő méréséhez egy **chrono** nevezetű **header** fájlt használunk, továbbá szükséges az **std** névtér behívása is, ahol az egyes elemek implementálása megtalálható. Az aktuális időpont lekérését a következő utasítással végezzük, amelyet a **start** lokális változóban tároljuk:

```
chrono::steady_clock::time_point start = chrono::steady_clock::now();
```

A rendezési algoritmusok lefutását követően ismét lekérjük az aktuális időt, amelyet az **end** lokális változó kap értékül:

```
chrono::steady_clock::time_point end = chrono::steady_clock::now();
```

A rendezéshez szükséges idő megjelenítését a következő utasítással végezzük, ahol mikromásodpercben kiszámoljuk az eltelt időtartamot (**end - start**):

```
chrono::duration_cast<chrono::microseconds>(end - start).count()
```

A **(main.cpp)** állomány tartalmazza a **main()** függvényt, a program belépési pontját. A főfüggvényben történő dinamikus példányosítás után hívjuk meg a szükséges függvényeket. A példányosítás során átadásra kerülnek azon tömb(ök), amelyeken a rendezési műveleteket elvégezzük. A függvényhívásoknál annak ellenére, hogy egy dinamikus helyfoglalású tömböt hoztunk létre, nem kellett & jelet tenni a tömbnevek elé, mivel a tömb neve a tömb címét jelenti. Gondoskodunk a lefoglalt erőforrások felszabadításáról a destruktor hívásának segítségével. Ahhoz, hogy véletlen szám generálás során ne ugyanazokat a számokat kapjuk, szükségünk van a processzor idő nullázására, amelyet a következő utasítással végzünk:

```
srand((unsigned)time(0));
```

Minta példa függvénynevek túlterhelésére Visual Studio Community fejlesztőkörnyezetben

Módosítsuk az előző minta példánkat úgy, hogy az megfeleljen a függvénynevek túlterhelés technikájának. Írjunk egy **SortByFunctionOverloading** nevezetű osztályt, ahol a rendezési algoritmusokat nem csak egész típusú adatok esetében implementáljuk, hanem lebegőpontos és karakter típusok esetén is. Az osztály adatai a következők:

- **SIZE** nevezetű statikus, konstans, egésztípusú érték, amelyet 100-al inicializálunk.

Ügyeljünk az egységbezárás elvére, tehát az adattag(ok) **private** láthatóságúak legyenek!

Valósítsuk meg az osztály metódusait:

- Írjuk meg az osztály explicit, alapértelmezett konstruktorát.
- Írjunk egy `ArrayFill(int *integerNumbers)` függvényt, amelynek felelőssége a 100 elemű, **egész típusú tömb**, véletlen számokkal (1 és 100 között) történő feltöltése.
- Írjunk egy `ArrayFill(double *doubleNumbers)` függvényt, amelynek felelőssége a 100 elemű, **lebegőpontos típusú tömb**, véletlen számokkal (1,0 és 100,0 között) történő feltöltése.
- Írjunk egy `ArrayFill(char *characters)` függvényt, amelynek felelőssége a 100 elemű, **karakter típusú tömb**, véletlen betűkkel (a és z között) történő feltöltése.
- Írjunk egy `ArrayCopy(const int *integerNumbers, int *integerArray)` függvényt, amely felelőssége az eredeti tömb átmásolása egy átmeneti tömbbe. Ennek célja, hogy a rendezési algoritmusok hatékonyságát ne torzítsa, mivel ha mindig újra inicializálnánk a rendezendő tömböt, akkor

folyamatosan eltérő értékeket tartalmazna, ezáltal megnövelve vagy csökkentve a rendezéshez szükséges időtartamot, így szükséges minden egyes rendezési algoritmusnál ugyanazokkal a tömb értékekkel dolgozni.

- Írjunk egy `ArrayCopy(const double *doubleNumbers, double *doubleArray)` függvényt, amely felelőssége a fentebb leírt tömbmásolás, azonban ebben az esetben **lebegőpontos** típusú számokkal.
- Írjunk egy `ArrayCopy(const char *characters, int *charArray)` függvényt, amely felelőssége a fentebb is részletezett tömbmásolás, azonban ebben az esetben **karakter típusú** értékekkel.
- Írjunk egy `BubbleSort(int *integerArray)` függvényt, amely a buborékos rendezés algoritmusát valósítja meg **egész szám típusokon**.
- Írjunk egy `BubbleSort(double *doubleArray)` függvényt, amely a buborékos rendezés algoritmusát valósítja meg **lebegőpontos szám típusokon**.
- Írjunk egy `BubbleSort(char *charArray)` függvényt, amely a buborékos rendezés algoritmusát valósítja meg **karakter típusú** tömbön.
- Írjunk egy `InsertionSort(int *integerArray)` függvényt, amely a beszűrő rendezés algoritmusát valósítja meg **egész szám típusokon**.
- Írjunk egy `InsertionSort(double *doubleArray)` függvényt, amely a beszűrő rendezés algoritmusát valósítja meg **lebegőpontos szám típusokon**.
- Írjunk egy `InsertionSort(char *charArray)` függvényt, amely a beszűrő rendezés algoritmusát valósítja meg **karakter típusú** tömbön.
- Írjunk egy `SelectionSort(int *integerArray)` függvényt, amely a közvetlen kiválasztásos rendezés algoritmusát valósítja meg **egész szám típusokon**.
- Írjunk egy `SelectionSort(double *doubleArray)` függvényt, amely a közvetlen kiválasztásos rendezés algoritmusát valósítja meg **lebegőpontos szám típusokon**.
- Írjunk egy `SelectionSort(char *charArray)` függvényt, amely a közvetlen kiválasztásos rendezés algoritmusát valósítja meg **karakter típusú** tömbön.
- Írjunk egy `ShellSort(int *integerArray)` függvényt, amely a Shell-rendezés algoritmusát valósítja meg **egész szám típusokon**.
- Írjunk egy `ShellSort(double *doubleArray)` függvényt, amely a Shell-rendezés algoritmusát valósítja meg **lebegőpontos szám típusokon**.
- Írjunk egy `ShellSort(char *charArray)` függvényt, amely a Shell-rendezés algoritmusát valósítja meg **karakter típusú** tömbön.
- Írjunk egy `PrintArray(const int *integerArray)` függvényt, amelynek felelőssége a paraméterül kapott **egész típusú** tömb értékeinek megjelenítése a képernyőn, adott formátumban.
- Írjunk egy `PrintArray(const double *doubleArray)` függvényt, amelynek felelőssége a paraméterül kapott **lebegőpontos típusú** tömb értékeinek megjelenítése a képernyőn, adott formátumban.

- Írjunk egy *PrintArray(const char *charArray)* függvényt, amelynek felelőssége a paraméterül kapott **karaktertípusú** tömb értékeinek megjelenítése a képernyőn, adott formátumban.
- Ne feledkezzünk meg az objektumpéldány által lefoglalt erőforrás felszabadításáról, a destruktork explicit módon történő megírásáról sem.

Az osztály implementálását követően dinamikus módszerrel (a memória dinamikus területén [**heap**]) készítsünk egy objektumpéldányt. Az objektumpéldány segítségével a feladat leírásnak megfelelően hívjuk meg az osztály egyes tagfüggvényeit. A megvalósítás során, ha tudjuk, hogy a függvénynek átadott paraméter a függvény futása során nem változik, használjunk **const** jelzőt. Amennyiben nincs tovább szükség az objektumpéldányra, gondoskodjunk annak megszüntetéséről, elkerülve ezzel a memóriaszivárgást. Mielőtt a feladatnak neki látnánk, tervezzük meg a statikus modell(eket).

Statikus tervezés

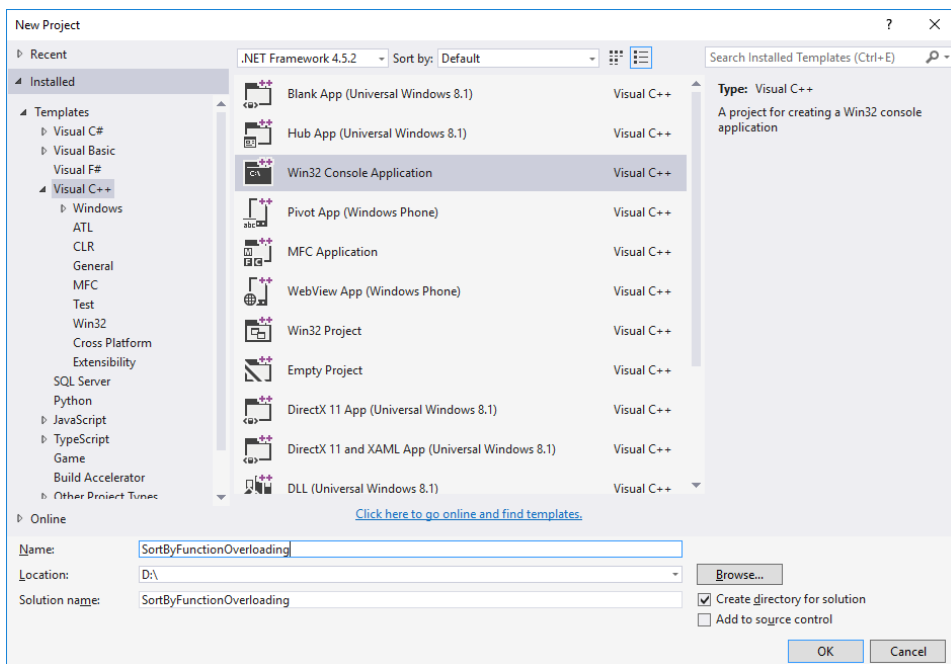
SortByFunctionOverloading	
-	<u>SIZE: int = 100 {readOnly}</u>
+	ArrayCopy(int*, int*): void
+	ArrayCopy(double*, double*): void
+	ArrayCopy(char*, char*): void
+	ArrayFill(int*): void
+	ArrayFill(double*): void
+	ArrayFill(char*): void
+	BubbleSort(int*): void
+	BubbleSort(double*): void
+	BubbleSort(char*): void
+	InsertionSort(int*): void
+	InsertionSort(double*): void
+	InsertionSort(char*): void
+	PrintArray(int*): void
+	PrintArray(double*): void
+	PrintArray(char*): void
+	SelectionSort(int*): void
+	SelectionSort(double*): void
+	SelectionSort(char*): void
+	ShellSort(int*): void
+	ShellSort(double*): void
+	ShellSort(char*): void
+	SortByFunctionOverloading()
+	~SortByFunctionOverloading()

32. ábra: Osztály diagram

A projekt létrehozása

A projekt megvalósítása során konzolalkalmazást készítünk, ahol a Visual Studio, szabványos C++-nak megfelelő szintaktikát alkalmaz. A projekt három fájlban kerül implementálásra. Az alkalmazás belépési pontja a **main** függvény, amely a három fájl közül az egyikben fog helyet foglalni. Az előzőekben megfogalmazott minta példa kerül implementálásra és összeállításra, továbbá Windows operációs rendszeren futtatható kóddá történő lefordításra **Visual Studio Community** fejlesztőkörnyezet felhasználásával. A lefordított 32 bites alkalmazás konzolban kerül futtatásra.

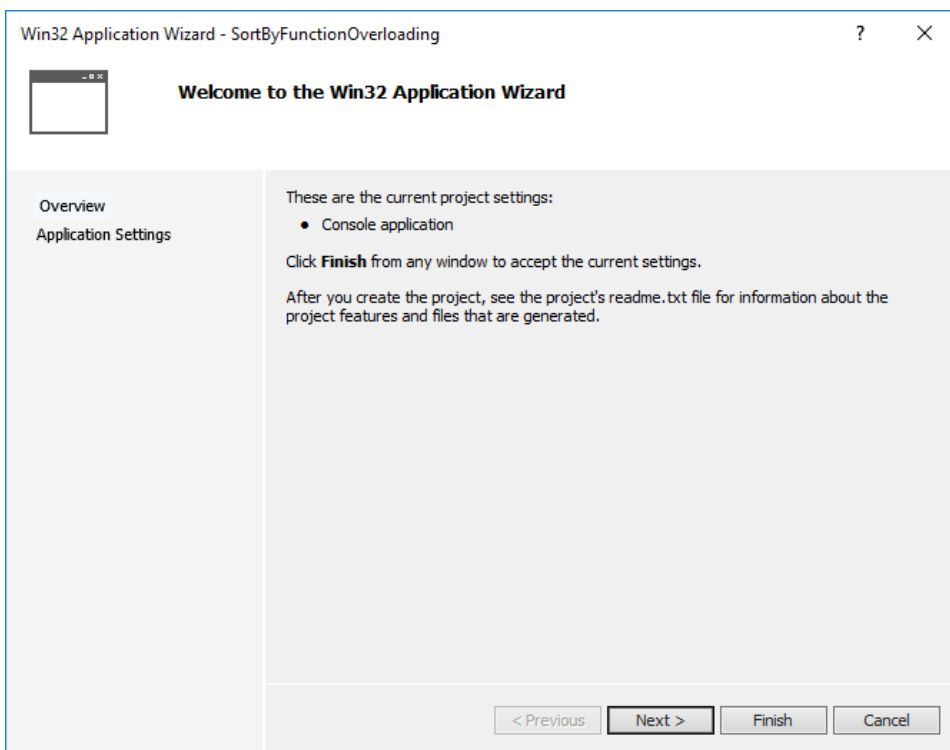
Új projekt a **File** menü **New** menüpontjában hozható létre, a **Project** almenüt kiválasztva, vagy akár a **Ctrl+Shift+N** gyorsindító gombkombinációval. Új projekt kiválasztása után megjelenő ablakban a **Visual C++ Win32** sablont (**Template**) kell kiválasztani és a **Win32 Console Application**-t, mely az alábbi ábrán látható.



33. ábra: Win32 Console Application projekt létrehozása

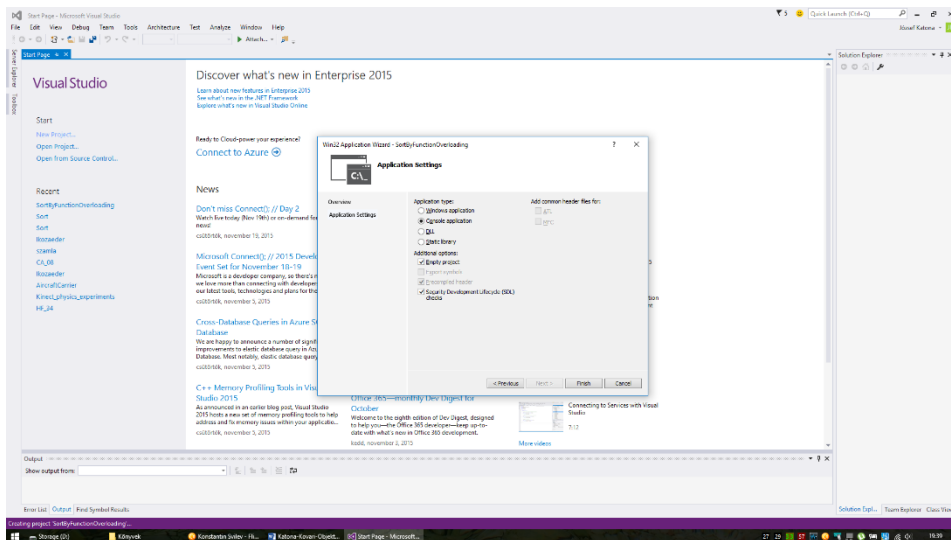
Az ablak alsó részén a projekt nevének (**Name**) a példának megfelelően az **SortByFunctionOverloading** név került megadásra. Ebben a részben lehet módosítani a Projekthez kapcsolódó fájlok mentési helyét is (**Location**). Egy összetett program általában több projektből tevődik össze, ezeket egységes

rendszerre a megoldás (**Solution**) fogja össze. Az OK gombra kattintást követően elindul az alkalmazás varázsló, mely a következő ábrán látható.



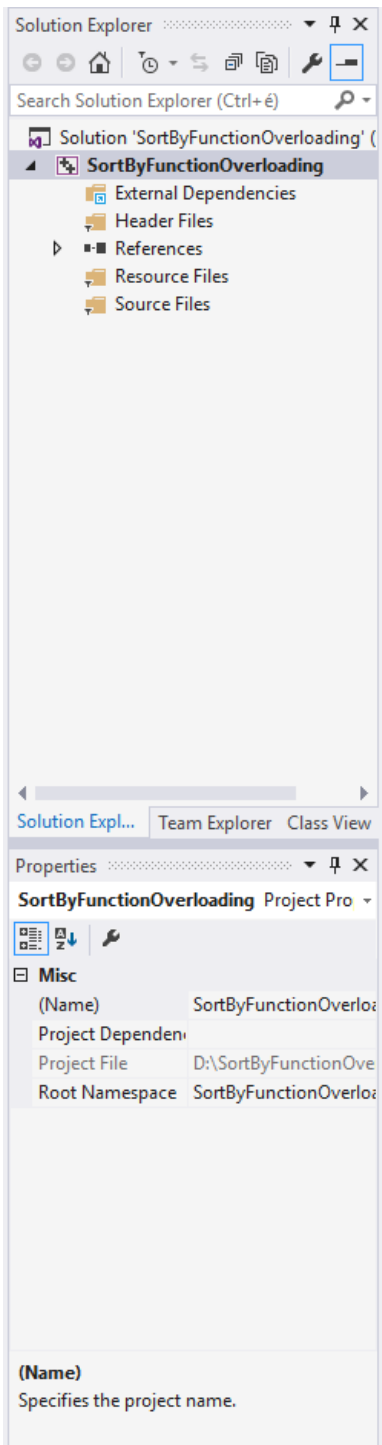
34. ábra: Win32 konzol alkalmazás varázsló

A **Next** gombra kattintva a beállítások ablak jelenik meg, itt megadhatjuk az alkalmazás típusát (**Console Application**), és néhány kiegészítő opciót. Mivel egyszerű alkalmazást készítünk, ezért célszerű az üres projekt (**Empty project**) opció kiválasztása, ahogy az alábbi képen látszódik.



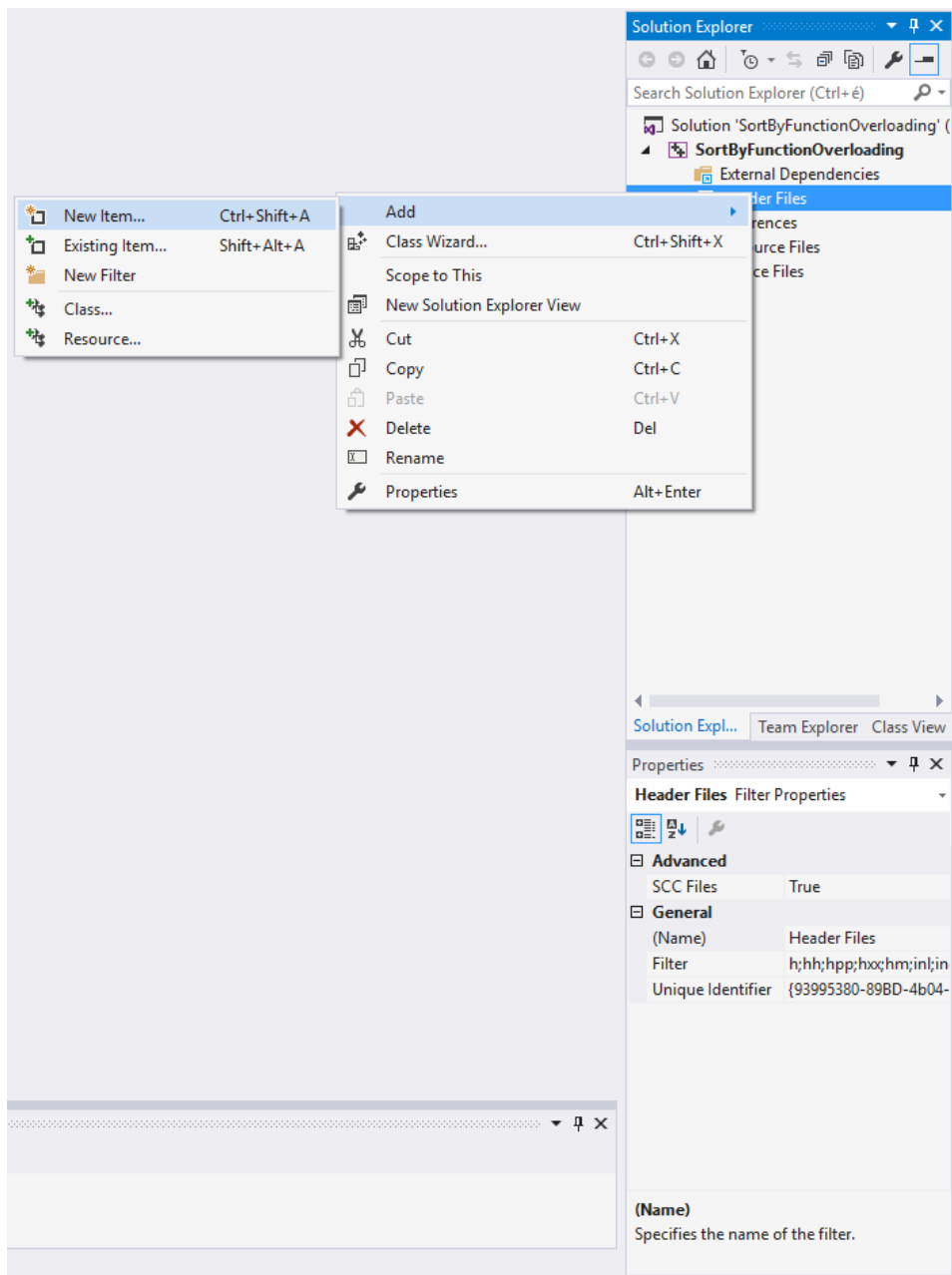
35. ábra: Alkalmazás beállításai

A **Finish** megnyomását követően a képernyő jobb szélén, a megoldás ablaka (**Solution Explorer**), alatta a beállítások ablak jelenik meg, ahogy a következő ábrán látható.



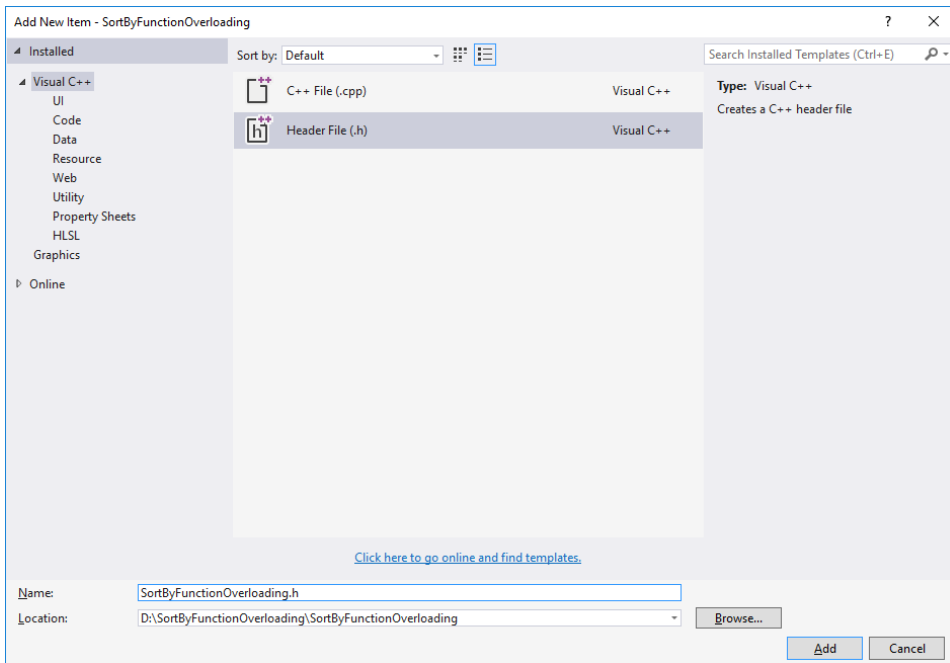
36. ábra: Megoldás és beállítások ablak

A program kódjának forrás és **header** fájljait (**Source Files, Header Files**) a megoldás ablakban a kiválasztott fájl típusra az egér jobb gombjával kattintva adható hozzá az alábbi ábrának megfelelően vagy a **Shift+A** billentyűkombináció megnyomásával.



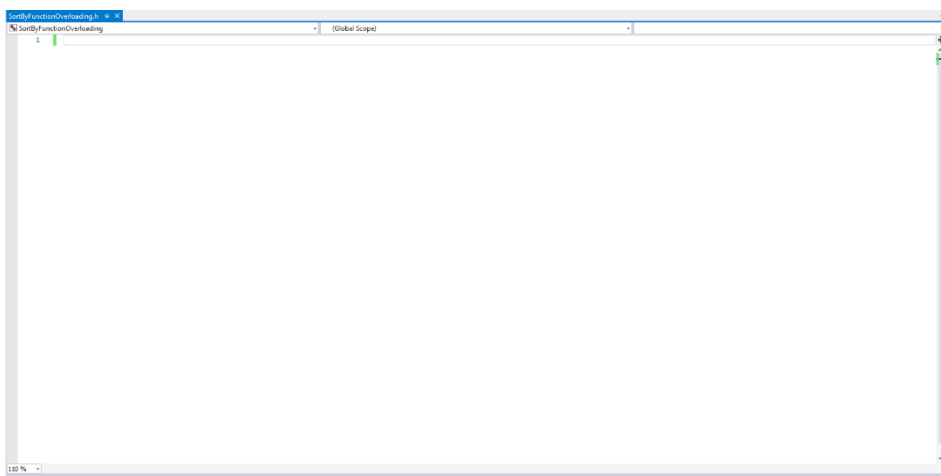
37. ábra: Új header fájl hozzáadása a projekthez

Új header fájl (Header File) választásával és a név (Name) az **SortByFunctionOverloading.h** megadásával hozható létre a **SortByFunctionOverloading.h** fájl, az alábbi ábrának megfelelően.



38. ábra: Új header fájl hozzáadása a projekthez, név megadása

Az új fájl hozzáadásával a képernyő középső részén megjelenik a forráskód szerkesztő az alábbi ábrának megfelelően.



39. ábra: *Forráskód szerkesztő*

SortByFunctionOverloading.h

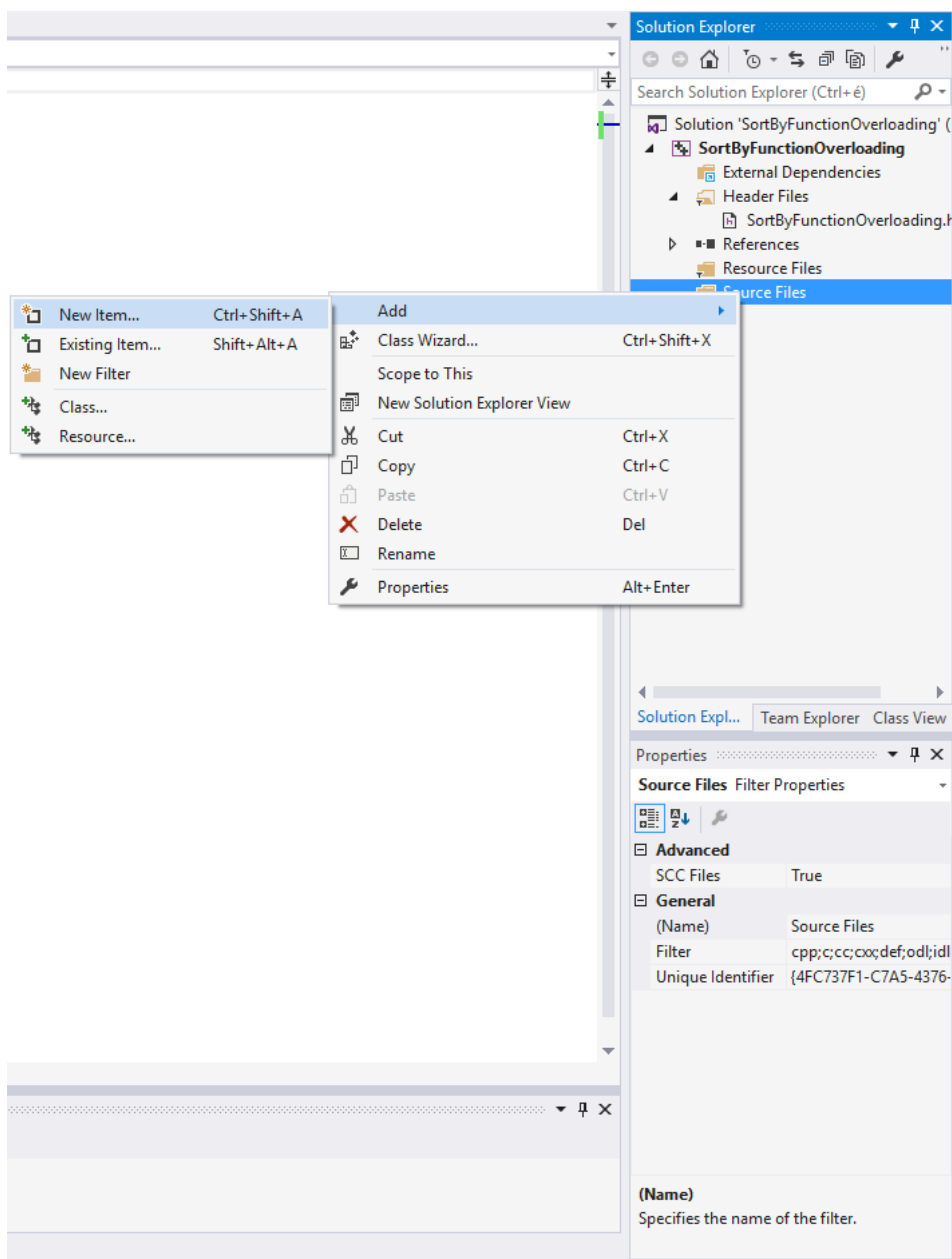
```
#ifndef SORTBYFUNCTIONOVERLOADING_H
#define SORTBYFUNCTIONOVERLOADING_H

class SortByFunctionOverloading
{
private:
    static const int SIZE = 100;

public:
    SortByFunctionOverloading() { }
    void ArrayFill(int *integerNumbers);
    void ArrayFill(double *doubleNumbers);
    void ArrayFill(char *characters);
    void ArrayCopy(const int *integerNumbers, int *integerArray) { for (int i = 0; i < SIZE; ++i) integerArray[i] = integerNumbers[i]; }
    void ArrayCopy(const double *doubleNumbers, double *doubleArray) { for (int i = 0; i < SIZE; ++i) doubleArray[i] = doubleNumbers[i]; }
    void ArrayCopy(const char *characters, char *charArray) { for (int i = 0; i < SIZE; ++i) charArray[i] = characters[i]; }
    void BubbleSort(int *integerArray);
    void BubbleSort(double *doubleArray);
    void BubbleSort(char *charArray);
    void InsertionSort(int *integerArray);
    void InsertionSort(double *doubleArray);
    void InsertionSort(char *charArray);
    void SelectionSort(int *integerArray);
    void SelectionSort(double *doubleArray);
    void SelectionSort(char *charArray);
    void ShellSort(int *integerArray);
    void ShellSort(double *doubleArray);
    void ShellSort(char *charArray);
    void PrintArray(const int *integerArray);
    void PrintArray(const double *doubleArray);
    void PrintArray(const char *charArray);
    ~SortByFunctionOverloading() { }
};

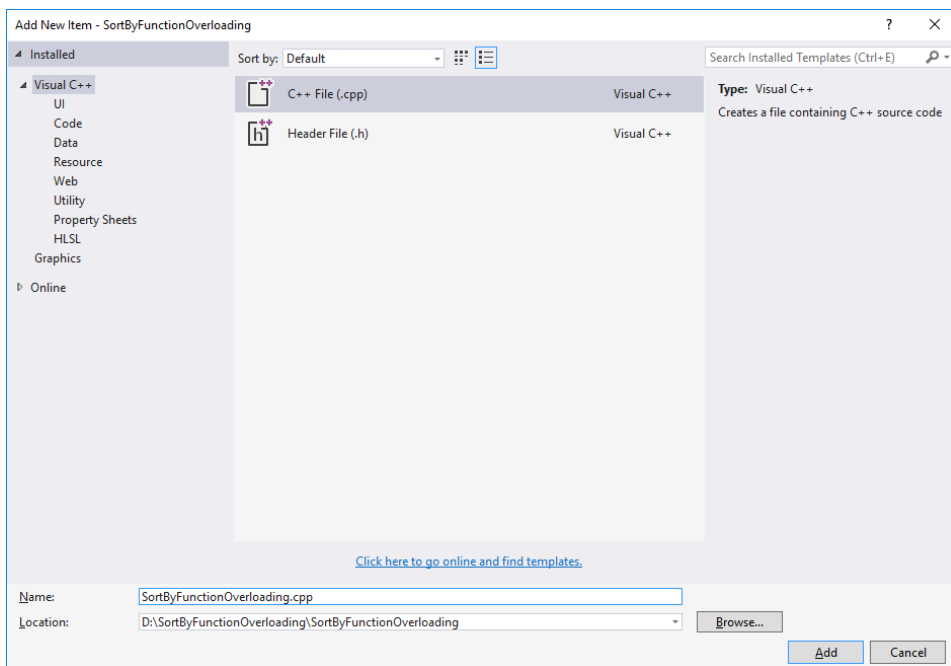
#endif
```

A deklarált osztályok megvalósításait a **SortByFunctionOverloading.cpp** forrásfájlok tartalmazzák, melyeket a **header** fájlokhoz hasonlóan, de a forrásfájl (**Source File**) hozzáadásával hozhatunk létre.



40. ábra: Új forrásfájl hozzáadása a projekthez

Új C++ fájl (C++ **File**) választásával és a név (**Name**) a **SortByFunctionOverloading.cpp** megadásával hozható létre.



41. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

SortByFunctionOverloading.cpp

```
#include <iostream>
#include <ctime>
#include <chrono>
#include <iomanip>
#include "SortByFunctionOverloading.h"

using namespace std;

void SortByFunctionOverloading::ArrayFill(int *integerNumbers)
{
    srand((unsigned)time(0));
    for (int i = 0; i < SIZE; ++i)
        integerNumbers[i] = (rand() % 100) + 1;
}

void SortByFunctionOverloading::ArrayFill(double *doubleNumbers)
{
    srand((unsigned)time(0));
    for (int i = 0; i < SIZE; ++i)
        doubleNumbers[i] = 1.0 + (double)rand() / (double)(RAND_MAX/(100.0-1.0));
}

void SortByFunctionOverloading::ArrayFill(char *characters)
{
    srand((unsigned)time(0));
    for (int i = 0; i < SIZE; ++i)
        characters[i] = (char)(rand() % 26) + 97;
}

void SortByFunctionOverloading::BubbleSort(int *integerArray)
{
    cout << "Buborekrendezes" << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(integerArray);

    int temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int i = 0; i < SIZE; i++)
    {
        for (int j = SIZE - 1; j > i; j--)
        {
            if (integerArray[j] < integerArray[j - 1])
            {
                temp = integerArray[j - 1];
                integerArray[j - 1] = integerArray[j];
                integerArray[j] = temp;
            }
        }
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(integerArray);
    cout << "Rendezesi ido: " <<
        chrono::duration_cast<chrono::microseconds>(end - start).count() << "
        mikromasodperc [us]";
    cout << endl; cout << endl;
}
```

```

void SortByFunctionOverloading::BubbleSort(double *doubleArray)
{
    cout << "Buborekrendezes" << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(doubleArray);

    double temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int i = 0; i < SIZE; i++)
    {
        for (int j = SIZE - 1; j > i; j--)
        {
            if (doubleArray[j] < doubleArray[j - 1])
            {
                temp = doubleArray[j - 1];
                doubleArray[j - 1] = doubleArray[j];
                doubleArray[j] = temp;
            }
        }
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => " << endl; PrintArray(doubleArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::BubbleSort(char *charArray)
{
    cout << "Buborekrendezes" << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(charArray);

    char temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int i = 0; i < SIZE; i++)
    {
        for (int j = SIZE - 1; j > i; j--)
        {
            if (charArray[j] < charArray[j - 1])
            {
                temp = charArray[j - 1];
                charArray[j - 1] = charArray[j];
                charArray[j] = temp;
            }
        }
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => " << endl; PrintArray(charArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

```

```

void SortByFunctionOverloading::InsertionSort(int *integerArray)
{
    cout << "Beszuro rendezes: " << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(integerArray);

    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int j = 1; j < SIZE; j++)
    {
        int key = integerArray[j];
        int i = j - 1;
        while (i > -1 && integerArray[i] > key)
        {
            integerArray[i + 1] = integerArray[i];
            i = i - 1;
        }
        integerArray[i + 1] = key;
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(integerArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::InsertionSort(double *doubleArray)
{
    cout << "Beszuro rendezes: " << endl;
    cout << "=====" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(doubleArray);

    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int j = 1; j < SIZE; j++)
    {
        double key = doubleArray[j];
        int i = j - 1;
        while (i > -1 && doubleArray[i] > key)
        {
            doubleArray[i + 1] = doubleArray[i];
            i = i - 1;
        }
        doubleArray[i + 1] = key;
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(doubleArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::InsertionSort(char *charArray)
{
    cout << "Beszuro rendezes: " << endl;
    cout << "=====" << endl;

```

```

cout << "Kiindulo allapot =>"; PrintArray(charArray);
chrono::steady_clock::time_point start = chrono::steady_clock::now();

for (int j = 1; j < SIZE; j++)
{
    char key = charArray[j];
    int i = j - 1;
    while (i > -1 && charArray[i] > key)
    {
        charArray[i + 1] = charArray[i];
        i = i - 1;
    }
    charArray[i + 1] = key;
}

chrono::steady_clock::time_point end = chrono::steady_clock::now();
cout << "Vegallapot => "; PrintArray(charArray);
cout << "Rendezesi ido: " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << "
mikromasodperc [us]";
cout << endl; cout << endl;
}

void SortByFunctionOverloading::SelectionSort(int *integerArray)
{
    cout << "Kozvetlen kivalaszto rendezes" << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(integerArray);

    int position, temp;

    chrono::steady_clock::time_point start = chrono::steady_clock::now();
    for (int i = 0; i < SIZE - 1; i++)
    {
        position = i;
        for (int j = i + 1; j < SIZE; j++)
        {
            if (integerArray[j] < integerArray[position])
                position = j;
        }

        if (position != i)
        {
            temp = integerArray[i];
            integerArray[i] = integerArray[position];
            integerArray[position] = temp;
        }
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(integerArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end -
start).count() << " mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::SelectionSort(double *doubleArray)
{
    cout << "Kozvetlen kivalaszto rendezes" << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(doubleArray);

```

```

int position;
double temp;

chrono::steady_clock::time_point start = chrono::steady_clock::now();
for (int i = 0; i < SIZE - 1; i++)
{
    position = i;
    for (int j = i + 1; j < SIZE; j++)
    {
        if (doubleArray[j] < doubleArray[position])
            position = j;
    }

    if (position != i)
    {
        temp = doubleArray[i];
        doubleArray[i] = doubleArray[position];
        doubleArray[position] = temp;
    }
}
chrono::steady_clock::time_point end = chrono::steady_clock::now();
cout << "Vegallapot => "; PrintArray(doubleArray);
cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
cout << endl; cout << endl;
}

void SortByFunctionOverloading::SelectionSort(char *charArray)
{
    cout << "Kozvetlen kivalaszto rendezes" << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(charArray);

    int position;
    char temp;

    chrono::steady_clock::time_point start = chrono::steady_clock::now();
    for (int i = 0; i < SIZE - 1; i++)
    {
        position = i;
        for (int j = i + 1; j < SIZE; j++)
        {
            if (charArray[j] < charArray[position])
                position = j;
        }

        if (position != i)
        {
            temp = charArray[i];
            charArray[i] = charArray[position];
            charArray[position] = temp;
        }
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(charArray);
    cout << "Rendezesi ido: " <<
        chrono::duration_cast<chrono::microseconds>(end - start).count() << "
        mikromasodperc [us]";
    cout << endl; cout << endl;
}

```

```

void SortByFunctionOverloading::ShellSort(int *integerArray)
{
    cout << "Shell-rendezes" << endl;
    cout << "======" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(integerArray);

    int temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int gap = SIZE / 2; gap > 0; gap /= 2)
        for (int i = gap; i < SIZE; i++)
            for (int j = i - gap; j >= 0 && integerArray[j] >
                integerArray[j + gap]; j -= gap) {
                temp = integerArray[j];
                integerArray[j] = integerArray[j + gap];
                integerArray[j + gap] = temp;
            }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(integerArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::ShellSort(double *doubleArray)
{
    cout << "Shell-rendezes" << endl;
    cout << "======" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(doubleArray);

    double temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int gap = SIZE / 2; gap > 0; gap /= 2)
        for (int i = gap; i < SIZE; i++)
            for (int j = i - gap; j >= 0 && doubleArray[j] >
                doubleArray[j + gap]; j -= gap) {
                temp = doubleArray[j];
                doubleArray[j] = doubleArray[j + gap];
                doubleArray[j + gap] = temp;
            }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(doubleArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::ShellSort(char *charArray)
{
    cout << "Shell-rendezes" << endl;
    cout << "======" << endl;
    cout << "Kiindulo allapot =>"; PrintArray(charArray);

    char temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int gap = SIZE / 2; gap > 0; gap /= 2)

```

```

    for (int i = gap; i < SIZE; i++)
        for (int j = i - gap; j >= 0 && charArray[j]>charArray[j + gap];
            j -= gap) {
            temp = charArray[j];
            charArray[j] = charArray[j + gap];
            charArray[j + gap] = temp;
        }

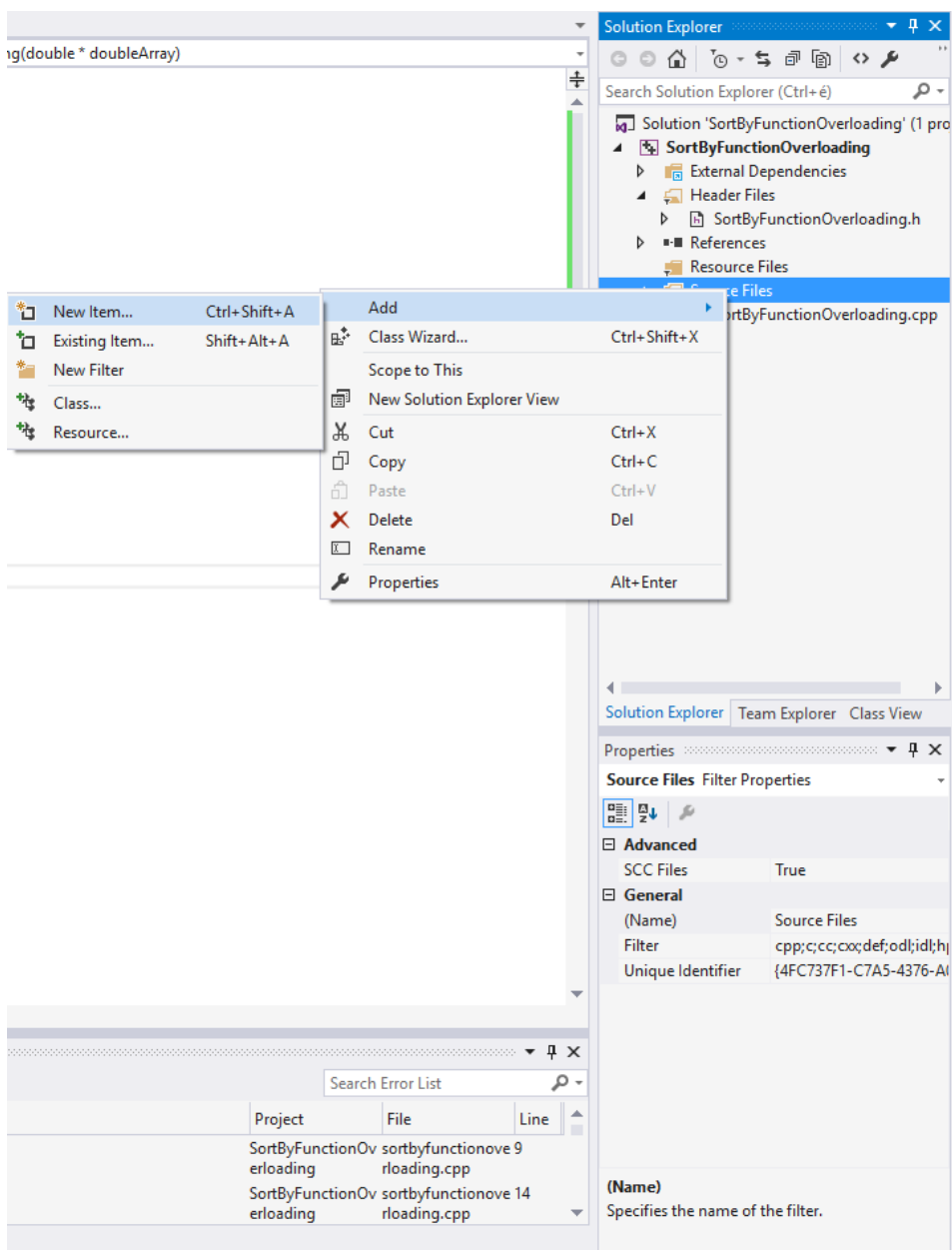
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(charArray);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

void SortByFunctionOverloading::PrintArray(const int *integerArray)
{
    cout << " { ";
    for (int i = 0; i < SIZE; ++i)
        cout << integerArray[i] << " ";
    cout << " } " << endl;
}

void SortByFunctionOverloading::PrintArray(const double *doubleArray)
{
    cout << " { ";
    for (int i = 0; i < SIZE; ++i)
        cout << setprecision(3) << doubleArray[i] << " ";
    cout << " } " << endl;
}

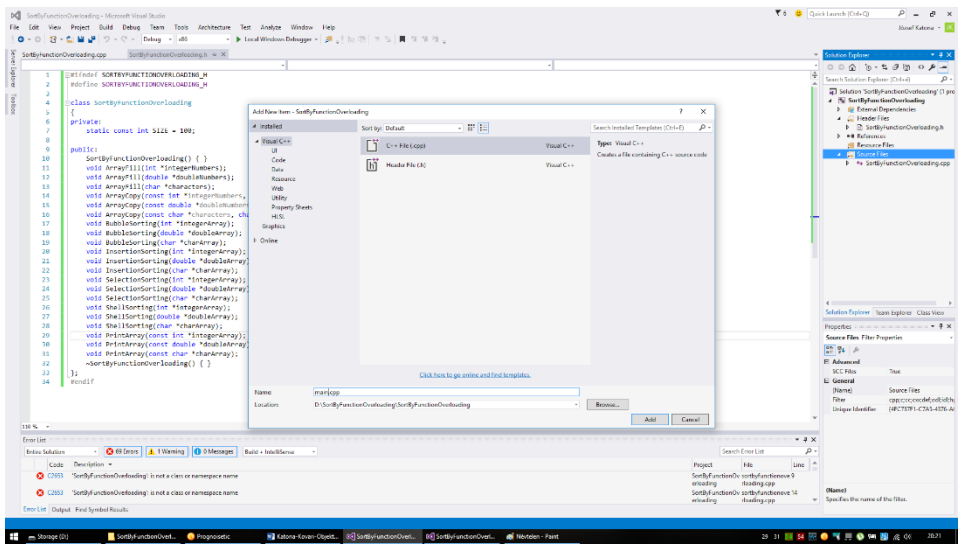
void SortByFunctionOverloading::PrintArray(const char *charArray)
{
    cout << " { ";
    for (int i = 0; i < SIZE; ++i)
        cout << charArray[i] << " ";
    cout << " } " << endl;
}

```



42. ábra: Új forrásfájl hozzáadása a projekthez

C++ fájl (C++ File) választásával és a név (Name) a **main.cpp** megadásával hozható létre.



43. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

main.cpp

```
#include <iostream>
#include <conio.h>
#include <ctime>
#include "SortByFunctionOverloading.h"

using namespace std;

int main()
{
    const int SIZE(100);
    int *integerNumbers = new int[SIZE];
    int *copiedIntegerNumbers = new int[SIZE];
    double *doubleNumbers = new double[SIZE];
    double *copiedDoubleNumbers = new double[SIZE];
    char *characters = new char[SIZE];
    char *copiedCharacters = new char[SIZE];
    SortByFunctionOverloading *objectOfSortByFunctionOverloading = new
    SortByFunctionOverloading();

    objectOfSortByFunctionOverloading->ArrayFill(integerNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(integerNumbers,
    copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->BubbleSort(copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(integerNumbers,
    copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->InsertionSort(copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(integerNumbers,
    copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->SelectionSort(copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(integerNumbers,
    copiedIntegerNumbers);
    objectOfSortByFunctionOverloading->ShellSort(copiedIntegerNumbers);

    objectOfSortByFunctionOverloading->ArrayFill(doubleNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(doubleNumbers,
    copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->BubbleSort(copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(doubleNumbers,
    copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->InsertionSort(copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(doubleNumbers,
    copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->SelectionSort(copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->ArrayCopy(doubleNumbers,
    copiedDoubleNumbers);
    objectOfSortByFunctionOverloading->ShellSort(copiedDoubleNumbers);
```

```

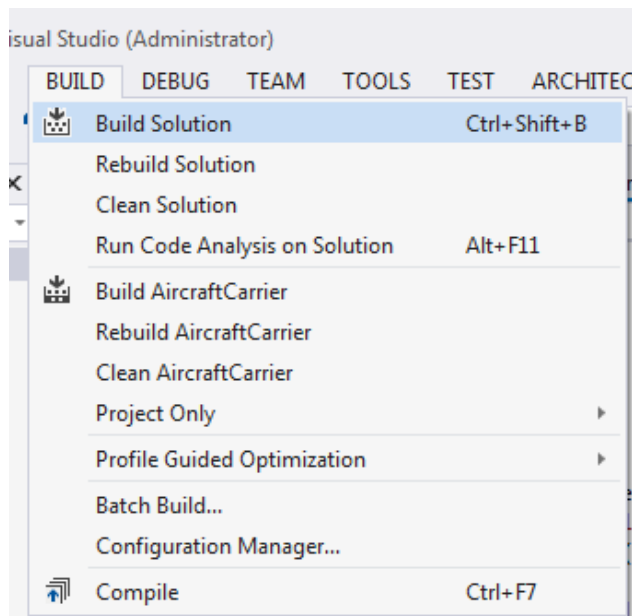
objectOfSortByFunctionOverloading->ArrayFill(characters);
objectOfSortByFunctionOverloading->ArrayCopy(characters,
copiedCharacters);
objectOfSortByFunctionOverloading->BubbleSort(copiedCharacters);
objectOfSortByFunctionOverloading->ArrayCopy(characters,
copiedCharacters);
objectOfSortByFunctionOverloading->InsertionSort(copiedCharacters);
objectOfSortByFunctionOverloading->ArrayCopy(characters,
copiedCharacters);
objectOfSortByFunctionOverloading->SelectionSort(copiedCharacters);
objectOfSortByFunctionOverloading->ArrayCopy(characters,
copiedCharacters);
objectOfSortByFunctionOverloading->ShellSort(copiedCharacters);

delete objectOfSortByFunctionOverloading;
delete[] characters;
delete[] copiedCharacters;
delete[] doubleNumbers;
delete[] copiedDoubleNumbers;
delete[] integerNumbers;
delete[] copiedIntegerNumbers;

_getch();
return 0;
}

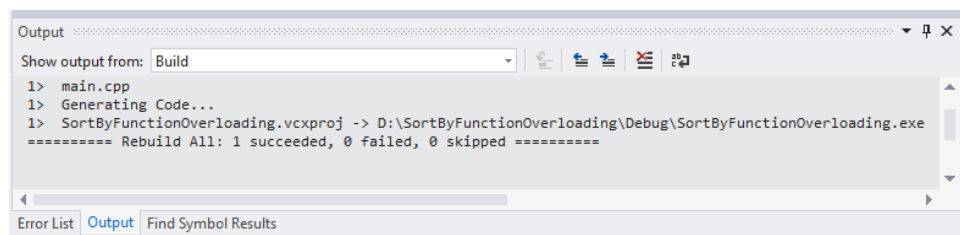
```

A forrásfájlok hozzáadása után a program forráskódja lefordítható, futtatható kód előállítását a **BUILD** menüpont **Build Solution** almenüpontjának kiválasztásával, vagy a **Ctrl+Shift+B** gyorsbillentyű kombináció megnyomásával tehető meg az alábbi ábrának megfelelően.



44. ábra: Program fordítása

Sikeres fordítás a kimenet (**Output**) ablakban látható.



45. ábra: Kimenet ablak

A program futásának eredménye konzolban jelenik meg, amely a következő ábrán látható.

```

Buborekrendezés
=====
Kiindulo allapot => { 64 20 75 55 54 33 30 25 97 50 50 49 74 47 8 64 81 81 71 41 72 38 39 64 78 52 18 84 53 22 5 62 71 87 98 82 7 5 93 6 48 37 72
69 26 10 67 81 41 71 15 24 54 97 29 43 6 95 10 85 34 60 87 70 85 10 22 15 16 4 39 57 39 59 24 76 26 65 33 59 11 2 57 25 3 74 73 79 26 53 1 56 5
40 75 40 43 23 21 }
Vegallapot => { 1 2 3 4 5 5 5 6 6 7 8 10 10 10 11 15 15 16 18 20 21 22 22 23 24 24 25 25 26 26 26 29 30 33 33 34 37 38 39 39 39 40 40 41 41
43 43 47 48 49 50 50 52 53 53 54 54 55 56 57 57 59 59 60 62 64 64 64 65 67 69 70 71 71 71 72 72 73 73 74 74 75 75 76 78 79 81 81 81 82 84 85 85 87
87 93 95 97 97 98 }
Rendezesi ido: 23 mikromasodperc [us]

Beszuro rendezes:
=====
Kiindulo allapot => { 64 20 75 55 54 33 30 25 97 50 50 49 74 47 8 64 81 81 71 41 72 38 39 64 78 52 18 84 53 22 5 62 71 87 98 82 7 5 93 6 48 37 72
69 26 10 67 81 41 71 15 24 54 97 29 43 6 95 10 85 34 60 87 70 85 10 22 15 16 4 39 57 39 59 24 76 26 65 33 59 11 2 57 25 3 74 73 79 26 53 1 56 5
40 75 40 43 23 21 }
Vegallapot => { 1 2 3 4 5 5 5 6 6 7 8 10 10 10 11 15 15 16 18 20 21 22 22 23 24 24 25 25 26 26 26 29 30 33 33 34 37 38 39 39 39 40 40 41 41
43 43 47 48 49 50 50 52 53 53 54 54 55 56 57 57 59 59 60 62 64 64 64 65 67 69 70 71 71 71 72 72 73 73 74 74 75 75 76 78 79 81 81 81 82 84 85 85 87
87 93 95 97 97 98 }
Rendezesi ido: 9 mikromasodperc [us]

Kozvetlen kivalasztó rendezes
=====
Kiindulo allapot => { 64 20 75 55 54 33 30 25 97 50 50 49 74 47 8 64 81 81 71 41 72 38 39 64 78 52 18 84 53 22 5 62 71 87 98 82 7 5 93 6 48 37 72
69 26 10 67 81 41 71 15 24 54 97 29 43 6 95 10 85 34 60 87 70 85 10 22 15 16 4 39 57 39 59 24 76 26 65 33 59 11 2 57 25 3 74 73 79 26 53 1 56 5
40 75 40 43 23 21 }
Vegallapot => { 1 2 3 4 5 5 5 6 6 7 8 10 10 10 11 15 15 16 18 20 21 22 22 23 24 24 25 25 26 26 26 29 30 33 33 34 37 38 39 39 39 40 40 41 41
43 43 47 48 49 50 50 52 53 53 54 54 55 56 57 57 59 59 60 62 64 64 64 65 67 69 70 71 71 71 72 72 73 73 74 74 75 75 76 78 79 81 81 81 82 84 85 85 87
87 93 95 97 97 98 }
Rendezesi ido: 16 mikromasodperc [us]

Shell-rendezés
=====
Kiindulo allapot => { 64 20 75 55 54 33 30 25 97 50 50 49 74 47 8 64 81 81 71 41 72 38 39 64 78 52 18 84 53 22 5 62 71 87 98 82 7 5 93 6 48 37 72
69 26 10 67 81 41 71 15 24 54 97 29 43 6 95 10 85 34 60 87 70 85 10 22 15 16 4 39 57 39 59 24 76 26 65 33 59 11 2 57 25 3 74 73 79 26 53 1 56 5
40 75 40 43 23 21 }
Vegallapot => { 1 2 3 4 5 5 5 6 6 7 8 10 10 10 11 15 15 16 18 20 21 22 22 23 24 24 25 25 26 26 26 29 30 33 33 34 37 38 39 39 39 40 40 41 41
43 43 47 48 49 50 50 52 53 53 54 54 55 56 57 57 59 59 60 62 64 64 64 65 67 69 70 71 71 71 72 72 73 73 74 74 75 75 76 78 79 81 81 81 82 84 85 85 87
87 93 95 97 97 98 }
Rendezesi ido: 6 mikromasodperc [us]

Buborekrendezés
=====
Kiindulo allapot => { 21 57 67.3 4.11 65 4.5 66.8 39.9 85.7 54.6 72.4 59.5 89.7 31.4 39.3 82 18.8 63 70.6 70.7 8.08 16.8 92.2 29 60.2 93.1 16.1 17
.9 55.1 16.9 48.8 36.3 72.8 61.7 58 51 67.4 58.8 31.4 19.3 59.2 22.7 45.8 93 80.1 30.4 96.5 53.5 48.3 14.3 86.7 48.6 96.2 78.7 30.7 95.6 42.2 17.6
75.6 26.4 14.6 30.2 1.08 12.7 26.8 26.3 79 77.5 18.4 11.1 79.2 20.9 42.4 30.8 88.2 45.8 32.8 90.9 24.6 46.6 21 13 32 25 38.8 36.6 21.7 90.6 80 54
.3 49.1 35.5 51.4 69.3 92.6 10 86.5 58.5 30.8 10.9 }
Vegallapot => { 1.08 4.11 4.5 8.08 10 10.9 11.1 12.7 13 14.3 14.6 16.1 16.8 16.9 17.6 17.9 18.4 18.8 19.3 20.9 21 21 21.7 22.7 24.6 25 26.3
26.4 26.8 29 30.2 30.4 30.7 30.8 30.8 31.4 31.4 32 32.8 35.5 36.3 36.6 38.8 39.3 39.9 42.2 42.4 45.8 45.8 46.6 48.3 48.6 48.8 49.1 51 51.4 53.5 54
.3 54.6 55.1 57 58 58.5 58.8 59.2 59.5 60.2 61.7 63 65 66.8 67.3 67.4 69.3 70.6 70.7 72.4 72.8 75.6 77.5 78.7 79 79.2 80 80.1 82 85.7 86.5 86.7 88
.2 89.7 90.6 90.9 92.2 92.6 93 93.1 95.6 96.2 96.5 }
Rendezesi ido: 23 mikromasodperc [us]

Beszuro rendezes:
=====
Kiindulo allapot => { 21 57 67.3 4.11 65 4.5 66.8 39.9 85.7 54.6 72.4 59.5 89.7 31.4 39.3 82 18.8 63 70.6 70.7 8.08 16.8 92.2 29 60.2 93.1 16.1 17
.9 55.1 16.9 48.8 36.3 72.8 61.7 58 51 67.4 58.8 31.4 19.3 59.2 22.7 45.8 93 80.1 30.4 96.5 53.5 48.3 14.3 86.7 48.6 96.2 78.7 30.7 95.6 42.2 17.6
75.6 26.4 14.6 30.2 1.08 12.7 26.8 26.3 79 77.5 18.4 11.1 79.2 20.9 42.4 30.8 88.2 45.8 32.8 90.9 24.6 46.6 21 13 32 25 38.8 36.6 21.7 90.6 80 54
.3 49.1 35.5 51.4 69.3 92.6 10 86.5 58.5 30.8 10.9 }
Vegallapot => { 1.08 4.11 4.5 8.08 10 10.9 11.1 12.7 13 14.3 14.6 16.1 16.8 16.9 17.6 17.9 18.4 18.8 19.3 20.9 21 21 21.7 22.7 24.6 25 26.3
26.4 26.8 29 30.2 30.4 30.7 30.8 30.8 31.4 31.4 32 32.8 35.5 36.3 36.6 38.8 39.3 39.9 42.2 42.4 45.8 45.8 46.6 48.3 48.6 48.8 49.1 51 51.4 53.5 54
.3 54.6 55.1 57 58 58.5 58.8 59.2 59.5 60.2 61.7 63 65 66.8 67.3 67.4 69.3 70.6 70.7 72.4 72.8 75.6 77.5 78.7 79 79.2 80 80.1 82 85.7 86.5 86.7 88
.2 89.7 90.6 90.9 92.2 92.6 93 93.1 95.6 96.2 96.5 }
Rendezesi ido: 8 mikromasodperc [us]

Kozvetlen kivalasztó rendezes
=====
Kiindulo allapot => { 21 57 67.3 4.11 65 4.5 66.8 39.9 85.7 54.6 72.4 59.5 89.7 31.4 39.3 82 18.8 63 70.6 70.7 8.08 16.8 92.2 29 60.2 93.1 16.1 17
.9 55.1 16.9 48.8 36.3 72.8 61.7 58 51 67.4 58.8 31.4 19.3 59.2 22.7 45.8 93 80.1 30.4 96.5 53.5 48.3 14.3 86.7 48.6 96.2 78.7 30.7 95.6 42.2 17.6
75.6 26.4 14.6 30.2 1.08 12.7 26.8 26.3 79 77.5 18.4 11.1 79.2 20.9 42.4 30.8 88.2 45.8 32.8 90.9 24.6 46.6 21 13 32 25 38.8 36.6 21.7 90.6 80 54
.3 49.1 35.5 51.4 69.3 92.6 10 86.5 58.5 30.8 10.9 }
Vegallapot => { 1.08 4.11 4.5 8.08 10 10.9 11.1 12.7 13 14.3 14.6 16.1 16.8 16.9 17.6 17.9 18.4 18.8 19.3 20.9 21 21 21.7 22.7 24.6 25 26.3
26.4 26.8 29 30.2 30.4 30.7 30.8 30.8 31.4 31.4 32 32.8 35.5 36.3 36.6 38.8 39.3 39.9 42.2 42.4 45.8 45.8 46.6 48.3 48.6 48.8 49.1 51 51.4 53.5 54
.3 54.6 55.1 57 58 58.5 58.8 59.2 59.5 60.2 61.7 63 65 66.8 67.3 67.4 69.3 70.6 70.7 72.4 72.8 75.6 77.5 78.7 79 79.2 80 80.1 82 85.7 86.5 86.7 88
.2 89.7 90.6 90.9 92.2 92.6 93 93.1 95.6 96.2 96.5 }
```

```
E:\C:\Gyrfu\FunctionOverloading\Debug\Gyrfu\FunctionOverloading.exe
.2 89.7 98.6 98.9 92.2 92.6 93 93.1 95.6 96.2 96.5 }
Rendezesi ido: 20 mikromasodperc [us]

Shell-rendezes
=====
Kiindulo allapot => { 21 57 67.3 4.11 65 4.5 66.8 39.9 85.7 54.6 72.4 59.5 89.7 31.4 39.3 82 18.8 63 70.6 70.7 8.08 16.8 92.2 29 60.2 93.1 16.1 17
.9 55.1 16.9 48.8 36.3 72.8 61.7 58 51 67.4 58.8 31.4 19.3 59.2 22.7 45.8 93 80.1 30.4 96.5 53.5 48.3 14.3 86.7 48.6 96.2 78.7 30.7 95.6 42.2 17.6
75.6 26.4 14.6 30.2 1.08 12.7 26.8 26.3 79 77.5 18.4 11.1 79.2 20.9 42.4 30.8 88.2 45.8 32.8 90.9 24.6 46.6 21 13 32 25 38.8 36.6 21.7 90.6 80 54
.3 49.1 35.5 51.4 69.3 92.6 10 86.5 58.5 30.8 10.9 }
Vegallapot => { 1.08 4.11 4.5 8.08 10 10.9 11.1 12.7 13 14.3 14.6 16.1 16.8 16.9 17.6 17.9 18.4 18.8 19.3 20.9 21 21 21.7 22.7 24.6 25 26.3
26.4 26.8 29 30.2 30.4 30.7 30.8 30.8 31.4 31.4 32 32.8 35.5 36.3 36.6 38.8 39.3 39.9 42.2 42.4 45.8 46.6 48.3 48.6 48.8 49.1 51 51.4 53.5 54
.3 54.6 55.1 57 58 58.5 58.8 59.2 59.5 60.2 61.7 63 65 66.8 67.3 67.4 69.3 70.6 70.7 72.4 72.8 75.6 77.5 78.7 79 79.2 80 80.1 82 85.7 86.5 86.7 88
.2 89.7 90.6 90.9 92.2 92.6 93 93.1 95.6 96.2 96.5 }
Rendezesi ido: 8 mikromasodperc [us]

Buborekrendezes
=====
Kiindulo allapot => { b h r w z n c e i c z x a v s y a k k u x y s v t b f x t t b v q a n j a q j o v o x n l d i m y z o i q m y h w k c b l w
i m q x q e e l a n z l x z s m z i z u w p k u u b q l i n p q o k l c t g }
Vegallapot => { a a a a a b b b b b c c c c d e e e f g h i i i i i j j k k k k k l l l l l m m m m m n n n n n o o o o p p q q q q q q
q r s s s t t t t t u u u u v v v v w w w x x x x x y y y z z z z z z }
Rendezesi ido: 34 mikromasodperc [us]

Beszuro rendezes:
=====
Kiindulo allapot => { b h r w z n c e i c z x a v s y a k k u x y s v t b f x t t b v q a n j a q j o v o x n l d i m y z o i q m y h w k c b l w
i m q x q e e l a n z l x z s m z i z u w p k u u b q l i n p q o k l c t g }
Vegallapot => { a a a a a b b b b b c c c c d e e e f g h i i i i i j j k k k k k l l l l l m m m m m n n n n n o o o o p p q q q q q q
q r s s s t t t t t u u u u v v v v w w w x x x x x y y y z z z z z z }
Rendezesi ido: 11 mikromasodperc [us]

Kozvetlen kivalasrto rendezes
=====
Kiindulo allapot => { b h r w z n c e i c z x a v s y a k k u x y s v t b f x t t b v q a n j a q j o v o x n l d i m y z o i q m y h w k c b l w
i m q x q e e l a n z l x z s m z i z u w p k u u b q l i n p q o k l c t g }
Vegallapot => { a a a a a b b b b b c c c c d e e e f g h i i i i i j j k k k k k l l l l l m m m m m n n n n n o o o o p p q q q q q q
q r s s s t t t t t u u u u v v v v w w w x x x x x y y y z z z z z z }
```

46. ábra: A program futásának eredménye

A feladathoz kapcsolódó magyarázatok

A „tradicionális” függvényeket alkalmazó mintapéldából az a következtetés vonható le, hogy a függvények rugalmasságának akadálya a szigorú típusellenőrzés. Így felmerülhet az olyan jellegű igény, hogy egy algoritmust egy függvénynevet használva, különböző típusú paraméterekkel is működjön. A függvénynevének azonossága fontos, mivel helyes nevezéktan használata esetén egy függvénynév mindig utal annak feladatára. Az ilyen igények kielégítésére a C++ nyelv a függvénynevek túlterhelésének lehetőségét kínálja. A túlterhelés

erőssége abban mutatkozik meg, hogy azonos hatókörben (ez a fentebbi példában az osztályunkat jelenti), azonos névvel, azonban eltérő típusú és mennyiségű paraméterlistával (gyűjtő néven: paraméterszignatúrával) eltérő függvényeket implementálhatunk. Ennek tudatában olvashatjuk alább a fentebbi minta feladathoz kapcsolódó magyarázatot.

Az osztály deklarációját a (**SortByFunctionOverloading.h**) állományban adjuk meg. Az osztály egy magán (**private**) láthatóságú konstans adattaggal rendelkezik, amely a jelenlegi feladat esetében, eltárolja a rendezni kívánt tömbök méretét. Konstansok használata különböző tömb méretek meghatározásánál bevett szokás, mivel ha a méretérték valamilyen okból kifolyólag változtatásra kerül, elég csak az állandó értékét átírni és nem kell a különböző műveletek (tömb feltöltés, rendezés vagy épp megjelenítés) végrehajtását segítő részeket nyomon követni és módosítani. Egy osztályon belül konstans megadása az alábbi formában történhet:

```
static const int SIZE = 100;
```

A **static** kulcsszó jelöli azt, hogy ezt a konstanst megosztva használják az osztályok példányai, így az közvetlenül az osztályhoz tartozik. Fontos szabály, hogy amíg a statikus osztály adattagokat az osztályon kívül inicializáljuk, addig az állandókat az osztályon belül is el láthatjuk kezdőértékkel. A **const** kulcsszó jelöli, hogy ebben az esetben egy állandóról van szó, majd következik a konstans típusának megadása, ezt követi a konstans neve, amelyet a kód könnyebb olvashatósága végett (jobban elkülönül a környezetétől) nagybetűvel került megfogalmazásra, végül a konstans kezdőértékkel történő ellátása következik, ebben az esetben 100 értékkel. Az adattag megfogalmazását követően létrehozásra kerül egy explicit, alapértelmezett, paraméter nélküli konstruktor. A következő függvények visszatérési típusaik **void**, amely jelentése szerint a függvények nem térnek vissza különböző típusú értékekkel:

- *ArrayFill(int *integerNumbers);*
- *ArrayFill(double *doubleNumbers);*
- *ArrayFill(char *characters);*
- *BubbleSort(int *integerArray);*
- *BubbleSort(double *doubleArray);*
- *BubbleSort(char *charArray);*
- *SelectionSort(int *integerArray);*
- *SelectionSort(double *doubleArray);*
- *SelectionSort(char *charArray);*
- *InsertionSort(int *integerArray);*
- *InsertionSort(double *doubleArray);*
- *InsertionSort(char *charArray);*
- *ShellSort(int *integerArray);*
- *ShellSort(double *doubleArray);*

- *ShellSort(char *charArray);*
- *PrintArray(const int *integerArray);*
- *PrintArray(const int *doubleArray);*
- *PrintArray(const int *charArray).*

A függvények paraméterlistája ugyan számmennyiségben nem, de paraméter típusokban eltér a korábbi példával ellentétben. A függvénynevek túlterhelési technikának köszönhetően ugyanazon függvénynevek, azonos hatókörben kerültek felhasználásra, eltérő típusú paraméterekkel. Az *ArrayCopy()* tagfüggvény **inline** módon került megvalósításra, a függvény felelőssége az eredeti tömb egy másolatának létrehozása. Természetesen e függvény is a különböző típus szám mennyiségétől függően került implementálásra, így az adott példában három alkalommal. A deklaráció végén megfogalmazásra került a(z) *~SortByFunctionOverloading()* nevezetű, üres paraméterlistával rendelkező destruktorktor, amely az erőforrások felszabadításáért lesz felelős.

A (**SortByFunctionOverloading.cpp**) állomány tartalmazza az osztály megvalósítását, tagfüggvényeinek implementálását. A tagfüggvények neve előtt a `::` hatókör operátor kerül felhasználásra azon célból, hogy egyértelműsítésre kerüljön melyik osztály tagfüggvényét szeretnénk kifejezni. Az *ArrayFill(int *integerNumbers)* függvény feladata a paraméterül kapott egész típusú tömb véletlen számokkal történő feltöltése 1 és 100 között. Az *ArrayFill(double *doubleNumbers)* függvény feladata a paraméterül kapott lebegőpontos típusú tömb véletlen számokkal történő feltöltése 1,0 és 100,0 között. Az *ArrayFill(char *characters)* függvény feladata a paraméterül kapott karakter típusú tömb véletlenbetűkkel történő feltöltése *a* és *z* között. A karakterek meghatározásánál az ASCII kódokat alkalmazzuk, ahol is véletlenszerűen kiválasztásra került egy számérték a betűket tartalmazó ASCII szám halmazból, majd azt típuskényszerítéssel karakterré alakítottuk. Az osztály implementációja során adott rendezési algoritmust végző tagfüggvények felelőssége a rájuk szabott rendezési műveletek implementálása. Ezek a tagfüggvények minden esetben paraméterül kapják a még rendezetlen tömböt, majd megjelenítik azt a konzolon, mint kiinduló állapot, majd közvetlenül a rendezési algoritmusok előtt elindítjuk az időmérést, amelyet a rendezés végeztével leállítunk. Minden olyan esetben, ahol a függvény feladata egy adott rendezési algoritmus elvégzése eltérő típusú tömbön, a túlterhelés technika került alkalmazásra. Ezt követően megjelenítjük – típustól függően – a már rendezett tömb – a *PrintArray()* függvény segítségével – értékeket, valamint azt, hogy mennyi időbe került a rendezés. Az idő méréséhez egy **chrono** nevezetű **header** fájlt használunk, továbbá szükséges az **std** névtér behívása is, ahol az egyes elemek implementálása megtalálható. Az aktuális időpont lekérését a következő utasítással végezzük, amelyet a **start** lokális változóban tároljuk:

```
chrono::steady_clock::time_point start = chrono::steady_clock::now();
```

A rendezési algoritmusok lefutását követően ismét lekérjük az aktuális időt, amelyet az **end** lokális változó kap értékül:

```
chrono::steady_clock::time_point end = chrono::steady_clock::now();
```

A rendezéshez szükséges idő megjelenítését a következő utasítással végezzük, ahol mikromásodpercben kiszámoljuk az eltelt időtartamot (**end - start**):

```
chrono::duration_cast<chrono::microseconds>(end - start).count()
```

A (**main.cpp**) állomány tartalmazza a **main()** függvényt, a program belépési pontját. A főfüggvényben történő dinamikus példányosítása után hívjuk meg a szükséges függvényeket. A példányosítás során átadásra kerülnek azon tömb(ök), amelyeken a rendezési műveletek elvégezzük, majd gondoskodunk a lefoglalt erőforrások felszabadításáról a destruktorkívételének segítségével. Ahhoz, hogy véletlen szám generálás során ne ugyanazokat a számokat kapjuk, szükséges a processzor idő nullázására, amelyet a következő utasítással végzünk:

```
srand((unsigned)time(0));
```

Az objektumpéldányon keresztül történő függvény hívás során a fordító illesztést végez. Ez azt jelenti, hogy attól függően, milyen típusú tömb kerül átadásra az annak megfelelő függvényt hívja meg, tehát önmagában még a függvény név nem határozza meg melyik függvény kerüljön hívásra. Röviden a fordító a hívás során használt paraméterezést sorban egyezteteti az összes azonos nevű függvény paraméterezésével és kizárólag a megfelelő kerül hívásra. Több eshetőség is előfordulhat a mintaillesztés során:

- Teljes típus egyezés;
- Automatikus konverziók utáni típus egyezés;
- Egyetlen illeszkedést sem talál, ekkor a fordító hibát jelez;
- Több illeszkedést talál, ekkor is hibát jelez a fordító.

Minta példa függványsablonok használatára Visual Studio Community fejlesztőkörnyezetben

Írjunk egy **SortByFunctionTemplate** nevezetű osztályt, amely néhány rendezési algoritmust implementál. A rendezési műveleteket egy száz elemű, általános típusú tömbön végezzük el, úgy, hogy közben a rendezési algoritmus hatékonyságát is prezentáljuk a rendezési művelethez szükséges idővel. A képernyőn jelenítsük meg a kiinduló és végállapotokat, valamint a rendezéshez szükséges időtartamot mikromásodpercben. Az osztály adatai a következők:

- **SIZE** nevezetű statikus, konstans, egész típusú érték, amelyet 100-al inicializálunk.

Ügyeljünk az egységbezárás elvére, tehát az adattag(ok) **private** láthatóságúak legyenek!

Valósítsuk meg az osztály metódusait:

- Írjuk meg az osztály explicit, alapértelmezett konstruktorát.
- Írjunk egy *ArrayFill(T *arrayToFill)* függvényt, amelynek felelőssége a 100 elemű tömb, karakter típus esetén véletlen karakterekkel (a és z között), egész szám típus esetén véletlen számokkal (1 és 100 között), lebegőpontos szám típus esetén véletlen számokkal (1.0 és 100.0 között) történő feltöltése.
- Írjunk egy *ArrayCopy(const T *arrayToCopy, T *copiedArray)* függvényt, amely felelőssége az eredeti tömb átmásolása egy átmeneti tömbbe. Ennek célja, hogy a rendezési algoritmusok hatékonyság mérése ne torzuljon, mivel ha mindig újra inicializálnánk a rendezendő tömböt, akkor folyamatosan eltérő értéket tartalmazna, így szükséges minden egyes rendezési algoritmusnál ugyanazzal a tömb értékekkel dolgozni.
- Írjunk egy *BubbleSort(T *arrayToBubbleSort)* függvényt, amely a buborékos rendezés algoritmusát valósítja meg általános (bármilyen) típus esetén.
- Írjunk egy *InsertionSort(T *arrayToInsertionSort)* függvényt, amely a beszűrő rendezés algoritmusát valósítja meg.
- Írjunk egy *SelectionSort(T *arrayToSelectionSort)* függvényt, amely a közvetlen kiválasztásos rendezés algoritmusát valósítja meg általános (bármilyen) típus esetén.
- Írjunk egy *ShellSort(T *arrayToShellSort)* függvényt, amely a Shell-rendezés algoritmusát valósítja meg általános (bármilyen) típus esetén.
- Írjunk egy *PrintArray(T *arrayToPrint)* függvényt, amelynek felelőssége a paraméterül kapott tömb értékeinek megjelenítése a képernyőn, adott formátumban általános (bármilyen) típus esetén.
- Ne feledkezzünk meg az objektumpéldány által lefoglalt erőforrás felszabadításáról, a destruktor megírásáról sem.

Az osztály implementálását követően dinamikus módszerrel (a memória dinamikus területén [**heap**]) készítsünk egy objektumpéldányt. Az objektumpéldány segítségével a feladat leírásnak megfelelően hívjuk meg az osztály egyes tagfüggvényeit. A megvalósítás során, ha tudjuk, hogy a függvénynek átadott paraméter a függvény futása során nem változik, használjunk **const** jelzőt. Amennyiben nincs tovább szükség az objektum példányra, gondoskodjunk annak megszüntetéséről, elkerülve ezzel a memóriaszivárgást. Mielőtt a feladatnak neki látnánk, tervezzük meg az statikus modelleket.

Statikus tervezés

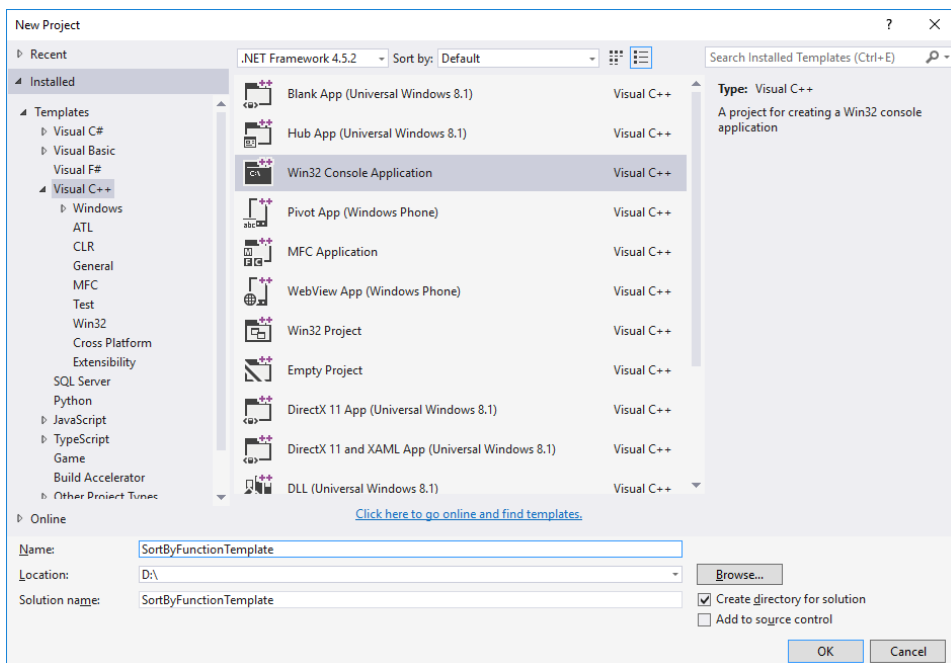
SortByFunctionTemplate	
-	<u>SIZE: int = 100 {readOnly}</u>
+	ArrayCopy(T*, T*): void
+	ArrayFill(T*): void
+	BubbleSort(T*): void
+	InsertionSort(T*): void
+	PrintArray(T*): void
+	SelectionSort(T*): void
+	ShellSort(T*): void
+	SortByFunctionTemplate()
+	~SortByFunctionTemplate()

47. ábra: Osztály diagram

A projekt létrehozása

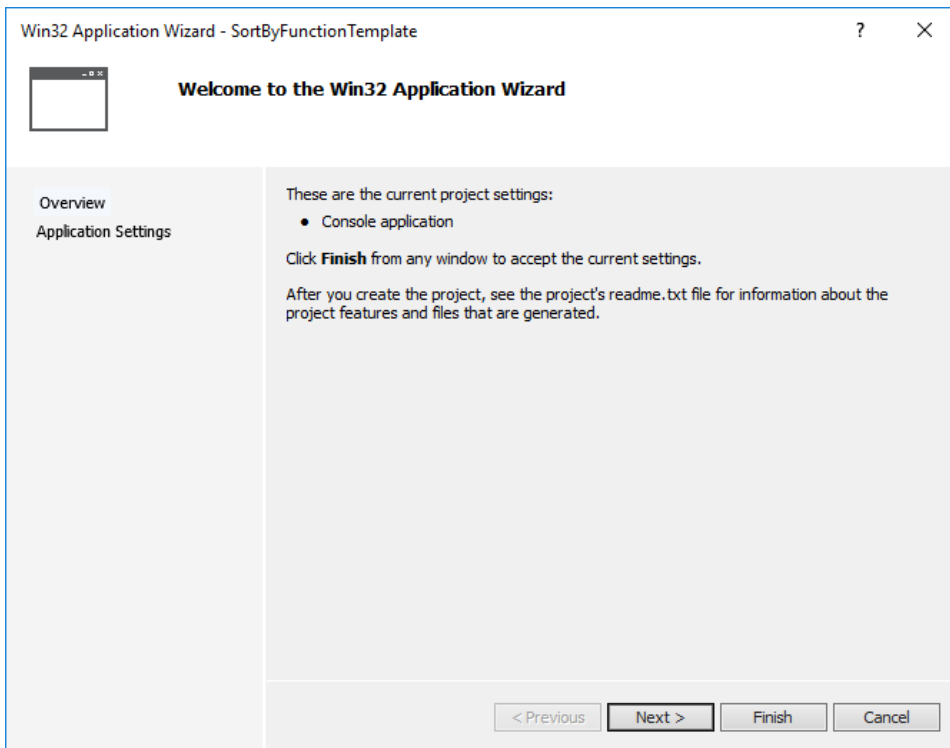
A projekt megvalósítása során konzolalkalmazást készítünk, ahol a Visual Studio, szabványos C++-nak megfelelő szintaktikát alkalmaz. A projekt három fájlban kerül implementálásra. Az alkalmazás belépési pontja a main függvény, amely a három fájl közül az egyikben fog helyet foglalni. Az előzőekben megfogalmazott minta példa kerül implementálásra és összeállításra, továbbá Windows operációs rendszeren futtatható kóddá történő lefordításra **Visual Studio Community** fejlesztőkörnyezet felhasználásával. A lefordított 32 bites alkalmazás konzolban kerül futtatásra.

Új projekt a **File** menü **New** menüpontjában hozható létre, a **Project** almenüt kiválasztva, vagy akár a **Ctrl+Shift+N** gyorsindító gombkombinációval. Új projekt kiválasztása után megjelenő ablakban a közül a **Visual C++ Win32** sablont (**Template**) kell kiválasztani és a **Win32 Console Application**-t, mely az alábbi ábrán látható.



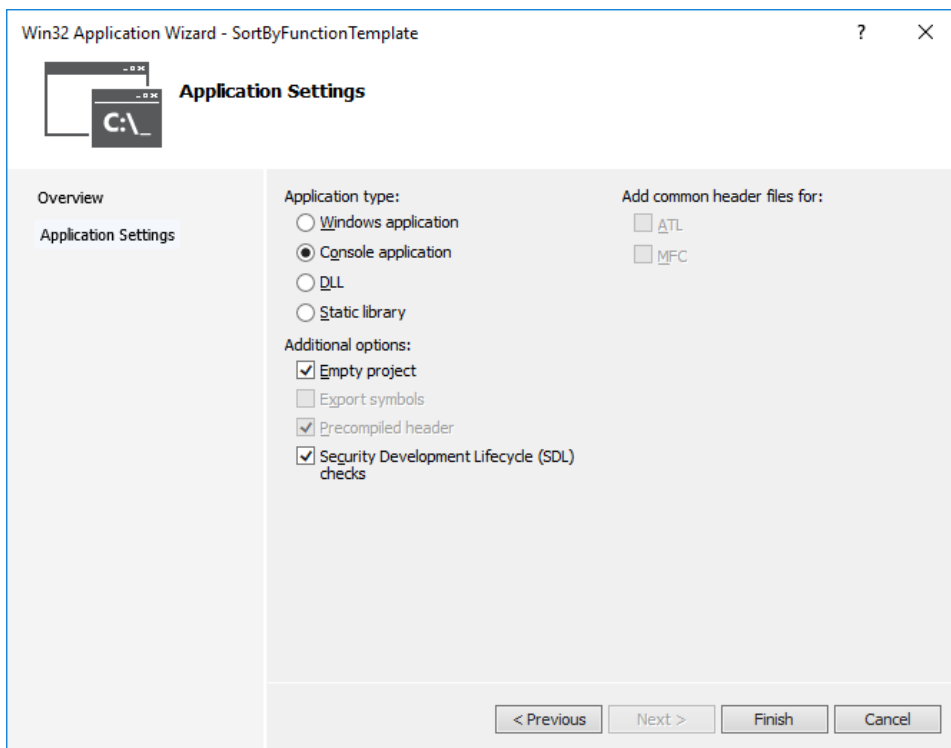
48. ábra: Win32 Console Application projekt létrehozása

Az ablak alsó részén a projekt nevének (**Name**) a példának megfelelően az **Sort** név került megadásra. Ebben a részben lehet módosítani a Projekthez kapcsolódó fájlok mentési helyét is (**Location**). Egy összetett program általában több projektből tevődik össze, ezeket egységes rendszerré a megoldás (**Solution**) fogja össze. Az OK gombra kattintást követően elindul az alkalmazás varázsló, mely a következő ábrán látható.



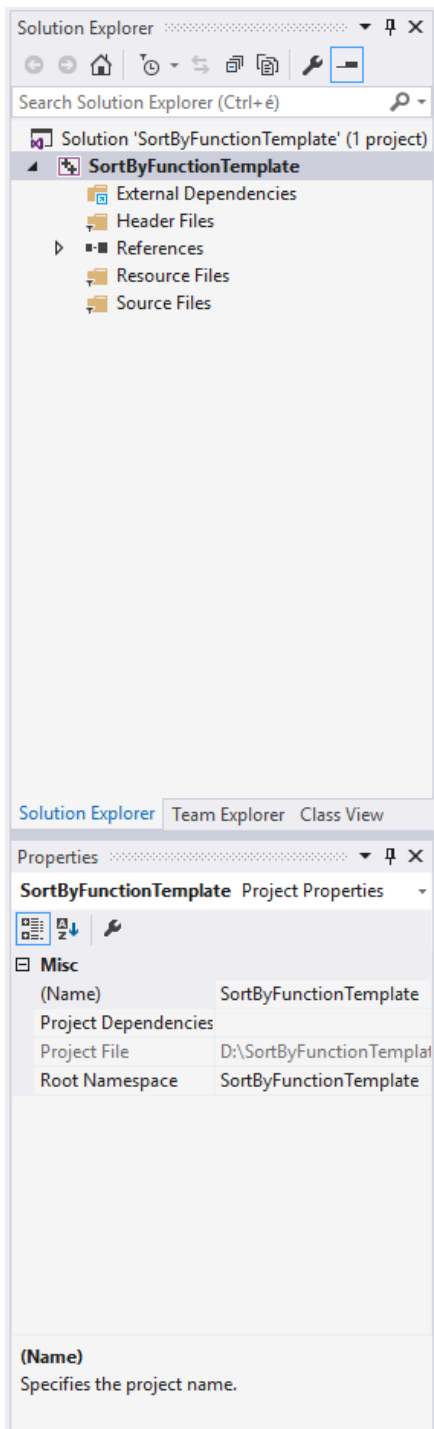
49. ábra: Win32 konzol alkalmazás varázsló

A **Next** gombra kattintva a beállítások ablak jelenik meg, itt megadhatjuk az alkalmazás típusát (**Console Application**), és néhány kiegészítő opciót. Mivel egyszerű alkalmazást készítünk, ezért célszerű az üres projekt (**Empty project**) opció kiválasztása, ahogy az alábbi képen látszódik.



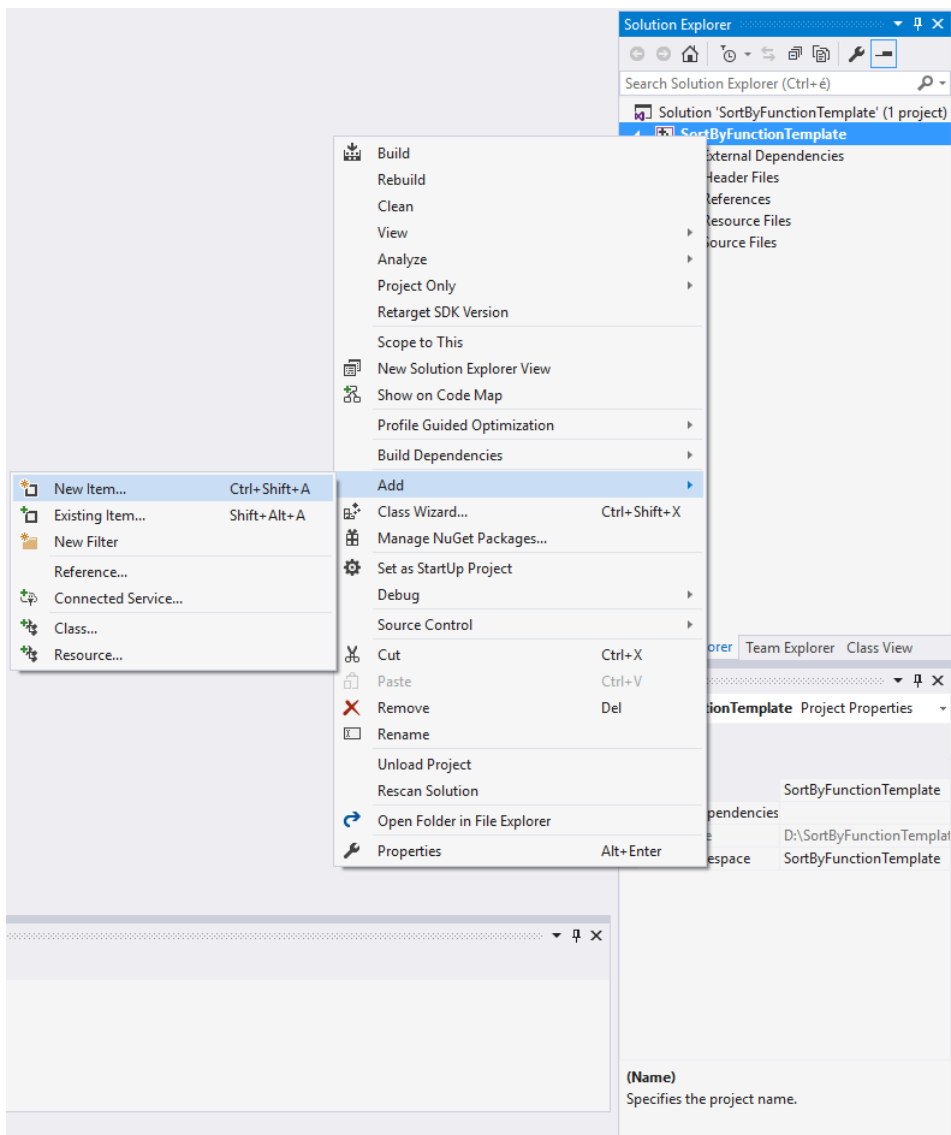
50. ábra: Alkalmazás beállításai

A **Finish** megnyomását követően a képernyő jobb szélén, a megoldás ablaka (**Solution Explorer**), alatta a beállítások ablak jelenik meg, ahogy a következő ábrán látható.



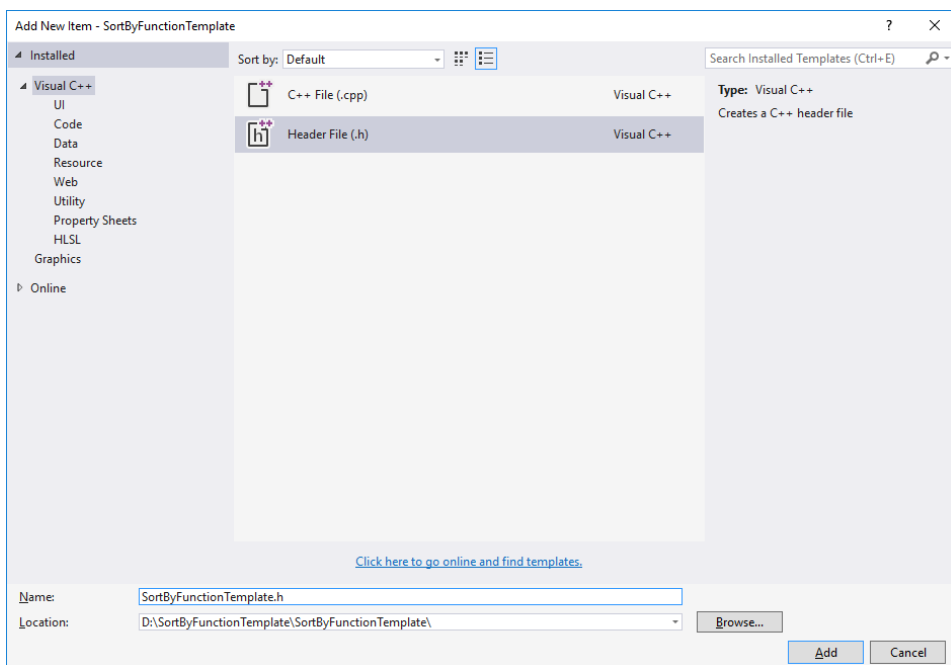
51. ábra: Megoldás és beállítások ablak

A program kódjának forrás és header fájljait (**Source Files, Header Files**) a megoldás ablakban a kiválasztott fájl típusra az egér jobb gombjával kattintva adható hozzá az alábbi ábrának megfelelően vagy a **Shift+A** billentyűkombináció megnyomásával.



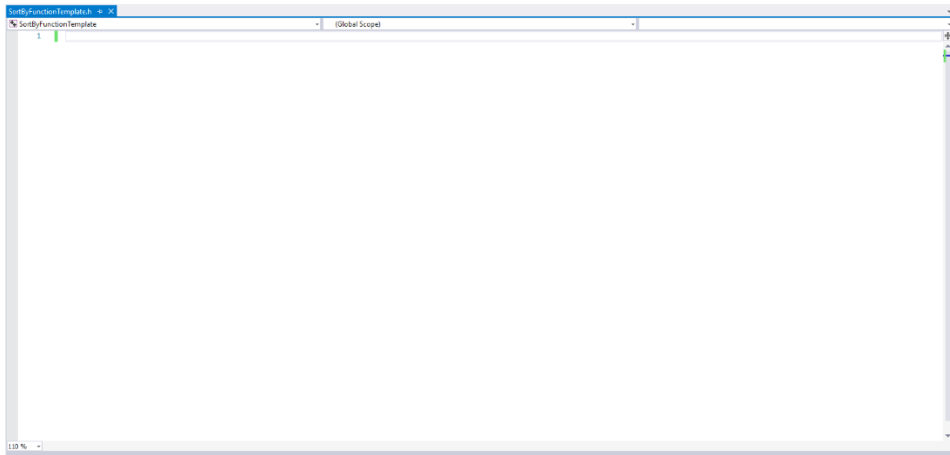
52. ábra: Új header fájl hozzáadása a projekthez

Új header fájl (Header File) választásával és a név (Name) az **SortByFunctionTemplate.h** megadásával hozható létre a **SortByFunctionTemplate.h** fájl, az alábbi ábrának megfelelően.



53. ábra: Új header fájl hozzáadása a projekthez, név megadása

Az új fájl hozzáadásával a képernyő középső részén megjelenik a forráskód szerkesztő az alábbi ábrának megfelelően.



54. ábra: Forráskód szerkesztő

SortByFunctionTemplate.h

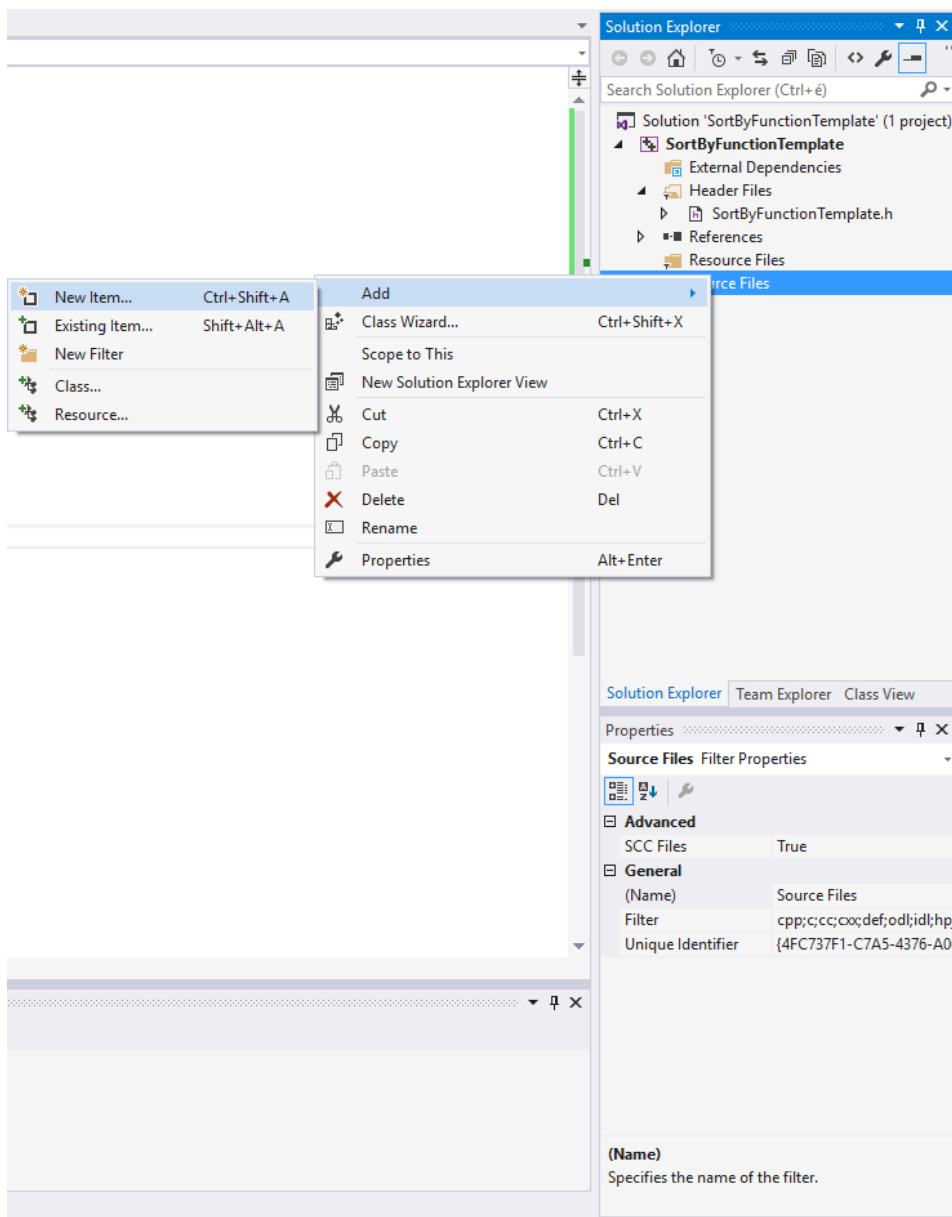
```
#ifndef SORTBYFUNCTIONTEMPLATE_H
#define SORTBYFUNCTIONTEMPLATE_H

class SortByFunctionTemplate
{
private:
    static const int SIZE = 100;

public:
    SortByFunctionTemplate() { }
    template <typename T> void ArrayFill(T *arrayToFill);
    template <typename T> void ArrayCopy(const T *arrayToCopy, T
    *copiedArray) { for (int i = 0; i < SIZE; ++i) copiedArray[i] =
    arrayToCopy[i]; }
    template <typename T> void BubbleSort(T *arrayToBubbleSort);
    template <typename T> void SelectionSort(T *arrayToSelectionSort);
    template <typename T> void InsertionSort(T *arrayToInsertionSort);
    template <typename T> void ShellSort(T *arrayToShellSort);
    template <typename T> void PrintArray(const T *arrayToCopy);
    ~SortByFunctionTemplate() { }
};

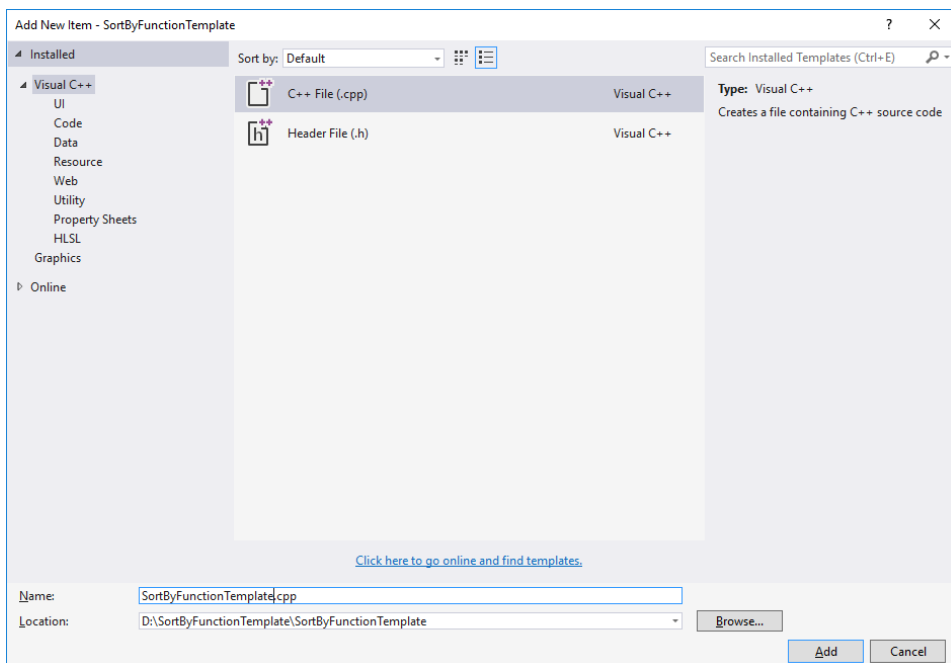
#endif
```

A deklarált osztályok megvalósításait a **SortByFunctionTemplate.cpp** forrásfájlok tartalmazzák, melyeket a **header** fájlokhoz hasonlóan, de a forrásfájl (**Source File**) hozzáadásával hozhatunk létre.



55. ábra: Új forrásfájl hozzáadása a projekthez

Új C++ fájl (C++ **File**) választásával és a név (**Name**) a **SortByFunctionTemplate.cpp** megadásával hozható létre.



56. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

SortByFunctionTemplate.cpp

```
#include <iostream>
#include <ctime>
#include <chrono>
#include "SortByFunctionTemplate.h"

using namespace std;

template <typename T> void SortByFunctionTemplate::ArrayFill(T *arrayToFill)
{
    srand((unsigned)time(0));
    for (int i = 0; i < SIZE; ++i)
        typeid(T) == typeid(char) ? arrayToFill[i] = (T)(rand() % 26) + 97
        : arrayToFill[i] = 1 + (T)rand() / (T)(RAND_MAX / (100 - 1));
}

template <typename T> void SortByFunctionTemplate::BubbleSort(T
*arrayToBubbleSort)
{
    cout << "Buborekrendezes" << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(arrayToBubbleSort);

    T temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int i = 0; i < SIZE; i++)
    {
        for (int j = SIZE - 1; j > i; j--)
        {
            if (arrayToBubbleSort[j] < arrayToBubbleSort[j - 1])
            {
                temp = arrayToBubbleSort[j - 1];
                arrayToBubbleSort[j - 1] = arrayToBubbleSort[j];
                arrayToBubbleSort[j] = temp;
            }
        }
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => " << endl; PrintArray(arrayToBubbleSort);
    cout << "Rendezesi ido: " <<
    chrono::duration_cast<chrono::microseconds>(end - start).count() << "
    mikromasodperc [us]";
    cout << endl; cout << endl;
}

template <typename T> void SortByFunctionTemplate::InsertionSort(T
*arrayToInsertionSort)
{
    cout << "Beszuro rendezes: " << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(arrayToInsertionSort);

    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int j = 1; j < SIZE; j++)
    {
        T key = arrayToInsertionSort[j];
        int i = j - 1;
        while (i > -1 && arrayToInsertionSort[i] > key)
```

```

        {
            arrayToInsertionSort[i + 1] = arrayToInsertionSort[i];
            i = i - 1;
        }
        arrayToInsertionSort[i + 1] = key;
    }

    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(arrayToInsertionSort);
    cout << "Rendezesi ido: " <<
    std::chrono::duration_cast<std::chrono::microseconds>(end -
    start).count() << " mikromasodperc [us]";
    cout << endl; cout << endl;
}

template <typename T> void SortByFunctionTemplate::SelectionSort(T
*arrayToSelectionSort)
{
    cout << "Kozvetlen kivalaszto rendezes" << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(arrayToSelectionSort);

    int position;
    T temp;

    chrono::steady_clock::time_point start = chrono::steady_clock::now();
    for (int i = 0; i < SIZE - 1; i++)
    {
        position = i;
        for (int j = i + 1; j < SIZE; j++)
        {
            if (arrayToSelectionSort[j] < arrayToSelectionSort[position])
                position = j;
        }

        if (position != i)
        {
            temp = arrayToSelectionSort[i];
            arrayToSelectionSort[i] = arrayToSelectionSort[position];
            arrayToSelectionSort[position] = temp;
        }
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Vegallapot => "; PrintArray(arrayToSelectionSort);
    cout << "Rendezesi ido: " <<
    std::chrono::duration_cast<std::chrono::microseconds>(end -
    start).count() << " mikromasodperc [us]";
    cout << endl; cout << endl;
}

template <typename T> void SortByFunctionTemplate::ShellSort(T
*arrayToShellSort)
{
    cout << "Shell-rendezes" << endl;
    cout << "===== " << endl;
    cout << "Kiindulo allapot =>"; PrintArray(arrayToShellSort);

    T temp;
    chrono::steady_clock::time_point start = chrono::steady_clock::now();

    for (int gap = SIZE / 2; gap > 0; gap /= 2)
        for (int i = gap; i < SIZE; i++)

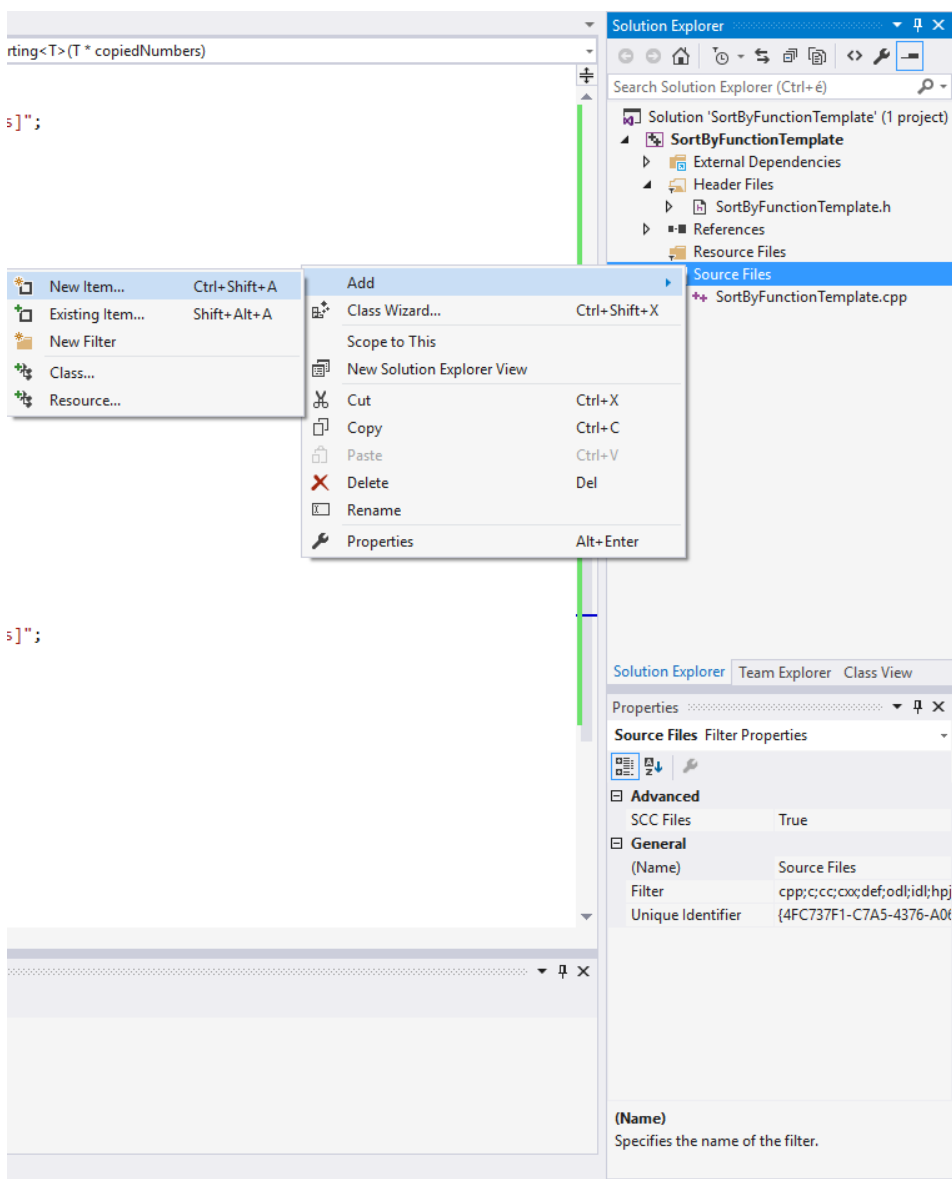
```

```

        for (int j = i - gap; j >= 0 && arrayToShellSort[j] >
            arrayToShellSort[j + gap]; j -= gap) {
            temp = arrayToShellSort[j];
            arrayToShellSort[j] = arrayToShellSort[j + gap];
            arrayToShellSort[j + gap] = temp;
        }

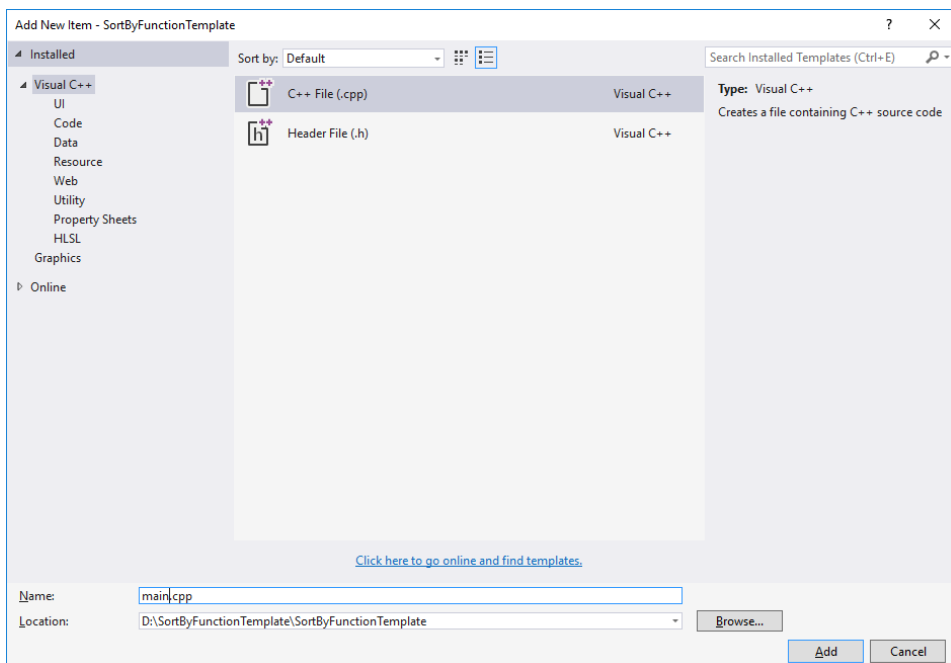
        chrono::steady_clock::time_point end = chrono::steady_clock::now();
        cout << "Vegallapot => "; PrintArray(arrayToShellSort);
        cout << "Rendezesi ido: " <<
            std::chrono::duration_cast<std::chrono::microseconds>(end -
            start).count() << " mikromasodperc [us]";
        cout << endl; cout << endl;
    }
    template <typename T> void SortByFunctionTemplate::PrintArray(const T
    *arrayToPrint)
    {
        cout << " { ";
        for (int i = 0; i < SIZE; ++i)
            cout << arrayToPrint[i] << " ";
        cout << " } " << endl;
    }
}

```



57. ábra: Új forrásfájl hozzáadása a projekthez

C++ fájl (**C++ File**) választásával és a név (**Name**) a **main.cpp** megadásával hozható létre.



58. ábra: Új forrásfájl hozzáadása a projekthez, név megadása

main.cpp

```
#include <iostream>
#include <conio.h>
#include <ctime>
#include "SortByFunctionTemplate.cpp"

using namespace std;

int main()
{
    const int SIZE(100);
    int *integerNumbers = new int[SIZE];
    int *copiedIntegerNumbers = new int[SIZE];
    double *doubleNumbers = new double[SIZE];
    double *copiedDoubleNumbers = new double[SIZE];
    char *characters = new char[SIZE];
    char *copiedCharacters = new char[SIZE];
    SortByFunctionTemplate *objectOfSortByFunctionTemplate = new
SortByFunctionTemplate();

    objectOfSortByFunctionTemplate->ArrayFill<int>(integerNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<int>(integerNumbers,
copiedIntegerNumbers);
    objectOfSortByFunctionTemplate->BubbleSort<int>(copiedIntegerNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<int>(integerNumbers,
copiedIntegerNumbers);
    objectOfSortByFunctionTemplate-
>InsertionSort<int>(copiedIntegerNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<int>(integerNumbers,
copiedIntegerNumbers);
    objectOfSortByFunctionTemplate-
>SelectionSort<int>(copiedIntegerNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<int>(integerNumbers,
copiedIntegerNumbers);
    objectOfSortByFunctionTemplate->ShellSort<int>(copiedIntegerNumbers);

    objectOfSortByFunctionTemplate->ArrayFill<double>(doubleNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<double>(doubleNumbers,
copiedDoubleNumbers);
    objectOfSortByFunctionTemplate->BubbleSort<double>(copiedDoubleNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<double>(doubleNumbers,
copiedDoubleNumbers);
    objectOfSortByFunctionTemplate-
>InsertionSort<double>(copiedDoubleNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<double>(doubleNumbers,
copiedDoubleNumbers);
    objectOfSortByFunctionTemplate-
>SelectionSort<double>(copiedDoubleNumbers);
    objectOfSortByFunctionTemplate->ArrayCopy<double>(doubleNumbers,
copiedDoubleNumbers);
    objectOfSortByFunctionTemplate->ShellSort<double>(copiedDoubleNumbers);
    objectOfSortByFunctionTemplate->ArrayFill<char>(characters);
```

```

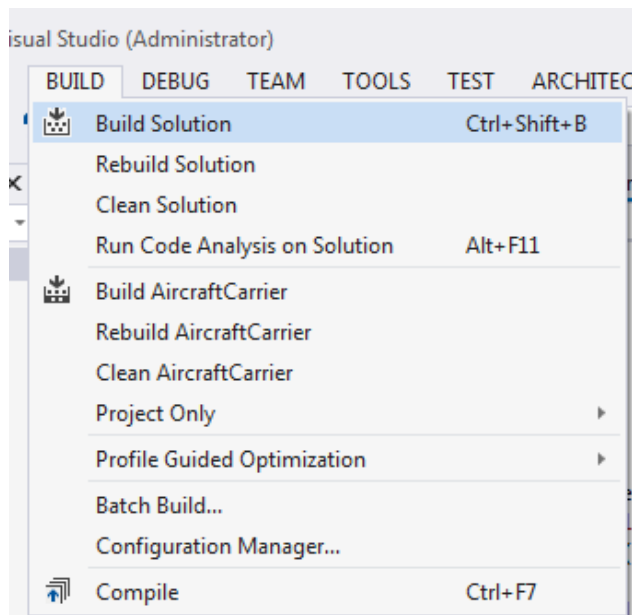
    objectOfSortByFunctionTemplate->ArrayCopy<char>(characters,
    copiedCharacters);
    objectOfSortByFunctionTemplate->BubbleSort<char>(copiedCharacters);
    objectOfSortByFunctionTemplate->ArrayCopy<char>(characters,
    copiedCharacters);
    objectOfSortByFunctionTemplate->InsertionSort<char>(copiedCharacters);
    objectOfSortByFunctionTemplate->ArrayCopy<char>(characters,
    copiedCharacters);
    objectOfSortByFunctionTemplate->SelectionSort<char>(copiedCharacters);
    objectOfSortByFunctionTemplate->ArrayCopy<char>(characters,
    copiedCharacters);
    objectOfSortByFunctionTemplate->ShellSort<char>(copiedCharacters);

    delete objectOfSortByFunctionTemplate;
    delete[] copiedCharacters;
    delete[] characters;
    delete[] copiedDoubleNumbers;
    delete[] doubleNumbers;
    delete[] copiedIntegerNumbers;
    delete[] integerNumbers;

    _getch();
    return 0;
}

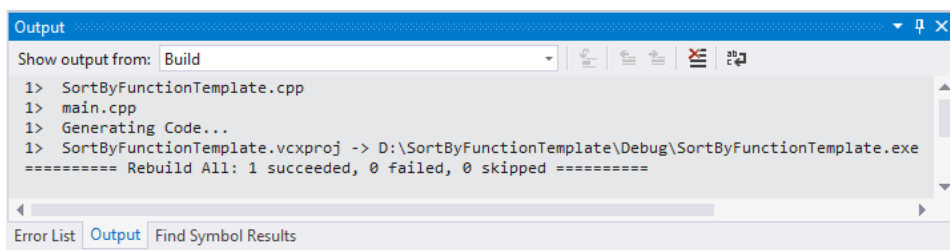
```

A forrásfájlok hozzáadása után a program forráskódja lefordítható, futtatható kód előállítását a **BUILD** menüpont **Build Solution** almenüpontjának kiválasztásával, vagy a **Ctrl+Shift+B** gyorsbillentyű kombináció megnyomásával tehető meg az alábbi ábrának megfelelően.



59. ábra: Program fordítása

Sikeres fordítás a kimenet (**Output**) ablakban látható.



60. ábra: Kimenet ablak

A program futásának eredménye konzolban jelenik meg, amely a következő ábrán látható.

```
D:\SortByFunctionTemplate\Debug\SortByFunctionTemplate.exe
Buborekrendezés
=====
Kiindulo allapot => { 41 67 46 90 7 2 67 10 39 27 41 84 84 86 46 10 33 49 9 42 41 1 54 27 75 83 87 97 31 90 23 64 55 76
77 4 23 73 23 31 2 72 63 77 54 73 80 70 100 61 43 9 6 99 19 56 14 4 70 34 79 1 3 41 29 12 9 71 1 81 5 74 3 91 99 15 22 7
0 76 80 31 49 43 97 43 63 32 1 23 63 89 14 47 23 53 95 90 14 47 1 }
Vegallapot => { 1 1 1 1 1 2 2 3 3 4 4 5 6 7 9 9 9 10 10 12 14 14 14 15 19 22 23 23 23 23 27 27 29 31 31 31 32 3
3 34 39 41 41 41 41 42 43 43 43 46 46 47 47 49 49 53 54 54 55 56 61 63 63 63 64 67 67 70 70 70 71 72 73 73 74 75 76 76 7
7 77 79 80 80 81 83 84 84 86 87 89 90 90 90 91 95 97 97 99 99 100 }
Rendezesi ido: 22 mikromasodperc [us]

Beszuro rendezes:
=====
Kiindulo allapot => { 41 67 46 90 7 2 67 10 39 27 41 84 84 86 46 10 33 49 9 42 41 1 54 27 75 83 87 97 31 90 23 64 55 76
77 4 23 73 23 31 2 72 63 77 54 73 80 70 100 61 43 9 6 99 19 56 14 4 70 34 79 1 3 41 29 12 9 71 1 81 5 74 3 91 99 15 22 7
0 76 80 31 49 43 97 43 63 32 1 23 63 89 14 47 23 53 95 90 14 47 1 }
Vegallapot => { 1 1 1 1 1 2 2 3 3 4 4 5 6 7 9 9 9 10 10 12 14 14 14 15 19 22 23 23 23 23 27 27 29 31 31 31 32 3
3 34 39 41 41 41 41 42 43 43 43 46 46 47 47 49 49 53 54 54 55 56 61 63 63 63 64 67 67 70 70 70 71 72 73 73 74 75 76 76 7
7 77 79 80 80 81 83 84 84 86 87 89 90 90 90 91 95 97 97 99 99 100 }
Rendezesi ido: 8 mikromasodperc [us]

Kozvetlen kivlasztos rendezes
=====
Kiindulo allapot => { 41 67 46 90 7 2 67 10 39 27 41 84 84 86 46 10 33 49 9 42 41 1 54 27 75 83 87 97 31 90 23 64 55 76
77 4 23 73 23 31 2 72 63 77 54 73 80 70 100 61 43 9 6 99 19 56 14 4 70 34 79 1 3 41 29 12 9 71 1 81 5 74 3 91 99 15 22 7
0 76 80 31 49 43 97 43 63 32 1 23 63 89 14 47 23 53 95 90 14 47 1 }
Vegallapot => { 1 1 1 1 1 2 2 3 3 4 4 5 6 7 9 9 9 10 10 12 14 14 14 15 19 22 23 23 23 23 27 27 29 31 31 31 32 3
3 34 39 41 41 41 41 42 43 43 43 46 46 47 47 49 49 53 54 54 55 56 61 63 63 63 64 67 67 70 70 70 71 72 73 73 74 75 76 76 7
7 77 79 80 80 81 83 84 84 86 87 89 90 90 90 91 95 97 97 99 99 100 }
Rendezesi ido: 16 mikromasodperc [us]

D:\SortByFunctionTemplate\Debug\SortByFunctionTemplate.exe
Shell-rendezés
=====
Kiindulo allapot => { 41 67 46 90 7 2 67 10 39 27 41 84 84 86 46 10 33 49 9 42 41 1 54 27 75 83 87 97 31 90 23 64 55 76
77 4 23 73 23 31 2 72 63 77 54 73 80 70 100 61 43 9 6 99 19 56 14 4 70 34 79 1 3 41 29 12 9 71 1 81 5 74 3 91 99 15 22 7
0 76 80 31 49 43 97 43 63 32 1 23 63 89 14 47 23 53 95 90 14 47 1 }
Vegallapot => { 1 1 1 1 1 2 2 3 3 4 4 5 6 7 9 9 9 10 10 12 14 14 14 15 19 22 23 23 23 23 27 27 29 31 31 31 32 3
3 34 39 41 41 41 41 42 43 43 43 46 46 47 47 49 49 53 54 54 55 56 61 63 63 63 64 67 67 70 70 70 71 72 73 73 74 75 76 76 7
7 77 79 80 80 81 83 84 84 86 87 89 90 90 90 91 95 97 97 99 99 100 }
Rendezesi ido: 9 mikromasodperc [us]

Buborekrendezés
=====
Kiindulo allapot => { 45.8212 39.0606 96.5 11.6455 52.4424 77.9242 94.8242 89.4848 35.8848 57.1667 17.1576 8.60303 45.08
48 84.6667 17.0212 11.9152 70.8545 64.8818 5.61212 52.2667 53.4091 36.2273 42.2848 73.2545 45.2333 42.4091 37.8636 67.63
94 48.3394 86.9485 2.76364 90.1273 87.5242 74.7333 36.1909 53.9697 38.7182 91.2061 96.7727 83.9212 71.8879 29.3152 41.78
48 63.4182 36.0242 7.19697 74.6424 42.2424 35.8909 90.1697 2.55455 43.1818 31.4848 64.8939 25.7212 40.3212 14.8879 63.2
69.8636 37.5909 84.8909 3.0697 85.5242 27.2909 87.3909 34.4152 14 55.0606 28.6485 56.4697 56.0061 71.7182 35.6818 97.254
5 42.2091 96.397 77.9273 19.0758 31.2061 73.403 11.8061 42.6727 40.3879 46.597 63.0242 57.4818 42.3333 33.7303 24.397 4.
96061 18.7636 25.8697 64.3424 96.8909 13.1758 90.2 33.7121 77.3485 73.4909 8.34545 }
Vegallapot => { 2.55455 2.76364 3.0697 4.96061 5.61212 7.19697 8.34545 8.60303 11.6455 11.8061 11.9152 13.1758 14
14.8879 17.0212 17.1576 18.7636 19.0758 24.397 25.7212 25.8697 27.2909 28.6485 29.3152 31.2061 31.4848 33.7121 33.7303 3
4.4152 35.6818 35.8848 35.8909 36.0242 36.1909 36.2273 37.5909 37.8636 38.7182 39.0606 40.3212 40.3879 41.7848 42.2091 4
2.2424 42.2848 42.3333 42.4091 42.6727 43.1818 45.0848 45.2333 45.8212 46.597 48.3394 52.2667 52.4424 53.4091 53.9697 55
.0606 56.0061 56.4697 57.1667 57.4818 63.0242 63.2 63.4182 64.3424 64.8818 64.8939 67.6394 69.8636 70.8545 71.7182 71.88
79 73.2545 73.403 73.4909 74.6424 74.7333 77.3485 77.9242 77.9273 83.9212 84.6667 84.8909 85.5242 86.9485 87.3909 87.524
2 89.4848 90.1273 90.1697 90.2 91.2061 94.8242 96.397 96.5 96.7727 96.8909 97.2545 }
Rendezesi ido: 24 mikromasodperc [us]

Beszuro rendezes:
=====
```

```
D:\SortByFunctionTemplate\Debug\SortByFunctionTemplate.exe
Kiindulo állapot => { 45.8212 39.0606 96.5 11.6455 52.4424 77.9242 94.8242 89.4848 35.8848 57.1667 17.1576 8.60303 45.08
48 84.6667 17.0212 11.9152 70.8545 64.8818 5.61212 52.2667 53.4091 36.2273 42.2848 73.2545 45.2333 42.4091 37.8636 67.63
94 48.3394 86.9485 2.76364 90.1273 87.5242 74.7333 36.1909 53.9697 38.7182 91.2061 96.7727 83.9212 71.8879 29.3152 41.78
48 63.4182 36.0242 7.19697 74.6424 42.2424 35.8909 90.1697 2.55455 43.1818 31.4848 64.8939 25.7212 40.3212 14.8879 63.2
69.8636 37.5909 84.8909 3.0697 85.5242 27.2909 87.3909 34.4152 14 55.0606 28.6485 56.4697 56.0061 71.7182 35.6818 97.254
5 42.2091 96.397 77.9273 19.0758 31.2061 73.403 11.8061 42.6727 40.3879 46.597 63.0242 57.4818 42.3333 33.7303 24.397 4.
96061 18.7636 25.8697 64.3424 96.8909 13.1758 90.2 33.7121 77.3485 73.4909 8.34545 }
Vegallapot => { 2.55455 2.76364 3.0697 4.96061 5.61212 7.19697 8.34545 8.60303 11.6455 11.8061 11.9152 13.1758 14.
14.8879 17.0212 17.1576 18.7636 19.0758 24.397 25.7212 25.8697 27.2909 28.6485 29.3152 31.2061 31.4848 33.7121 33.7303 3
4.4152 35.6818 35.8848 35.8909 36.0242 36.1909 36.2273 37.5909 37.8636 38.7182 39.0606 40.3212 40.3879 41.7848 42.2091 4
2.2424 42.2848 42.3333 42.4091 42.6727 43.1818 45.0848 45.2333 45.8212 46.597 48.3394 52.2667 52.4424 53.4091 53.9697 55
.0606 56.0061 56.4697 57.1667 57.4818 63.0242 63.2 63.4182 64.3424 64.8818 64.8939 67.6394 69.8636 70.8545 71.7182 71.88
79 73.2545 73.403 73.4909 74.6424 74.7333 77.3485 77.9242 77.9273 83.9212 84.6667 84.8909 85.5242 86.9485 87.3909 87.524
2 89.4848 90.1273 90.1697 90.2 91.2061 94.8242 96.397 96.5 96.7727 96.8909 97.2545 }
Rendezési ido: 9 mikromasodperc [us]

Kozvetlen kivalaszto rendezes
=====
Kiindulo állapot => { 45.8212 39.0606 96.5 11.6455 52.4424 77.9242 94.8242 89.4848 35.8848 57.1667 17.1576 8.60303 45.08
48 84.6667 17.0212 11.9152 70.8545 64.8818 5.61212 52.2667 53.4091 36.2273 42.2848 73.2545 45.2333 42.4091 37.8636 67.63
94 48.3394 86.9485 2.76364 90.1273 87.5242 74.7333 36.1909 53.9697 38.7182 91.2061 96.7727 83.9212 71.8879 29.3152 41.78
48 63.4182 36.0242 7.19697 74.6424 42.2424 35.8909 90.1697 2.55455 43.1818 31.4848 64.8939 25.7212 40.3212 14.8879 63.2
69.8636 37.5909 84.8909 3.0697 85.5242 27.2909 87.3909 34.4152 14 55.0606 28.6485 56.4697 56.0061 71.7182 35.6818 97.254
5 42.2091 96.397 77.9273 19.0758 31.2061 73.403 11.8061 42.6727 40.3879 46.597 63.0242 57.4818 42.3333 33.7303 24.397 4.
96061 18.7636 25.8697 64.3424 96.8909 13.1758 90.2 33.7121 77.3485 73.4909 8.34545 }
Vegallapot => { 2.55455 2.76364 3.0697 4.96061 5.61212 7.19697 8.34545 8.60303 11.6455 11.8061 11.9152 13.1758 14.
14.8879 17.0212 17.1576 18.7636 19.0758 24.397 25.7212 25.8697 27.2909 28.6485 29.3152 31.2061 31.4848 33.7121 33.7303 3
4.4152 35.6818 35.8848 35.8909 36.0242 36.1909 36.2273 37.5909 37.8636 38.7182 39.0606 40.3212 40.3879 41.7848 42.2091 4
2.2424 42.2848 42.3333 42.4091 42.6727 43.1818 45.0848 45.2333 45.8212 46.597 48.3394 52.2667 52.4424 53.4091 53.9697 55
.0606 56.0061 56.4697 57.1667 57.4818 63.0242 63.2 63.4182 64.3424 64.8818 64.8939 67.6394 69.8636 70.8545 71.7182 71.88
79 73.2545 73.403 73.4909 74.6424 74.7333 77.3485 77.9242 77.9273 83.9212 84.6667 84.8909 85.5242 86.9485 87.3909 87.524
2 89.4848 90.1273 90.1697 90.2 91.2061 94.8242 96.397 96.5 96.7727 96.8909 97.2545 }
Rendezési ido: 18 mikromasodperc [us]

Shell-rendezés
=====
Kiindulo állapot => { 45.8212 39.0606 96.5 11.6455 52.4424 77.9242 94.8242 89.4848 35.8848 57.1667 17.1576 8.60303 45.08
48 84.6667 17.0212 11.9152 70.8545 64.8818 5.61212 52.2667 53.4091 36.2273 42.2848 73.2545 45.2333 42.4091 37.8636 67.63
94 48.3394 86.9485 2.76364 90.1273 87.5242 74.7333 36.1909 53.9697 38.7182 91.2061 96.7727 83.9212 71.8879 29.3152 41.78
48 63.4182 36.0242 7.19697 74.6424 42.2424 35.8909 90.1697 2.55455 43.1818 31.4848 64.8939 25.7212 40.3212 14.8879 63.2
69.8636 37.5909 84.8909 3.0697 85.5242 27.2909 87.3909 34.4152 14 55.0606 28.6485 56.4697 56.0061 71.7182 35.6818 97.254
5 42.2091 96.397 77.9273 19.0758 31.2061 73.403 11.8061 42.6727 40.3879 46.597 63.0242 57.4818 42.3333 33.7303 24.397 4.
96061 18.7636 25.8697 64.3424 96.8909 13.1758 90.2 33.7121 77.3485 73.4909 8.34545 }
Vegallapot => { 2.55455 2.76364 3.0697 4.96061 5.61212 7.19697 8.34545 8.60303 11.6455 11.8061 11.9152 13.1758 14.
14.8879 17.0212 17.1576 18.7636 19.0758 24.397 25.7212 25.8697 27.2909 28.6485 29.3152 31.2061 31.4848 33.7121 33.7303 3
4.4152 35.6818 35.8848 35.8909 36.0242 36.1909 36.2273 37.5909 37.8636 38.7182 39.0606 40.3212 40.3879 41.7848 42.2091 4
2.2424 42.2848 42.3333 42.4091 42.6727 43.1818 45.0848 45.2333 45.8212 46.597 48.3394 52.2667 52.4424 53.4091 53.9697 55
.0606 56.0061 56.4697 57.1667 57.4818 63.0242 63.2 63.4182 64.3424 64.8818 64.8939 67.6394 69.8636 70.8545 71.7182 71.88
79 73.2545 73.403 73.4909 74.6424 74.7333 77.3485 77.9242 77.9273 83.9212 84.6667 84.8909 85.5242 86.9485 87.3909 87.524
2 89.4848 90.1273 90.1697 90.2 91.2061 94.8242 96.397 96.5 96.7727 96.8909 97.2545 }
Rendezési ido: 9 mikromasodperc [us]

Buborekrendezés
=====
Kiindulo állapot => { v u p w e u a i h c w l m m o i b i c p f i w q l u d h l x k m p n f c b c f o q x v c y l e m g
c t p v f u s u d v v f d h w b w f b s z m e z x f r t g h l x c x v d t l c a t t u z l a t t a z }
Vegallapot => { a a a a b b b b c c c c c c c d d d d e e e f f f f f f g h h h h i i i k l l l l l l l l m m
m m n o o p p p p q q r s s t t t t t t t t t t u u u u u u v v v v v v w w w w w x x x x y z z z z }
Rendezési ido: 22 mikromasodperc [us]

D:\SortByFunctionTemplate\Debug\SortByFunctionTemplate.exe
Beszuro rendezes:
=====
Kiindulo állapot => { v u p w e u a i h c w l m m o i b i c p f i w q l u d h l x k m p n f c b c f o q x v c y l e m g
c t p v f u s u d v v f d h w b w f b s z m e z x f r t g h l x c x v d t l c a t t u z l a t t a z }
Vegallapot => { a a a a b b b b c c c c c c c d d d d e e e f f f f f f g h h h h i i i k l l l l l l l l m m
m m n o o p p p p q q r s s t t t t t t t t t t u u u u u u v v v v v v w w w w w x x x x y z z z z }
Rendezési ido: 9 mikromasodperc [us]

Kozvetlen kivalaszto rendezes
=====
Kiindulo állapot => { v u p w e u a i h c w l m m o i b i c p f i w q l u d h l x k m p n f c b c f o q x v c y l e m g
c t p v f u s u d v v f d h w b w f b s z m e z x f r t g h l x c x v d t l c a t t u z l a t t a z }
Vegallapot => { a a a a b b b b c c c c c c c d d d d e e e f f f f f f g h h h h i i i k l l l l l l l l m m
m m n o o p p p p q q r s s t t t t t t t t t t u u u u u u v v v v v v w w w w w x x x x y z z z z }
Rendezési ido: 13 mikromasodperc [us]

Shell-rendezés
=====
Kiindulo állapot => { v u p w e u a i h c w l m m o i b i c p f i w q l u d h l x k m p n f c b c f o q x v c y l e m g
c t p v f u s u d v v f d h w b w f b s z m e z x f r t g h l x c x v d t l c a t t u z l a t t a z }
Vegallapot => { a a a a b b b b c c c c c c c d d d d e e e f f f f f f g h h h h i i i k l l l l l l l l m m
m m n o o p p p p q q r s s t t t t t t t t t t u u u u u u v v v v v v w w w w w x x x x y z z z z }
Rendezési ido: 7 mikromasodperc [us]
```

A feladathoz kapcsolódó magyarázatok

Az előző mintapéldák alapján kijelenthető, hogy a függvények kiválóan segítenek abban, hogy az alkalmazások biztonságosabbá és könnyebben karbantarthatóbbá váljanak. Azonban egyes esetekben olyan akadályokba ütközhetünk, mint a paraméterhez történő típus rendelés. A fentebb ismertet gyakorlati példán bemutatott függvénynevek túlterhelése egy olyan technika, amely segítségével a típusmegkötésektől bizonyos mértékig függetleníteni tudjuk a függvényeket. Ellenben az ilyen jellegű technikával csak véges számú típusal dolgozhatunk, ráadásul a fejlesztői szempontot és a kódkarbantarthatóságot figyelembe véve a megismételt forráskódok nagyban nehezítik a fejlesztők számára a kódok javítását. Így jogosan felmerülhet az olyan jellegű igény, ahol a függvényt csak egyszer kelljen megírni, azonban az tetszőleges mennyiségben felhasználható legyen, tetszőleges típusú paraméterekkel. Az ilyen igények kielégítésére a C++ a generikus programozás eszközeit nyújtja a fejlesztő számára. Az ilyen jellegű fejlesztés a programkód olyan jellegű megvalósítását teszi lehetővé, amelyek nem függenek a típusoktól, illetve az ilyen általánosított típusok nagyban növelik a kód újrafelhasználhatóságát. A C++ nyelv a generikus programozás technikát kínálja a sablonok (**template**) segítségével, amelyek fordítási időben kerülnek megvalósításra.

Az osztály deklarációját a (**SortByFunctionTemplate.h**) állományban adjuk meg. Az osztály egy magán (**private**) láthatóságú konstans adattaggal rendelkezik, amely a jelenlegi feladat esetében, eltárolja a rendezni kívánt tömbök méretét. Konstansok használata különböző tömb méretek meghatározásánál bevett szokás, mivel ha a méretérték valamilyen okból kifolyólag változtatásra kerül, elég csak az állandó értékét átírni és nem kell a különböző műveletek (tömb feltöltés, rendezés vagy épp megjelenítés) végrehajtását segítő részeket nyomon követni és módosítani. Egy osztályon belül konstans megadása az alábbi formában történhet:

```
static const int SIZE = 100;
```

A **static** kulcsszó jelöli azt, hogy ezt a konstanst megosztva használják az osztályok példányai, így az közvetlenül az osztályhoz tartozik. Fontos szabály, hogy amíg a statikus osztály adattagokat az osztályon kívül inicializáljuk, addig az állandókat az osztályon belül is el láthatjuk kezdőértékekkel. A **const** kulcsszó jelöli, hogy ebben az esetben egy állandóról van szó, majd következik a konstans típusának megadása, ezt követi a konstans neve, amelyet a kód könnyebb olvashatósága végett (jobban elkülönülnek a környezettől) nagybetűvel került megfogalmazásra, végül a konstans kezdőértékkel történő ellátása következik, ebben az esetben a 100 értékkel. Az adattag megfogalmazását követően létrehozásra kerül egy explicit, alapértelmezett, paraméter nélküli konstruktor. A

függvénysablonok deklarációját a **template** kulcsszóval kezdjük, majd < > jelek között adjuk meg a sablon paramétereit. Ezek a paraméterek leginkább általánosított típusnevek, ellenben változók is szerepelhetnek közöttük. Az általánosított típusnevek előtt a **class** vagy a **typename** kulcsszavakat használjuk. Így tehát egy függvény deklaráció a tömb feltöltés esetében a következő módon néz ki:

```
template <typename T> void ArrayFill(T *arrayToFill);
```

A *BubbleSort(T *arrayToBubbleSort)*, *SelectionSort(T *arrayToSelection)*, *InsertionSort(T *arrayToInsertion)*, *ShellSort(T *arrayToShellSort)*, *PrintArray(const T *arrayToCopy)* visszatérési típusaik **void**, amely jelentése szerint a függvények nem térnek vissza különböző típusú értékkel (természetesen ezeket a típusokat is lehet általánosítani). Az *ArrayCopy(const T *arrayToCopy, T *copiedArray)* tagfüggvény **inline** módon került megvalósításra, a függvény felelőssége az eredeti tömb egy másolatának létrehozása. A deklaráció végén megfogalmazásra került a(z) *~SortByFunctionTemplate()* nevezetű, üres paraméterlistával rendelkező destruktorktor, amely az erőforrások felszabadításáért lesz felelős.

A (**SortByFunctionOverloading.cpp**) állomány tartalmazza az osztály megvalósítását, tagfüggvényeinek implementálását. A tagfüggvények neve előtt a :: hatókör operátor kerül felhasználásra azon célból, hogy egyértelműsítésre kerüljön melyik osztály tagfüggvényét szeretnénk kifejteni. A *template <typename T> void SortByFunctionTemplate::ArrayFill(T *arrayToFill)* függvény feladata a paraméterül kapott általános típusú tömb véletlen számokkal történő feltöltése egész típus esetén 1 és 100 között, lebegőpontos típus esetén 1,0 és 100,0 között, karakter típus esetén véletlenbetűkkel történő feltöltése a és z között. A karakterek meghatározásánál az ASCII kódokat alkalmaztuk, ahol is véletlenszerűen kiválasztásra került egy számérték a betűket tartalmazó ASCII szám halmazból, majd azt típuskényszerítéssel karakterré alakítottuk. Az osztály implementációja során adott rendezési algoritmust végző tagfüggvények felelőssége a rájuk szabott rendezési műveletek implementálása. Ezek a tagfüggvények minden esetben paraméterül kapják a még rendezetlen tömböt, majd megjelenítik azt a konzolon, mint kiinduló állapot, majd közvetlenül a rendezési algoritmusok előtt elindítjuk az időmérést, amelyet a rendezés végeztével leállítunk. Minden olyan esetben, ahol a függvény feladata egy adott rendezési algoritmus elvégzése általános típusú tömbön, a függvénysablonok kerültek alkalmazásra. Ezt követően megjelenítjük – típustól függetlenül – a már rendezett tömb – a *PrintArray()* függvény segítségével – értékeket, valamint azt, hogy mennyi időbe került a rendezés. Az idő méréséhez egy **chrono** nevezetű **header** fájlt használunk, továbbá szükséges az **std** névtér behívása is, ahol az egyes elemek implementálása megtalálható. Az aktuális időpont lekérését a következő utasítással végezzük, amelyet a start lokális változóban tároljuk:

```
chrono::steady_clock::time_point start = chrono::steady_clock::now();
```

A rendezési algoritmusok lefutását követően ismét lekérjük az aktuális időt, amelyet az **end** lokális változó kap értékül:

```
chrono::steady_clock::time_point end = chrono::steady_clock::now();
```

A rendezéshez szükséges idő megjelenítését a következő utasítással végezzük, ahol mikromásodpercben kiszámoljuk az eltelt időtartamot (**end - start**):

```
chrono::duration_cast<chrono::microseconds>(end - start).count()
```

A (**main.cpp**) állomány tartalmazza a **main()** függvényt, a program belépési pontját. A főfüggvényben történő dinamikus példányosítása után hívjuk meg a szükséges függvényeket. A példányosítás során átadásra kerülnek azon tömb(ök), amelyeken a rendezési műveletek elvégezzük, majd gondoskodunk a lefoglalt erőforrások felszabadításáról a destruktor hívásának segítségével. Ahhoz, hogy véletlen szám generálás során ne ugyanazokat a számokat kapjuk, szükséges a processzor idő nullázására, amelyet a következő utasítással végzünk:

```
srand((unsigned)time(0));
```

Az objektumpéldányon keresztül történő függvény hívás során a fordító típus behelyettesítést végez. Ez azt jelenti, hogy attól függően, milyen típusú tömb kerül átadásra az annak megfelelő típus kerül behelyettesítésre az általános típus helyére. Röviden a fordító a hívás során az adott típussal/típusokkal elkészíti a függvény egy példányát, így minden egyes függvénysablon adott típusokkal specializált változata. Így az általánosított tömbfeltöltésért felelős függvény egész számtípussal specializált változata a következő módon néz ki:

```
objectOfSortByFunctionTemplate->ArrayFill<int>(integerNumbers);
```

Gyakorló példa

Írjunk egy Gyorsrendezést reprezentáló osztályt (*QuickSortByTemplate*), mely képes egy tetszőleges típusú 10 elemű tömb rendezésére.

Az osztály tagváltozói a következők:

- A rendezendő tömb (*array), mely tetszőleges típusú és 10 elemből áll.
- A rendezendő tömb mérete (size), mely egy egész típusú számérték.

Ügyeljünk az egységbezárás elvére, tehát az adattagok private láthatóságú szinttel rendelkezzenek!

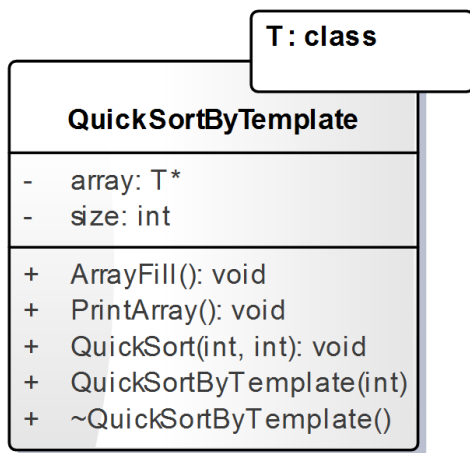
Valósítsuk meg az osztály metódusait:

- Írjuk meg az osztály konstruktorát, melyben az osztály T típusú tömbjét és méretét inicializáljuk.

- Írjunk egy *ArrayFill()* függvényt, amely gondoskodik a tömb véletlenértékekkel történő feltöltéséről.
- Írjunk egy *QuickSort(int leftBound, int rightBound)* függvényt, amely implementálja a gyorsrendezés algoritmusát. A függvény legyen képes megjeleníteni, hogy hányszor került (rekurzívan) meghívásra egy adott rendezés során.
- Írjunk egy egyszerű *PrintArray()* függvényt, amely megjeleníti a T típusú tömb elemeit.
- Ne feledkezzünk meg az objektumpéldány által lefoglalt erőforrás felszabadításáról, a destruktork megírásáról sem.
- Írjunk egy egyszerű *main* függvényt, ahol létrehozuk az objektumpéldányt, majd az objektumon keresztül hívjuk a szükséges függvényeket.

Az osztály implementálását követően dinamikus módszerrel (a memória dinamikus területén [**heap**]) készítünk egy objektumpéldányt. Az objektumpéldány segítségével a feladat leírásnak megfelelően hívjuk meg az osztály egyes tagfüggvényeit. Amennyiben nincs tovább szükség az objektum példányra, gondoskodjunk annak megszüntetéséről, elkerülve ezzel a memóriaszivárgást.

Osztály diagram



62. ábra: Osztálydiagram

Megoldás

QuickSortByTemplate.h

```
#ifndef QUICKSORTBYTEMPLATE_H
#define QUICKSORTBYTEMPLATE_H

template <class T> class QuickSortByTemplate
{
private:
    T *array;
    int size;

public:
    QuickSortByTemplate(int size);
    void ArrayFill();
    void QuickSort(int leftBound, int rightBound);
    void PrintArray();
    ~QuickSortByTemplate() { delete[] array; }
};

#endif
```

QuickSortByTemplate.cpp

```
#include <iostream>
#include <ctime>
#include "QuickSortByTemplate.h"

template <class T> QuickSortByTemplate<T>::QuickSortByTemplate(int
size)
{
    array = new T[size];
    this->size = size;
}

template <class T> void QuickSortByTemplate<T>::ArrayFill()
{
    srand((unsigned)time(0));
    for (int i = 0; i < size; ++i)
        typeid(T) == typeid(char) ? array[i] = (T)(rand() % 26) +
97 : array[i] = 1 + (T)rand() / (T)(RAND_MAX / (100 - 1));
}

template <class T> void QuickSortByTemplate<T>::QuickSort(int
leftBound, int rightBound)
{
    static unsigned int calls = 0;
    if (rightBound <= leftBound)
        return;

    std::cout << " Gyorsrendezesert felelos fuggveny hivasa #: " <<
++calls << std::endl;
    PrintArray();
    int pivotIndex = leftBound + (rand() % (rightBound -
leftBound));
    T pivot = array[pivotIndex];

    std::cout << " Vezerelem: " << "[" << pivotIndex << "]" " <<
pivot << std::endl;

    int leftPosition = leftBound;
    int rightPosition = rightBound;

    while (leftPosition < rightPosition)
    {
        while (leftPosition < rightPosition && array[leftPosition]
< pivot)
            ++leftPosition;

        while (rightPosition > leftPosition &&
array[rightPosition] > pivot)
            --rightPosition;

        if (leftPosition < rightPosition)
        {
            if (array[leftPosition] != array[rightPosition])
            {
                std::swap(array[leftPosition],
array[rightPosition]);
                std::cout << " Csere ~> [" << rightPosition <<
"] " << array[rightPosition] << " & [" <<
leftPosition << "]" << array[leftPosition] <<
std::endl;
            }
        }
    }
}
```

```

        }
        else
            ++leftPosition;
    }
}
std::cout << " A tomb allapota a rekurziv hivas elott: " <<
std::endl;
PrintArray();
QuickSort(leftBound, leftPosition - 1);
QuickSort(rightPosition + 1, rightBound);
}
template <class T> void QuickSortByTemplate<T>::PrintArray()
{
    std::cout << " { ";
    for (int i = 0; i < size; ++i)
        std::cout << array[i] << " ";
    std::cout << " } " << std::endl;
}

```

main.cpp

```
#include <iostream>
#include <conio.h>
#include "QuickSortByTemplate.cpp"

int main()
{
    QuickSortByTemplate<int> *objectOfQuickSortByTemplate = new
    QuickSortByTemplate<int>(10);
    objectOfQuickSortByTemplate->ArrayFill();
    objectOfQuickSortByTemplate->QuickSort(0, 9);
    delete objectOfQuickSortByTemplate;

    _getch();
    return 0;
}
```

Program futásának eredménye:

```
D:\Dunaújvárosi Egyetem\Publikációk\Köny...  —  □  ×

Gyorsrendezesert felelos fuggveny hivasa #: 1
{ 36 35 86 59 90 10 64 43 5 7 }
Vezerelem: [5] 10
Csere ~> [9] 36 & [0] 7
Csere ~> [8] 35 & [1] 5
Csere ~> [5] 86 & [2] 10
A tomb allapota a rekurziv hivas elott:
{ 7 5 10 59 90 86 64 43 35 36 }
Gyorsrendezesert felelos fuggveny hivasa #: 2
{ 7 5 10 59 90 86 64 43 35 36 }
Vezerelem: [0] 7
Csere ~> [1] 7 & [0] 5
A tomb allapota a rekurziv hivas elott:
{ 5 7 10 59 90 86 64 43 35 36 }
Gyorsrendezesert felelos fuggveny hivasa #: 3
{ 5 7 10 59 90 86 64 43 35 36 }
Vezerelem: [7] 43
Csere ~> [9] 59 & [3] 36
Csere ~> [8] 90 & [4] 35
Csere ~> [7] 86 & [5] 43
A tomb allapota a rekurziv hivas elott:
{ 5 7 10 36 35 43 64 86 90 59 }
Gyorsrendezesert felelos fuggveny hivasa #: 4
{ 5 7 10 36 35 43 64 86 90 59 }
Vezerelem: [3] 36
Csere ~> [4] 36 & [3] 35
A tomb allapota a rekurziv hivas elott:
{ 5 7 10 35 36 43 64 86 90 59 }
Gyorsrendezesert felelos fuggveny hivasa #: 5
{ 5 7 10 35 36 43 64 86 90 59 }
Vezerelem: [6] 64
Csere ~> [9] 64 & [6] 59
Csere ~> [9] 86 & [7] 64
A tomb allapota a rekurziv hivas elott:
{ 5 7 10 35 36 43 59 64 90 86 }
Gyorsrendezesert felelos fuggveny hivasa #: 6
{ 5 7 10 35 36 43 59 64 90 86 }
Vezerelem: [8] 90
Csere ~> [9] 90 & [8] 86
A tomb allapota a rekurziv hivas elott:
{ 5 7 10 35 36 43 59 64 86 90 }
```

```
D:\Dunaújvárosi Egyetem\Publikációk\Könyvek\QuickSortByTemplate\Debug\QuickSortByTemplate.e...
Gyorsrendezesert felelos fuggveny hivasa #: 1
{ 35.3939 47.3121 52.7303 17.4455 15.8636 91.7303 56.8455 3.57576 21.9303 2.71212 }
Vezerelem: [4] 15.8636
Csere ~> [9] 35.3939 & [0] 2.71212
Csere ~> [7] 47.3121 & [1] 3.57576
Csere ~> [4] 52.7303 & [2] 15.8636
A tomb allapota a rekurziv hivas elott:
{ 2.71212 3.57576 15.8636 17.4455 52.7303 91.7303 56.8455 47.3121 21.9303 35.3939 }
Gyorsrendezesert felelos fuggveny hivasa #: 2
{ 2.71212 3.57576 15.8636 17.4455 52.7303 91.7303 56.8455 47.3121 21.9303 35.3939 }
Vezerelem: [0] 2.71212
A tomb allapota a rekurziv hivas elott:
{ 2.71212 3.57576 15.8636 17.4455 52.7303 91.7303 56.8455 47.3121 21.9303 35.3939 }
Gyorsrendezesert felelos fuggveny hivasa #: 3
{ 2.71212 3.57576 15.8636 17.4455 52.7303 91.7303 56.8455 47.3121 21.9303 35.3939 }
Vezerelem: [8] 21.9303
Csere ~> [8] 52.7303 & [4] 21.9303
A tomb allapota a rekurziv hivas elott:
{ 2.71212 3.57576 15.8636 17.4455 21.9303 91.7303 56.8455 47.3121 52.7303 35.3939 }
Gyorsrendezesert felelos fuggveny hivasa #: 4
{ 2.71212 3.57576 15.8636 17.4455 21.9303 91.7303 56.8455 47.3121 52.7303 35.3939 }
Vezerelem: [7] 47.3121
Csere ~> [9] 91.7303 & [5] 35.3939
Csere ~> [7] 56.8455 & [6] 47.3121
A tomb allapota a rekurziv hivas elott:
{ 2.71212 3.57576 15.8636 17.4455 21.9303 35.3939 47.3121 56.8455 52.7303 91.7303 }
Gyorsrendezesert felelos fuggveny hivasa #: 5
{ 2.71212 3.57576 15.8636 17.4455 21.9303 35.3939 47.3121 56.8455 52.7303 91.7303 }
Vezerelem: [8] 52.7303
Csere ~> [8] 56.8455 & [7] 52.7303
A tomb allapota a rekurziv hivas elott:
{ 2.71212 3.57576 15.8636 17.4455 21.9303 35.3939 47.3121 52.7303 56.8455 91.7303 }
Gyorsrendezesert felelos fuggveny hivasa #: 6
{ 2.71212 3.57576 15.8636 17.4455 21.9303 35.3939 47.3121 52.7303 56.8455 91.7303 }
Vezerelem: [8] 56.8455
A tomb allapota a rekurziv hivas elott:
{ 2.71212 3.57576 15.8636 17.4455 21.9303 35.3939 47.3121 52.7303 56.8455 91.7303 }
```

```
D:\Dunaújvárosi Egyetem\Publikációk\Kö...  —  □  ×

Gyorsrendezesert felelos fuggveny hivasa #: 1
{ u c k k k d k n b b }
Vezerelem: [5] d
Csere ~> [9] u & [0] b
Csere ~> [8] k & [2] b
Csere ~> [5] k & [3] d
A tomb allapota a rekurziv hivas elott:
{ b c b d k k k n k u }
Gyorsrendezesert felelos fuggveny hivasa #: 2
{ b c b d k k k n k u }
Vezerelem: [1] c
Csere ~> [2] c & [1] b
A tomb allapota a rekurziv hivas elott:
{ b b c d k k k n k u }
Gyorsrendezesert felelos fuggveny hivasa #: 3
{ b b c d k k k n k u }
Vezerelem: [0] b
A tomb allapota a rekurziv hivas elott:
{ b b c d k k k n k u }
Gyorsrendezesert felelos fuggveny hivasa #: 4
{ b b c d k k k n k u }
Vezerelem: [8] k
Csere ~> [8] n & [7] k
A tomb allapota a rekurziv hivas elott:
{ b b c d k k k k n u }
Gyorsrendezesert felelos fuggveny hivasa #: 5
{ b b c d k k k k n u }
Vezerelem: [4] k
A tomb allapota a rekurziv hivas elott:
{ b b c d k k k k n u }
Gyorsrendezesert felelos fuggveny hivasa #: 6
{ b b c d k k k k n u }
Vezerelem: [4] k
A tomb allapota a rekurziv hivas elott:
{ b b c d k k k k n u }
Gyorsrendezesert felelos fuggveny hivasa #: 7
{ b b c d k k k k n u }
Vezerelem: [8] n
A tomb allapota a rekurziv hivas elott:
{ b b c d k k k k n u }
```

Felhasznált irodalom

- Andrei Alexandrescu, Herb Sutter: *C++ kódolási szabályok*. Kiskapu Kft. 2005.
- B. Stroustrup: *A C++ programozási nyelv I II. kötet*. Kiskapu Kiadó, 2001.
- Benedek Zoltán - Levendovszky Tihamér: *Szoftverfejlesztés C++ nyelven*. SZAK kiadó, 2007.
- Benkő Tiborné - Poppe András: *Objektumorientált C++ (Együtt könnyebb a programozás)*, ComputerBooks, 2010.
- Craig Larman: *Applying UML and patterns*, Second Edition, Prentice-Hall, Inc., USA, 2002.
- Fóthi Ákos: *Bevezetés a programozáshoz*. ELTE Eötvös Kiadó, 2005.
- Hans-Erik Eriksson, Magnus Penker: *Business Modeling with UML*, John Wiley & Sons, Inc., New York, 2000.
- Ian Sommerville: *Software Engineering*. Seventh Edition, Pearson Education, Inc., USA, 2004.
- Kent Beck: *Implementációs minták*. Panem kiadó, 2008.
- Kondorosi Károly - László Zoltán - Szirmay-Kalos László: *Objektumorientált szoftverfejlesztés*. ComputerBooks, 2004.
- Mark Priestley: *Practical Object-Oriented Design with UML*. McGraw-Hill Publishing Company, Great Britain, 2000.
- Roger S. Pressman: *Software Engineering*. Fifth Edition, McGraw-Hill Book Company, USA, 2001.
- Sike Sándor – Varga László: *Szoftvertechnológia és UML*. ELTE, 2007
- Stephen R. Schach: *Object-Oriented and Classical Software Engineering*. Eighth Edition McGraw-Hill, New York, 2011.