SEMINAR 1

Contents

1. OBJECTIVES

- Solve a problem using modular programming in C.
- Discuss memory management in C and implement various data structures (static and dynamic).

2. PROBLEM STATEMENT

The **Stargate Program** needs an application to help keeping track of the planets and alien races that have been discovered so far. Each **Planet** has a unique symbol composed of exactly 7 symbols, a name, the Solar System it belongs to and the distance to Earth (measured in thousands light-years). The Stargate team needs this application to help them in the following ways:

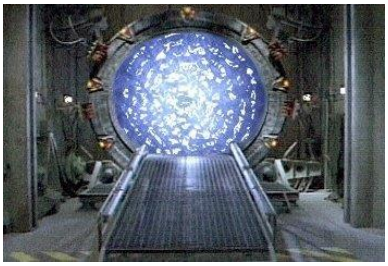


Image source: https://www.pinterest.com/natbackstrom/sg-1/

a. The application must allow adding and deleting planets.
b. The application should offer the possibility to display all the planets whose symbols contain a given combination as a substring (if the combination is empty, all the planets should be considered).
c. The application should allow displaying all the planets in a given Solar System (if the Solar System is empty, all planets should be considered), whose distances to Earth are less than a given value, sorted ascending by distance.
d. The application must provide the option to undo and redo the last change.

3. STATIC ALLOCATION

- There is no need for explicit memory allocation, this happens automatically, when variables are declared.
- All fields of the *Planet* structure are statically allocated.

```
typedef struct
{
	char symbols[8];
	char name[50];
	char solarSystem[50];
	double distanceToEarth;
} Planet;
```

- The vector of planets is statically allocated.

```
typedef struct
{
	Planet planets[100];
	int length;
} PlanetRepo;
```

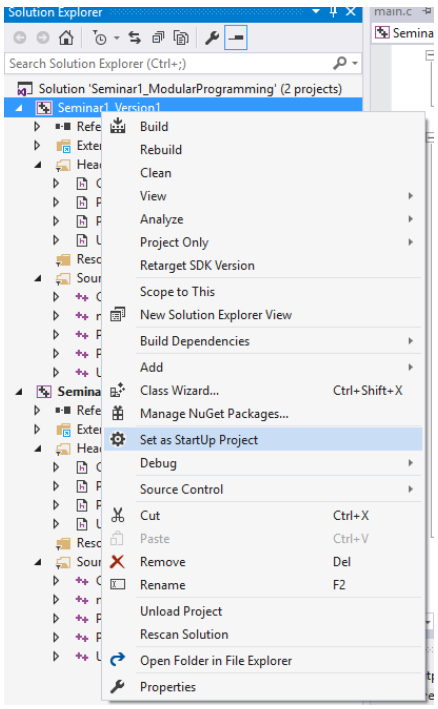
- All objects in the application are statically allocated.

```
int main()
{
	PlanetRepo repo = createRepo();
	Controller ctrl = createController(&repo);
	UI ui = createUI(&ctrl);
	// …

	return 0;
}
```

Please see Seminar1_ModularProgramming.zip → project Seminar1_Version1.

Obs.: In a Visual Studio solution, one can have several projects. If you want to run a certain project, right click on the project and choose “Set as StartUp Project”.



4. DYNAMIC ALLOCATION

- Memory is allocated when we need it.
- We are **responsible** with de-allocating it, once we no longer need it.
- Necessary functions: **malloc**, **free** (header *stdlib.h*).
- The objects we are working with will have to provide functions for *creation and destruction*.
- E.g. Creating and destroying a Planet:

```
Planet* createPlanet(char* symbols, char* name, char* solarSystem, double
distanceToEarth)
{
	Planet* p = (Planet*)malloc(sizeof(Planet));
	p->symbols = (char*)malloc(strlen(symbols) + 1);
	strcpy(p->symbols, symbols);
	p->name = (char*)malloc(strlen(name) + 1);
	strcpy(p->name, name);
	p->solarSystem = (char*)malloc(strlen(solarSystem) + 1);
	strcpy(p->solarSystem, solarSystem);
	p->distanceToEarth = distanceToEarth;

	return p;
}

void destroyPlanet(Planet* p)
{
	// free the memory which was allocated for the component fields
	free(p->symbols);
	free(p->name);
	free(p->solarSystem);

	// free the memory which was allocated for the planet structure
	free(p);
}
```

- The vector of planets will contain pointers, not objects.

```
typedef struct
{
	Planet* planets[100];
	int length;
} PlanetRepo;
```

- All objects in the application are dynamically allocated. Then they must also be destroyed.

```
int main()
{
	PlanetRepo* repo = createRepo();
	Controller* ctrl = createController(repo);
	UI* ui = createUI(ctrl);
	// …

	destroyUI(ui);
	return 0;
}
```

Please see Seminar1_ModularProgramming.zip → project Seminar1_Version2.