КУРСОВИ ПРОЕКТИ ПО СТРУКТУРИ ОТ ДАННИ

Факултет по математика и информатика на СУ „Св. Климент Охридски", Специалност „Информатика“, 2 курс

Зимен семестър 2015/2016

Обща информация за проектите

Проектите се оценяват по редица от критерии, всеки от които носи точки. Общата сума от точките надвишава необходимия брой за оценка отличен. Тъй като курсът се фокусира върху алгоритмите и структурите данните и тяхната реализация на езика C++, **най-важното изискване за проектите е те да включват подходящи реализации на различни алгоритми и структури от данни**. Проект без такива реализации се оценява с нула точки. Други важни критерии за оценка на проектите са:

- **Дали решението работи коректно**, съгласно спецификацията. Решение, което не работи и/или не се компилира носи минимален брой (или нула) точки.
- **Дали решението отговаря на заданието на проекта**.
- **Дали решението използва подходящи алгоритми и структури от данни**. Това е най-важното изискване към проектите. Тъй като курсът се фокусира върху алгоритмите и структурите от данни и тяхната реализация, проекти в които няма реализация на подходящи алгоритми и структури от данни се оценяват с нула точки. Включването им в проектите трябва да бъде обосновано. По време на защитата се очаква да можете да отговорите на различни въпроси, като например: (1) кои алгоритми сте реализирали, (2) защо сте избрали именно тях, (3) как сте ги реализирали, (4) каква е тяхната сложност и др. Ако за някои части на проекта не можете да реализирате подходящ алгоритъм, можете да използвате и brute force решения но те носят минимален брой (или нула) точки.
- **Дали решението е било добре тествано**. Проверява се какви тестове са били проведени за приложението (напр. unit test-ове). Очаква се по време на защитата да можете да посочите как сте тествали приложението, за да проверите дали то работи коректно и как се държи в различни ситуации.
- **Оформление на решението**. Проверява се дали кодът е добре оформен, дали е спазена конвенция за именуване на променливите, дали е добре коментиран и т.н.

- **Дали проектът е документиран**. Проверява се дали кодът е добре коментиран и дали въз основа на коментарите автоматично се генерира техническа документация (напр. с Doxygen). Този критерий не е задължителен и носи бонус точки.

Оценката на всеки от проектите се формира от онази негова част, която е била самостоятелно разработена от вас. Допустимо е да използвате код написан от някой друг (напр. готова библиотека или помощ от ваш приятел/колега), но (1) той не носи точки към проекта и (2) това трябва да бъде ясно обявено при предаването и защитата на проекта, като посочите коя част от проекта сте разработили самостоятелно. Това означава, че:

1. Използваният наготово код трябва да се маркира ясно, като поставите коментари на подходящи места.
2. По време на защитата трябва да посочите кои части сте разработили самостоятелно и кои са взети от други източници.

Когато в даден проект се използва чужд код се оценяват и (1) способността ви за внедряване на този код във вашето решение (напр. в случаите, когато се използва външна библиотека) и (2) дали добре разбирате какво прави чуждият код.

1: Memory Manager

В рамките на този проект трябва да се разработи собствен memory allocator.

За целта трябва да реализирате следните функции:

- `MyMalloc`, която получава като вход брой байтове, които трябва да се заделят и връща указател към заделения от нея блок. При неуспех функцията да връща NULL;
- `MyFree`, която освобождава паметта заделена с `MyMalloc`.

Сами можете да изберете типа на функциите и на техните параметри, например:

```
char* MyMalloc(size_t Size);
void MyFree(char* pBlock);
```

Също така сами трябва да изберете какво да бъде тяхното поведение при възникване на грешка (например ако `MyAlloc` не може да задели памет или когато на `MyFree` бъде подаден адрес, който не е бил заделен с `MyAlloc` и т.н.).

Началният адрес на блока, който връща `MyMalloc` трябва да бъде подравнен на 8 байта.

Можете сами да изберете какъв да бъде алгоритъмът, който да реализирате. Той трябва да бъде такъв, че да позволява операциите да протичат бързо и да се намали фрагментацията.

При реализацията на двете функции не е необходимо да използвате примитиви на операционната система. За работата на двете функции е достатъчно да заделите и поддържате достатъчно голям обем памет, от който да заделяте и в който да освобождавате необходимите на потребителя блокове.

Ето пример за това как трябва да може да работят двете функции:

```
int *pArr1 = (int*) MyMalloc(100 * sizeof(int));
int *pArr2 = (int*) MyMalloc(100 * sizeof(int));
pArr1[0] = pArr2[0] = 10;
MyFree(pArr1);
MyFree(pArr2);
```

При работата върху проекта, полезни могат да ви бъдат следните външни източници:

- Data alignment: Straighten up and fly right. http://www.ibm.com/developerworks/library/pa-dalign/
- Memory, Part 2: Implementing a Memory Allocator. https://github.com/angrave/SystemProgramming/wiki/Memory,-Part-2%3A-Implementing-a-Memory-Allocator
- Dynamic Memory Allocation: Basic Concepts. http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/17-allocation-basic.pptx
- Dynamic Memory Allocation: Advanced Concepts. http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/18-allocation-advanced.pptx

2: Key-Value Store

В рамките на този проект трябва да разработите собствен Key-Value Store.

За целта трябва да реализирате следните функции:

- `Store`, която получава указател към блок в паметта и неговият размер. След това функцията записва съдържанието на блока в някакъв вид хранилище и връща уникален ключ, с който той може по-късно да бъде извлечен;
- `Load`, която получава ключ и връща съдържанието на блока, който съответства на този ключ;
- `Erase`, която получава ключ и изтрива съответния му блок.

Вашите данни трябва да могат да преживяват рестартиране на системата. Например можете да ги записвате в общ файл или да поддържате различен файл за всеки блок.

Сами можете да изберете типа на функциите и на техните параметри, а също и какъв да бъде типът на ключовете. Ако те са символни низове, функциите може да изглеждат например така:

- `char* Store(const char* pBlock, size_t Size);`
- `bool Load(const char* pKey, char*& pBlock, size_t& Size);`
- `bool Erase(const char* pKey);`

Също така сами трябва да изберете какво да бъде тяхното поведение при възникване на грешка (например ако на `Lookup` бъде подаден невалиден ключ, ако `Store` не може да съхрани блока и т.н.).

Можете сами да изберете какъв да бъде алгоритъмът, който да реализирате. Той трябва да бъде такъв, че да позволява операциите да протичат бързо и да се намали рискът от загуба на информация.

Възможно е да реализирате съхранението в паметта, но трябва да гарантирате, че при внезапно спиране на тока, данните няма да бъдат повредени и да има минимални загуби. Разбираемо е, че ако токът спре по време на запис, данните от текущата операция може да не бъдат съхранени. Същевременно, незавършилата операция не трябва да повреждa хранилището.