ПЛАВЕН ПРЕХОД ОТ ЗАДАЧИ КЪМ ПРОЕКТИ В УВОДНИТЕ КУРСОВЕ ПО ПРОГРАМИРАНЕ

Павел Азълов

Резюме. Как да изградим уводните си курсове по програмиране така, че те да са достъпни за настоящите студенти, разбира се без да се налага да се правят компромиси в съдържанието и в изискванията към тях? Това е централният въпрос в статията, на който се прави опит за отговор чрез *плавен преход от задачи към проекти*. Идеята на подхода се състои в прилагането на *проблемно-ориентирано* обучение в курса Програмиране 1 и плавното му продължение в *проектно-ориентирано* обучение в Програмиране 2. Направен е анализ на трудностите, които студентите имат в тези курсове, като се посочват и препоръки за преодоляването им. Предлага се всяко множество от взаимно-свързани задачи да се структурира в редица от задачи, при което всяка задача може да се декомпозира до предшествуващи я задачи от същата редица. По аналогия със задачите, проектите също се формулират и предлагат като редици от проекти, при която част от компонетите (функции, класове) от един проект се използват при проектирането и реализацията на следвашите в редицата проекти. Плавният преход от задачи към проекти се постига чрез декомпозирането на проектите от вторият курс по програмиране до задачи, някои от които следва да се разглеждат още в първия курс.

Keywords: Computer Science Education, Computer Science I, Computer Science II, Problem-based learning, Project-based Learning, Sequences of problems, Sequences of projects

1. Въведение

В тази статия е представен подход за обучение, основан на задачи и проекти в уводните курсове по програмиране, които условно тук са наречени *Програмиране* 1 (П1) и *Програмиране* 2 (П2). Тези два курса традиционно се преподават във всички университети в специалност „Информатика“ и сродните на нея „Компютърно инженерство“, „Софтуерни технологии“ и „Информационни системи“. Наименованията на курсовете са различни, но основните теми, съдържащи се в тях, са много близки. Това унифициране на университетските програми до голяма степен е следствие на „рамките“, определени от публикации на асоциациите по компютърни науки ACM и IEEE, които са лидерите от световен мащаб.

След „*златния век*“ на информатиката в образованието, когато тя беше една от най-желаните специалности в университетите по света, през последното десетилетие се забеляза спад в приема на студенти в информатичните специалности и едновременно с това съществен спад в нивото на обучаващите се студентите [5, 16]. Причините са разнообразни и не винаги са едни същи в различните държави

[13]. У нас, за щастие, все още информатиката е специалност с висок рейтинг сред кандидатстудентите. Но трудно е да се вярва, че световната тенденция няма да даде отражение и в нашите университети. Разнообразието на университетите, в които може да се кандидатства, е твърде голямо и по напълно разбираеми причини много от добрите кандидати избират реномирани световни университети в Европа и Америка. Всичко това налага преосмислянето на методите за преподаване, особено в уводните информатични курсове. Как да изградим курсовете си така, че те да са достъпни за нивото на настоящите студенти, разбира се без да се налага да се правят компромиси в съдържанието и в изискванията към тях? Това е централният въпрос в статията, на който се прави опит за отговор чрез т. нар. *плавен преход от решаване на задачи към разработка на проекти*. Конкретно тук се разглеждат два курса, в които единият (П2) е естествено продължение на другия (П1). Не е трудно да се види, че идеята е приложима и за други курсове, при това не само в специалността „Информатика“. По-долу е споделен опитът, натрупан от няколко университета от Европа и САЩ, в които авторът е преподавал тези дисциплини.

Дисциплините П1 и П2

Ето кратко представяне на дисциплините П1 и П2. Те са уводни курсове в информатиката, които се реализират на определен език за програмиране като например C++, Java, Python и други. В тях студентите изучават основни понятия, методи и техники на програмиране, необходими и за много други дисциплини.

Основните акценти в П1 са теми, ориентирани към изучаването на основни типове данни, управляващи структури, функциите като средство за модулно изграждане на програми, някои подходи за стуктуриране на данни, файлове и др. При завършване на курса от студентите се очаква да умеят да решават несложни задачи чрез създаване на програми, да могат да анализират, тестват и документират програмен код, използвайки определена среда за програмиране.

Основните теми в П2 обикновено включват: обектно-ориентирано програмиране, рекурсивно програмиране, структури от данни и приложения (стек, опашка, списъци и др.), родово програмиране (tempaltes, generic), изключения, управление на паметта, основи на алгоритмичния анализ и др. При завършване на курса от студентите се очаква да умеят да реализират нетривиални програмни проекти, прилагайки изучените подходи и техники на програмиране и структуриране на данните, като следват съответна методология за анализ, проектиране, кодиране, документиране и тестване.

2. Задачи и проекти в курсовете по програмиране

Използването на задачи и проекти е добре позната практика в много дисциплини. Има обаче известни различия в смисъла, в които те се използват. За някои автори

двете понятия са идентични, за други различията не са съществени, а за трети те са различни [9, 10]. Това терминологично „разминаване“ се дължи преди всичко на областта (съответна дисциплина), в която тези методи се прилагат. В курсовете по П1 и П2 двете понятия са добре различими и смисълът, който конкретно се влага в тях в статията е описан по-долу.

Задачи по програмиране

Задачата по програмиране е задание, за изпълнението на което се изисква да се напише програма. Обикновено за решаването на задачата е необходимо директно прилагане на знанията от малка по обем тема, например глава от учебник. Задачите по програмиране са добре формулирани (както по математика) и имат добре определени крайни резултати. Като структура програмата, с която се решава една задача по програмиране, не е сложна и се състои от една до две-три функции, а съответният алгоритъм е интуитивно ясен. В зависимост от начина, по който се използват задачите в целия курс, се познават два основни подхода.

Когато с решаването на задачи се цели практически да се илюстрират свойствата на въвежданите понятия (тип данни, управляваща структура, метод), тогава подходът се нарича *обучение чрез решаване на задачи* (*problem solving*) [9, 10, 15]. Обикновено при този подход в рамките на всяка тема се въвеждат необходимите понятия, подкрепени с примери, а накрая се предлага множество от задачи за самостоятелна работа. Най-често в учебниците по П1 се използва подходът *обучение чрез решаване на задачи* и той е масово прилаган.

Ако целият курс е изграден на базата на задачи, тогава подходът е *проблемно-ориентиран* (*problem-based*) [9, 10, 11, 14, 15]. При този подход задачите определят темата, т.е. те са водещи и едновременно с решаването на задачата се извършва и въвеждане на новите понятия и методи. Този подход се приема добре от студентите, защото въвеждането на всяко ново понятие е предшествано от необходимостта от неговото използване. Трябва обаче да се отбележи, че прилагането на проблемно-ориентирания подход през целия курс не винаги е целесъобразно. На практика много преподаватели, включително и авторът на тази статия, прилагат комбинация от двата подхода, в която водещ е проблемно-ориентираният подход и това е съображението за него да използваме същия термин (*проблемно-ориентирания подход*). Този подход позволява да се препоръча конкретен учебник на студентите, а в клас преподавателят да въвежда някои теми и понятия чрез задачи, подходящи за съответната тема.

Проекти по програмиране

Програмният проект е задание, изпълнението на което изисква решаването на няколко свързани помежду си задачи. В този смисъл на проекта може да се гледа като на сложна задача, обединяваща понятията, въведени в няколко раздела от це-

лия курс. Пълното му изпълнение изисква сериозни усилия и многократно повече време, отколкото времето за решаването на една задача. Много често крайният резултат от един проект има практическа стойност и понякога самият той може да бъде модул от друг проект. Обикновено проектите се формулират без да се посочват всичките детайли. Това означава, че от студентите се очаква сами да вземат решения при уточняване структурата на входните данни, на декомпозицията на проекта на отделни модули, на алгоритмите за решаване на отделните задачи от проекта и др. По този начин те имат пълната свобода да обсъждат своите идеи и творчески да реализират решенията си при проектирането на програмите и избора на алгоритми. Пълното завършване на проекта обикновено изисква разбиването на цялостната работа на два или повече етапа (фази). Обемът от работата и трудността, която трябва да се преодолее в рамките на един проект, по естествен начин определя необходимостта от работа в екип.

Методът на обучение, при който водещи в целия курс са проектите, се нарича *проектно-ориентиран* (*project-based*) подход [1, 4, 10, 11]. Реализацията на този метод не е лесна. На проектите се дава голямо тегло от крайната оценка на студента, което създава напрежение и стрес по време на семестъра. Необходими са специфични грижи от страна на преподавателя преди започването на курса и особено по време на провеждането му, когато той влиза в ролята на консултант и съветник. За успешното прилагане на подхода е необходима солидна предварителна подготовка на студентите, силна мотивация да завършат проекта в срок и умение да работят в екипи.

2.1. Трудности в уводните курсове по програмиране

Затрудненията, които обикновено студентите срещат в двата уводни курса по програмиране, са разнообразни, но посочените по-долу са от особена важност:

– Повечето от студентите нямат необходимото ниво на абстрактно мислене. Това се усеща още в самото начало на първия курс, когато се въвежда понятието „*променлива*" като абстракция на „поле от паметта", и напълно се потвърждава при разглеждането на понятията „*тип данни*", „*функция*", „*параметър*". Нека добавим, че абстракцията е централна тема във втория курс при въвеждане на *класовете* и *структурите от данни*, дефинирани като класове. Всичко това води до сериозни трудности при усвояването на материала в двата курса някои от студентите да не са в състояние да ги завършат.

– Студентите нямат добра предварителна представа за избраната от тях специалност (Computer Science/Computer Engineering/Software Engineering/Information Systems) [5]. Някои от тях не са убедени, че са направили правилен избор и затова *не са достатъчно мотивирани* да получат солидна подготовка по програмиране. По тази причина по време или след завършване на П1 около 15% от студентите

осъзнават, че трябва да променят специалноста си в някоя по-приложна област, като например информационни технологии.

– Студентите не осъзнават добре необходимостта от *време*, през което да се упражняват в писането, тестването и документирането на програми. Въпреки изричното изискване да тестват програмите в своите домашни задания, те свеждат тестването до еднократно изпълнение на програмата с не добре подбрани (тривиални) входни данни. Твърде късно те разбират, че средата за програмиране е техният най-добър учител в първите месеци на обучението им по програмиране. Те пренебрегват ролята от проектирането на програмите и в повечето случаи директно преминават към писането на програмен код.

– Студентите трудно приемат да работят в екипи и причините са разнообразни, някои от които са напълно разбираеми в първия курс [8, 12].

– Поради претовареност с други дисциплини и с дейности извън университета, студентите нямат необходимото време за да работят върху заданията си по програмиране.

Важно е да се отбележи, че студенти, които трудно покриват минималните изисквания на П1, още по-трудно успяват в П2. Ето защо от самото начало към тези студенти трябва да се обръща специално внимание. Понякога обмислянето на смяна на специалността може да се окаже твърде полезна за самия студент, ако това се извърши непосредствено след П1. Това не е лош вариант, защото другият, при който студентът напуска университета, е много по-лош.

2.2. Идеи за преодоляване на трудностите в уводните курсове по програмиране

Анализът на посочените по-горе проблеми дава възможност да се направят някои изводи и да се формулират съответни идеи за решения. Някои от тях са подсказани директно или индиректно от самите студенти, а повечето са резултат от проведени експерименти.

Мотивираният студент винаги успява. Ако приемем тази мисъл за максима, ясно е, че всеки опит за *допълнителна мотивация* на студентите е важен. За постигането ѝ могат да се прилагат посочените по-долу подходи.

Домашни задания. Задачите за домашна работа в П1 трябва да се разискват в клас както при задаването им, така и след проверката им. Проектите в П2 следва да се разглеждат обстойно при задаването им, а резултатите да се дискутират подробно при тяхното представяне в клас. Това не е загубено време. По този начин се повишава мотивицията на студентите и те с по-голямо старание подготвят заданията си.

Тестове без оценка. Част от мотивацията на студентите идва от техния текущ успех. Повечето студенти, които имат слаб текущ успех, бързо се демотивират и крайният резултат е напускане на курса. Ето защо е добре редовно (почти всяка

седмица) студентите да имат кратки и прости тестове без оценка. Резултатите от тях са изключително полезни за самите студенти. Едновременно с това резултатите от тестовете информират и преподавателя за нивото на усвояване на материята.

Въпреки че и в двата курса се въвеждат много теоретични понятия и техники, то практическият компонент в тях надделява. Създаването на практически умения, необходими при решаването на задачи, както и изграждането на професионален стил на програмиране са основни цели. Идеи за тяхното постигане в рамките на тези два курса са представени накратко по-долу.

Самообучение от собствени грешки. Студентите разбират понятията и методите в П1 най-вече от примери, но ги научават, когато сами решават задачи. Много добре те се учат от грешките, които допускат при тестовете (без оценка) и при решаването на задачи.

Обучение чрез модификация на програмни текстове. Преди да започне да пише собствени програми всеки начинаещ програмист започва с четене и анализиране на чужди програмни текстове. Следващата стъпка от обученито е извършването на експерименти чрез модифициране на програмни кодове. Третата, може би най-съществена стъпка, е тази, при която от програмен текст (програма, функция) с определена функционалност се генерира нов програмен текст с променена функционалност. През този етап студентите се учат да „разчитат“ чужд програмен текст и след това чрез съответни промени да го адаптират към новите изисквания.

Редици от задачи и проекти. Решаването на всяка задача по математика изисква съответен анализ. Така е и със задачите по програмиране. Чрез съответен анализ първоначалната задача се разбива на по-прости и евентуално познати подзадачи. Някои от подзадачите също подлежат на разбиване, докато всички подзадачи са познати. Това е основен подход при декомпозирането на програмите на отделни модули (функции/класове). Не се познава общ метод, по който да се извършва декомпозирането на произволна задача в множество от по-прости задачи. Това се постига с опита от решаването на разнообразни конкретни задачи. Един метод, който авторът прилага, е описан с пример в т. 4. Същността на метода основава т. нар. „*редица от задачи*”. Всяка редица от задачи е множество от няколко задачи, които могат се подредят линейно по такъв начин, че за решението на всяка задача (без първата) да се използват решенията или идеите на предишни задачи от същата редица [2]. Става ясно, че всяка задача е някакво обобщение или „разширение” на една или няколко други задачи от редицата. По този начин, учейки студентите на *обобщаване* и *декомпозиране,* пряко се въздейства върху формирането на абстрактното им мислене. Проведените експерименти с този подход показват, че той е твърде подходящ за П1. Програмните проекти в П2 също могат да бъдат формулирани като *редици от проекти*. Примери на такива редици са даден в т. 4, 5 и т. 6.

Проектиране на програми. Дискусията, отнасяща се до проектиране, стил и документиране на програми, трябва да започне още в теми от П1 [7]. В противен случай студентите омаловажават тези аспекти на програмирането, считат ги за загубено време и в тях не се изграждат необходимите навици, необходим при разработването на програми. И ако в П1 това не е от решаващо значение, то в П2 студентите вече изпитват сериозни затруднения при цялостното разработване на програмни проекти.

Работа в екип. Добре е още в начало на П2 студентите да започнат да работят в екипи по двама или трима. Това е особено важно при работа върху проекти. Така студентите обменят идеи помежду си и тези, които са с по-слаба подготовка, имат възможност да направят съществен прогрес за кратко време. Начинът, по който се формират екипите, е от съществено значение и по тази тема много публикации [6, 12]. Тук ще отбележим само, че макар и несъвършена, процедурата за формиране на екипи, при която всички екипи имат реални шансове успешно да завършат проекта, е напълно приемлива за П2.

3. Какво означава понятието "*плавен преход*"?

Подходът, който се описва тук, е изграден на базата на многогодиншни наблюдения и експерименти, целящи преодоляването на проблемите, описани по-горе. Обобщените изводи могат да се представят в три основни групи:

– Решаването на задачи е важен елемент от цялостното обучение в уводните курсове по програмиране. Решавайки задачи, студентите придобиват практически опит и това трябва да бъде в основата на П1.

– В П1 студентите изпитват трудности да работят върху програмни проекти. Това е лесно обяснимо. В рамките на един семестър не е лесно да се усвоят синтактичните и семантичните особености на изучавания език за програмиране. По тази причина не е реалистично те да работят върху проекти, в които акцентът е повече върху проектирането на програмите. Да отбележим, че има публикации, в които авторите предлагат да се използват проекти още в П1 [8], но навярно в тези случаи става дума за студенти, които са добре мотивирани и имат добра подготовка още от средното училище.

– Директното преминаване от решаването на задачи в П1 към разработването на проекти в П2 *стресира и обезкуражава* голяма част от студентите. Ето защо този преход следва да се извърши *плавно*. Прилагането му трябва да се обмисли още по време на П1, като се разглеждат задачи, които по-късно се използват в проекти на П2. Идеята да се изграждат нови понятия и методи чрез използване на познати такива е основна практика, но в случая на уводните курсове по програмиране могат да се посочат два специфични положителни ефекта:

(1) Въвеждането на нов проект, който съдържа решавани преди това задачи, изисква по-малко време, а студентите сръчно се ориентират в декомпозицията му

на отделни модули. Те започват работа със самочувствие и увереност, че могат да се справят. Това проличава от активното им участие още при първото обсъждане на проекта в клас.

(2) Когато една задача е разглеждана и решавана като отделна програма или функция в П1, използването ѝ като подзадача в програмен проект в П2 обикновено изисква известно модифициране. Така на студентите се подсказва, че трябва да адаптират програмен код след съответна творческа модификация. Работата върху адаптирането на дадена функция за нови цели е първата стъпка, целяща стимулирането им към разработката на софтуер за многократно използване. Те вече имат известен опит от П1, но в подобни случаи реално осъзнават необходимостта от документирането на програмия код и в частност се убеждават в необходимостта да документират своите програми.

От задачи към проекти

Подходът на *плавен преход* от задачи към проекти обикновено изисква частично обновяване на съдържанието на П1, отнасящо се до задачите, разглеждани в час и задачите за извънкласна работа. Обновяването на съдържанието на П2 е също необходимо и то се отнася до проектите, които ще се разработват в рамките на този курс. В случая проектите са определящи, защото от тяхното декомпозиране се „извличат" задачи, които следва да се разгледат в П1. Ето накратко и описанието на процедурата, описана в пет стъпки, която може да се използва при обновяването на двата курса:

1°. Формулират се проекти, които ще бъдат разглеждани в П2. Ако преподавателят има известна представа за общата подготовка на студентите от П1, то това ще му помогне по-добре да прецени броя на проектите и тяхната сложност.

2°. От декомпозирането на проектите се формулират задачи, които са подходящи за разглеждане в теми на П1. Понякога задачите могат да бъдат формулирани в по-опростен вариант. В такива случаи при разпознаването им като подзадачи в проекти на П2 ще се наложи извесно обобщаване. Обикновено това се реализира с въвеждане на допълнителни параметри на функции и/или чрез промяна на функционалността им.

3°. В П1 допълнително се разглеждат задачи (в клас или като домашни задания), чиято идея и/или структура има отношение към някои от проектите в П2.

4°. Ако някоя от задачите от т. 2° или т. 3° е твърде сложна, следва да се „разбие" на редица от задачи.

5°. Решенията (т. 2° и т. 3°) би следвало да са достъпни за всички студенти. В случая може да се използват съотвени директории на сървера, на който се съхранява учебната документация за курсовете П1 и П2. Това дава възможност всички студенти да имат достъп до решенията (от даден момент нататък) на всички задачи.

Броят на проектите може да бъде два или три и по изключение четири. Точният брой зависи най-вече от нивото на студентите и тяхната мотивация да работят върху проекти. Това отнапред не винаги се знае, поради което подготовката на П1 и П2 се прави при предположение, че броят на проектите ще бъде три, а след завършването на П1 техният брой се решава окончателно. Първият проект е най-труден за студентите, затова и помощта от страна на преподавателя трябва да е най-голяма. Като резултат това означава, че процедурата, описана по-горе, трябва да се извърши поне за първия проект. Опитът, който студентите добиват от първия проект, позволява по-голяма самостоятелност при работата върху следващитите проекти.

Описаната процедура се реализира лесно, ако и двете дисциплини се предават от един и същ преподвател. Преподаването им от различни преподаватели може да породи някои затруднения, но те не са непродолими.

По-долу следва кратко представяне на дейностите (фазите), които се извършват по време на всеки от проектите [3, 7].

Дейности преди започване на проекта. Тази фаза е подготвителна. Ако е необходимо, могат се въвеждат и нови понятия. Следва формулиране на самия проект. Знанията, небходими за разработването на проекта, се посочват явно, като се дават и указания за налични ресурси (учебник, Интернет връзки и др). Накрая се формират и екипите, като специално се посочва и съответният ръководител.

Работа върху проекта. Необходимото време за работа по един проект е от две до три седмици. В началото е важно студентите добре да разберат заданието на проекта. Те имат възможност да поставят въпроси в клас за изясняване на някои детайли. Всеки екип разпределя самостоятелно задълженията между своите членове. Това става в едно от техните заседания, които могат да се проведат и online. Най-важните решения, които те вземат, се документират и това става част от пълната документация на проекта.

Представяне на проекта. Представянето на проекта се извършва официално в клас в рамките на 8-10 минути. Презентацията протича по предварително изяснен сценарий, за да няма загуба на време. Основни акценти при презентацията са модулната структура на проекта, използваните алгоритми, данните, с които е тестван проектът и въпросите, зададени от членовете на другите екипи.

Оценка на проекта. При задаване на проекта се указват и критериите за оценка. А те включват: (1) модулна структура; (2) „работещ“ програмен код; (3) документация и стил на програмиране; (4) оценка на реализираните алгоритми; (5) избор на данните за тестване и (6) умението за презентация на групов проект. Ден преди официалната презентация на проекта студентите предават електронен вариант, достъп до който има само преподавателят.

Дейности след завършване на проекта. След завършването на всички презентации студентите избират един (не повече от два) - най-добрият проект, който се публикува

на сървера. Достъп до проекта имат всички студенти. Те могат допълнително да анализират текстовете на програмите и да взаимстват идеи от тях за някои от следващите проекти. Това е една допълнителна възможност за студентите да установят по-тесен контакт помежду си. За следващия проект съставът на екипите се променя. Всичко това се извършва с намерението следващият проект да бъде по-успешен.

Обратната връзка за току що завършилия проект, е от съществено значение за преподавателя при организирането и провеждането на следващите проекти. Източници на информация могат да бъдат: (1) споделени мнения на студенти; (2) директно от презентацията на проекта от студентите; (3) от формуляра за самооценка, който всеки студент предава заедно с проекта си и е част от общата документация на проекта.

4. Пример: Редица от задачи „*Пресмятане на прости изрази*“

Една задача, която многократно се разглежда в П1 и П2 по различни поводи и в различни варианти, е задачата за пресмятане на изрази. Тя е следната:

Р*. [*Основна задача*] Даден е израз без скоби със следния общ вид:

$$d_1 \otimes d_2 \otimes d_3 \otimes \ldots \otimes d_n ,$$

в който d_1, d_2, d_3, …, d_n са цели положителни числа, а със знака $\otimes$ са означени произволни двуаргументни аритметични операции {+, -, *, /, %}, за които се предполага, че са с еднакъв приоритет. Да се напише програма за пресмятането на израза.

Решаването на тази задача изисква съответна предварителна подготовка. За тази цел тя може да се декомпозира в редица от следните четири задачи, които водят до нейното пълно решение.

Р1. Да се пресметне израз, имащ вида: $d_1 \otimes d_2$.

Задачата може да се решава преди въвеждането на функции, но добре е да се разгледа по-късно и нейн вариант, реализиран чрез функция със следния прототип:

```
int p1(int a1, char op, int a2);
```

Р2. Да се пресметне израз, имащ вида: $d_1 \otimes d_2 \otimes d_3$.

Задача Р2 може да се решава преди въвеждане на функции, използвайки вложени управляващи структури, но добре е да разгледа и нейн вариант, реализиран чрез функция с прототип:

```
int p2(int a1, char op1, int a2, char op2, int a3);
```

Сега решението на задача Р1 може директно да се използва в решението на задача Р2. За целта е достатъчно да се отбележи, че $d_1 \otimes d_2 \otimes d_3 = (d_1 \otimes d_2) \otimes d_3 = d_4 \otimes d_3$.

Р3. Естественото обобщение на Р2 води до решаването на първоначалната задача и реализацията й чрез функция с прототип:

```
int p3(int a[], char op[], int n);
```

Р4. Изразът може да се зададе чрез знаков низ. В този случай задачата изисква разглгеждането на подзадача, с която се разпознават целите числа (друга важна

задача) и операциите, след което може да се приложи решението на P3. Функцията, решаваща задачата, ще е с прототип:

```
int p4(string exp);
```

Ето и зависимостта между задачите от редицата `P1`, `P2`, `P3`, `P4`, посочена със стрелки:

`P1` → `P2`, `P1` → `P3`, `P2` → `P3`, `P3` → `P4`, `P3` → `P*`, `P4` → `P*`

Пресмятането на изрази, в които се използват скоби и естественият приоритет на операциите, е още едно обобщение на първоначалната задача P*, но за нея е добре да се построи отделна редица от задачи, която би включвала и използването на структурата от данни *стек*.

5. Пример: PolyLine – Програмен проект за свързани списъци

По-долу накратко е представен примерен проект.

Формулировка на проекта. *Полигон* е редица от точки $P_1, P_2, \ldots, P_n$, $n \geq 1$ в равнината, наречени негови *върхове*. Върховете са представени с координатите си (фиг.1). Като „изродени“ варианти на полигон се допуска и полигон, състоящ се от един връх и „празен“ полигон (без нито един връх). Да се напише клас `PolyLine` за опериране с полигони. Представянето на обект от класа (произволен полигон) да се извърши чрез свързан линеен списък.

Множеството от операции на класа трябва да включва поне следните:

– Конструктори: конструктор по подразбиране (създава „празен“ полигон), конструктор за копиране и конструктор, построяващ полигон с данни от текстов файл, съдържащ полигон, записан в XML-формат (фиг. 1).

– Достъп до координатите на връх от полигон.

– Изменение на структурата на полигон чрез добавяне (вмъкване) на нов връх, отстраняване на връх и промяна на координатите на връх.

– Пресмятане на дължината на полигон.

– Съхраняване на полигон в текстов файл, представен в XML формат.

За тестване на класа да се напише функция, която под формата на меню да позволява тестване на всяка от операциите.

От този проект могат да бъдат „отделени“ множество операции, които да се разгледат като задачи в П1. Такива са например:

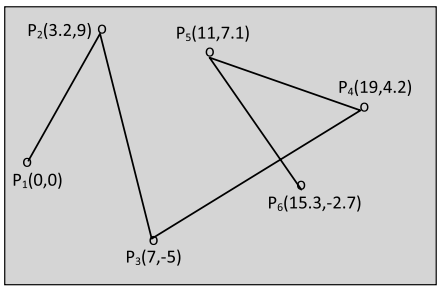
– преобразуване на знаков низ в цяло число;

– преобразуване на знаков низ в реално число;

– пресмятане дължината на отсечка, определена с координатите на крайните си точки;

– четене на данни от текстов файл;

– създаване на текстов файл, чиято структура е в XML-формат (силно опростен вариант).




```
<PolyLine>
<P1> <x>0</x> <y>0</y> </P1>
<P2>  <x>3.2</x>  <y>9</y> </P2>
<P3> <x>7</x> <y>-5</y> </P3> <P4> <x>19
</x> <y>4.2</y> </P4>
<P5>
  <x>11</x>  <y>7.1</y>
</P5>
<P6> <x>15.3</x> <y>-2.7</y> </P6>
</PolyLine>
```

Фигура1. Полигон с шест върха, представен в XML-формат

В някои случаи проектът може да бъде опростен или усложнен в зависимост от общото ниво на студентите в курса. Следват няколко идеи: използване на класове от библиотеките на средите за програмиране (например `STL vector`, `STL list` в `C++`). Ако решим да завишим алгоритмичната страна на проекта, могат да се изискват допълнителни операции, с които да се проверява дали полигонът е затворен ($P_1 \equiv P_n$), дали е изпъкнал, дали е с непресичащи се страни, да се пресметне лицето му, в случай, че е затворен и е с непресичащи се страни, и др.

6. Пример на редица от проекти

Акцентът в разгледания пример пада върху свързани линейни списъци. В П2 могат да се предложат и следнните два проекта, в които основната структура от данни е стек и съответно двоични дърво.

– `XML_Stack:` Проект за четене на XML-файл и проверка за коректност в структурата му. Централна роля в проекта е на структурата от данни *стек*, с която се проверява коректното влагане на таговете на XML елементите. Това е важна задача, чиито първи и най-прост вариант трябва да се разгледа в П1 под формата на следната задача: „Даден е аритметичен израз, записан в знаков низ. Да се определи коректността на вложение на скобите на израза.“

– `BTree:` Проект за представяне на данните в двоични дървета и опериране с тях. Понеже XML-файловете имат йерархична структура, естествено е данните да се представят на външна памет в XML-формат.

Връзката между отделните проекти се обуславя от структурата на входните данни, които в случая са текстови XML-файлове. Предлагайки тези проекти в реда `XML_Stack`, `PolyLine` и `BTree`, студентите имат възможността да използват, евентуално след съответна модификация, функции и класове от предишни задачи и проекти. Има няколко съображения за включването на XML-файлове в тази редица от проекти. Ето три от тях:

– Централна тема в П2 са структурите от данни и реализацията им с класове. Понеже данните с твърде сложна структура могат относително лесно да се съхраняват на външна памет във файлове с XML-формат, то използването на този формат е напълно естествено;

– XML-файловете са текстови файлове и това позволява с текстов редактор директно да се създават разнообразни входни тестове за проектите. Всеки от проектите изисква създаване на конструктори, с които се „строят“ обекти, чиито данни се четат от текстов файл;

– За изясняването на структурата на XML-файловете са необходими не повече от десетина минути.

7. Заключителни бележки

Основната идея на подхода на плавен предход от задачи към проекти по програмиране е да се осигурят условия за успешното преодоляване на трудностите, които студентите срещат още в първото и особено във второто ниво на курсовете по програмиране. Описаният подход е експериментиран многократно в класове, в които броят на студентите е до около 25 в клас. Естествено е в случаите, когато класовете са големи, някои от етапите на организирането на проектите да се провеждат по време на семинарни занятия. Създаването на атмосфера на колегиалност и стимулирането на работа в екипи съществено подпомага успешното прилагане на представения подход.

Въпреки че подходът е ориентиран към уводните курсове по програмиране, той е приложим и за други дисциплини, включително и извън рамките на специалността „Информатика“.

ЛИТЕРАТУРА

Andreas Breiter, Gorschwin Fey, and Rolf Drechsler. (2005). Project-Based Learning in Student Teams in Computer Science Education. *Facta Universitais* (NIS), Vol. 18, No. 2, 165-180

Azalov, P., F. Zlatarova. (2003). Teaching Programming through Successive Problem Transformations. *The Journal of Computing Sciences in Colleges*, Vol.18, No.4, 175-182.

Azalov, P., D. Richards. (2004). Project-Based Teaching of Intermediate Programming. *Proceedings of the International Symposium* IGIP/IEEE-ES/ASEE, Switzerland, 30-35.Barg M, Fekete A., Greening T., Hollands O., Kay J., and H. Kingston J. Problem-Based Learning for Foundation Computer Science Courses, 1-27. http://sydney.edu.au/engineering/it/~judy/PBL/tr_cse_pbl99.pdf (сайтът е посетен за последен път на 17 май 2013)

Carter L. (2006). Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science. SIGCSE '06, 27-31

David Casperson. Experience with Team Projects in a second-semester C++ Programming Course. http://www.cs.ubc.ca/wccce/Program03/papers/Casperson.html (сайтът е посетен за последен път на 17 май 2013)

ACM SIGCSE Bulletin, Volume 12 (1), 25-31.

Joel Adams. (1998). Chance-It: An Object-Oriented Capstone Project For CS-1. SIGCSE '98 *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer Science Education*, 10-14.

Jorge E. Pérez, Javier García, Isabel Muñoz, Almudena Sierra Alonso, Pilar López Puche. (2010). Cooperative Learning vs. Project Based Learning. *IEEE EDUCON Education Engineering 2010* – The Future of Global Learning Engineering Education. Session T1A, 87-98.

Julie E. Mills, David F. Treagust. (2003). Engineering Education - Is Problem-Based or Project-Based Learningthe Answer? *Australasian Journal of Engineering Education,* 1-16.

Kuru S. (Ed.). (2007). Problem Based Learning. TREE – Teaching and Research in Engineering in Europe Special Interest Group B5 "Problem based and project oriented learning" Isik University. http://www3.unifi.it/tree/dl/oc/b5.pdf (сайтът е посетен за последен път на 17 май 2013).

Lingard R. (2011). Teaching and Assessing Teamwork Skills in Engineering and Computer Science. *Frontiers in Education Conference* (FIE), F1C-1 - F1C-5.

Kinnunen P. and Lauri Malmi. Why Students Drop Out CS1 Course? ICER'06, pp.97-108, 2006.

Samuel B. Fee & Amanda M. Holland-Minkley. (2010) Teaching Computer Science through Problems, not Solutions. *Computer Science Education*, Volume 20, No 2, 129-144. Special Issue: Innovative Pedagogies in Computer Science Education.

Savery, J. R. (2006). Overview of Problem-based Learning: Definitions and Distinctions. *Interdisciplinary Journal of Problem-based Learning*: Vol. 1 (1), 1-16.

Zweben S. (2013). Computing Degree and Enrollment Trends (from the 2001-2011 CRA Taulbee Survey, Computing Research Association) http://cra.org/resources/taulbee/ (сайтът е посетен за последен път на 17 май 2013).

SMOOTH TRANSITION FROM PROBLEMS TO PROJECTS IN INTRODUCTORY PROGRAMMING COURSES

Abstract. How to build up our courses that are comprehensible to the current students without affecting the established requirements and without allowing compromises relevant to the course content? This is the main question discussed in the paper. The corresponding answer is based on the *smooth transition from problems to projects*. The idea of this approach consists of the implementation of the *problem-based teaching* in the Computer Science I course and then continuing with the *project-based teaching* in the Computer Science II course. The analysis of the possible difficulties, which could be met by the students attending these courses, and corresponding ways to overcome them are also presented in the text.

Every set of interconnected problems is structured as a sequence of problems so that each problem could derive from preceding problems of the same sequence. Similarly, projects are organized as a sequence of projects whose components such as functions and classes could be incorporated into the design and development of next projects. The smooth transition from problems to projects can be achieved by decomposing the projects to problems which should have been considered already in Computer Science I.

Pavel Azalov
✉ Associate Prof. of Computer Science, Ph. D.
Pennsylvania State University, USA
E-mail: pka10@psu.edu