Записи

Трифон Трифонов

Увод в програмирането,
спец. Компютърни науки, 1 поток, 2018/19 г.

17 януари – 20 февруари 2019 г.

Логическо описание

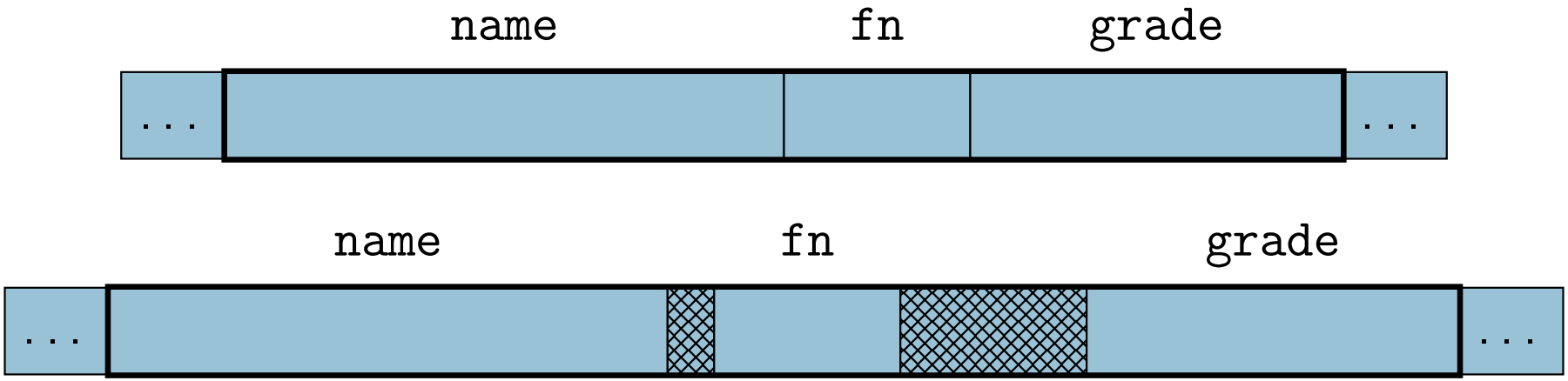
Записът е:

- съставен тип данни
- представя крайна редица от елементи
- редицата е с фиксирана дължина
- елементите могат да са от различни типове
- произволен достъп до всеки елемент

Дефиниция на запис

- `struct` <име> `{` <поле> { <поле> } `};`
- <поле> ::= <тип> <идентификатор> {`,` <идентификатор> }`;`
- **Примери:**
- `struct Point { double x, y; };`
- ```
  struct Student {
    char name[45];
    int fn;
    double grade;
  };
  ```
  ```

Физическо представяне




```
struct Student {
  char name[45];
  int fn;
  double grade;
};
```

- `sizeof(S)` — големина на структурата `S`
- `sizeof(Point)` = 16
- `sizeof(Student)` = 64
- Защо?
- Полетата в структурите се подравняват до адрес кратен на големината им
- Улеснява обработката от процесора

Променливи от тип запис

[struct] <тип_запис> <име> [= { <израз> {, <израз> } }]
{, <име> [= { <израз> {, <израз> } }] };

Примери:

- `Point p1, p2 = { 1.2, 3.4 };`
- `Student s1 = { "Иван Колев", 61234, 5.75 };`

Операции над записи

- Присвояване (=)
  - Могат да се присвояват само структури от един и същи тип
  - `Point p3 = p1; p3 = p2;`
  - ~~`Student s1 = p1;`~~
- Достъп до поле (.)
  - <променлива>.<име_на_поле>
  - `p1.x = 1.3; p2 = p1; p2.y = -p2.y;`
  - `s1.fn = 41000; cout << s1.grade;`
  - `cin.getline(s1.name, 45); s2 = s1;`
  - `int* p = &s1.fn;`
  - `char* s = s1.name;`
- Няма операции за вход и изход
  - ~~`cin >> s1;`~~
  - ~~`cout << p1;`~~

Масив от записи

Можем да комбинираме свободно съставните типове данни, за да създаваме произволно сложни потребителски типове данни.

- `Student s[10] = { { "Петър Петров", 80000, 5.5},`
  `{ "Стефани Стефанова", 60000, 6 } };`
- `strcpy(s[2].name, "Иван Иванов");`
- `cout << s[1].fn;`
- ```
  for(int i = 0; i < n; i++)
    cin >> s[i].grade;
  ```
  ```

Запис от записи

```
struct Team {
  Student s1, s2;
  char name[30];
};
```

- `Team team = { { "Диана", 80003, 5 },`
  `{ "Радослав", 60004, 6}, "Дислав"};`
- `cout << team.name << ' ' << team.s2.name;`
- `double teamGrade = (team.s1.grade + team.s2.grade) / 2;`

Записи и функции

- Записите като параметри
  - Предават се по стойност, като простите типове данни
    - за разлика от масивите!
  - промените във функциите са локални
- Записите като върнат резултат
  - Връщат се по стойност, като простите типове данни
  - Връща се копие на записа

Задачи за записи

1. Да се въведе масив от студенти
2. Да се изведат студентите в таблица
3. Да се намери средния успех на всички студенти
4. Да се подредят студентите по ФН

Указатели и препратки на записи

Можем да правим указатели и препратки на записите и техните полета

- `Student* ps1 = &s1, *ps2 = nullptr;`
- `ps2 = ps1; *ps2 = s2;`
- `Student& s3 = s1;`
- `cout << s3.name;`
- `s3 = s2;`
- Записите могат да се предават като параметри на функции по стойност, указател и препратка

Достъп до поле на запис чрез указател

<указател_към_запис> -> <поле>

- еквивалентно на (*<указател_към_запис>).<поле>
- `ps1->grade += 0.5;`
- `cout << ps2->fn;`
- `Team* pteam = &team; cout << pteam->s1.name;`

Рекурсивни записи

```
struct Employee {
  char name[64];
  Employee boss;
};
```

- записът се дефинира чрез себе си
- `sizeof(Employee)` = ?
- забранена рекурсия!

Рекурсивни записи — правилният начин

```
struct Employee {
  char name[64];
  Employee* boss;
};
```

- записът се дефинира чрез себе си...
- ... но съдържа указател към себе си
- `sizeof(Employee)` = 72
- ```
  Employee rector = { "Герджиков", nullptr },
           dean = { "Първанов", &rector },
           chair = { "Нишева", &dean },
           lector = { "Трифонов", &chair };
  ```
- `cout << lector.boss->boss->boss->name;`
  ```

Абстракция със структури от данни

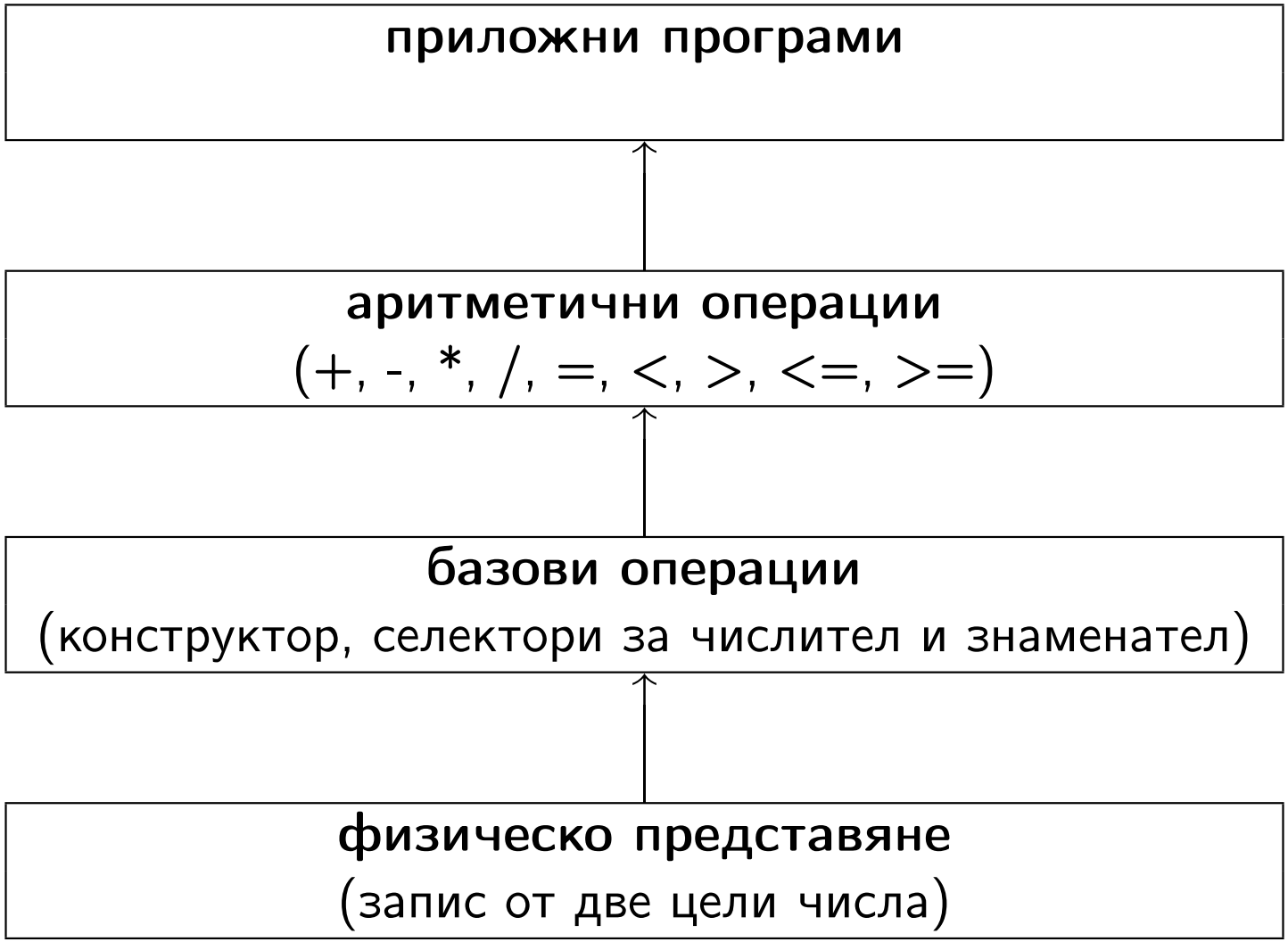
Записите позволяват дефиниране на потребителски типове данни и операции над тях.

Пример: Тип "рационално число"

- **Логическо описание:** обикновена дроб
- **Физическо представяне:** запис с числител и знаменател
- **Базови операции:**
  - конструиране на рационално число
  - получаване на числител
  - получаване на знаменател
- **Аритметични операции:**
  - събиране, изваждане
  - умножение, деление
  - сравнение
- **Приложни програми**

Идея: да изолираме вътрешното представяне на типа данни от операциите над него

Нива на абстракция



Обектно-ориентирано програмиране

- структуриране на данните на концептуално ниво като “обекти”
- обектите имат **свойства и методи** за работа с тях
- могат да се дефинират **нива на достъп** до компоненти на обекта
- представянето на обекта обикновено се **капсулира**
- еднотипни обекти се описват чрез **класове** от обекти
- възможностите на обект могат да се разширяват (**наследяване**)
- един обект може да има различни проявления (**полиморфизъм**)