Записи

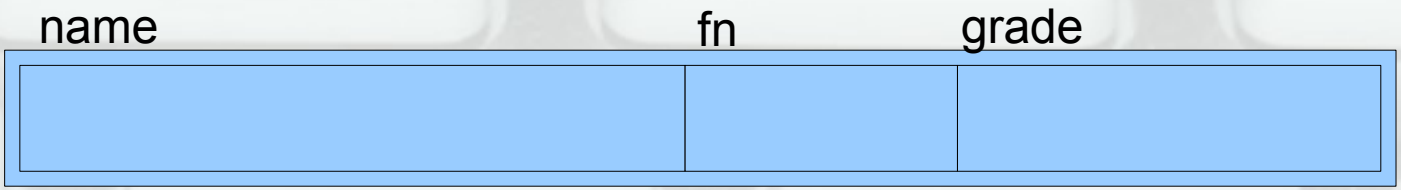
Логическо описание

- 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> };
- struct Complex { double re, im; };
- struct Student {
  char name[100];
  int fn;
  double grade;
  };

Физическо представяне



- sizeof(S) — големина на структурата S
- подравняване до машинната дума

Дефиниция на променливи от тип запис

- [struct] <тип_запис> <име>
  [= { <израз> {, <израз> } }]
  {, <име>
  [= { <израз> {, <израз> } }] };
- Complex z1, z2 = { 1.2, 3.4 };
- Student s1 = { “Иван Колев”, 44444, 4.4 };

Операции над записи

- Присвояване (=)
- Достъп до поле (.)
- Не са позволени вход (>>) и изход (<<)!

Операция за достъп до поле

- <променлива_запис>.<име_на_поле>
- Примери:
- z1.re = 1.3; z2 = z1; z2.im = -z2.im;
- s1.fn = 41000; cout << s1.grade; cin.getline(s1.name); s2 = s1;
- int* p = &s1.fn; char* s = s1.name;

Масив от записи

- Student s[10] = { { “Петър Петров”, 80000, 6 }, { “Стефан Стефанов”, 80001, 5.5 } };
- 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[100];
  };
- Team team = { { “Иван”, 80003, 5 },
  { “Мария”, 80004, 6}, “И&М” };
- cout << team.name << team.s2.name;
- double avg = (team.s1.grade +
  team.s2.grade) / 2;

Записи и функции

- Записите като параметри
  - предават се по стойност като прости типове данни (за разлика от масиви!)
  - промените във функциите са локални
- Записите като върнат резултат
  - връщат се по стойност като прости типове данни
  - връща се копие на записа

Задачи

- Да се въведе масив от студенти
- Да се изведат студентите в таблица
- Да се намери средния успех на всички студенти
- Да се подредят студентите по факултетен номер

Указатели и псевдоними на записи

- Student* ps1 = &s1, *ps2 = NULL;
- ps2 = ps1; *ps2 = s2;
- Student& s3 = s1;
- cout << s3.name;
- s3 = s2;
- Записите могат да се предават по стойност, указател и псевдоним

Операция за достъп до поле на запис чрез указател

- <указател_към_запис>-><поле>
- еквивалентно на (*<указател_към_запис>).<поле>
- ps1->grade += 0.5;
- cout << ps2->fn;

Рекурсивни записи

- struct Employee {
  char name[100];
  Employee boss;
  };
- записът се дефинира чрез себе си
- колко памет заема този запис?
- забранена рекурсия!

Рекурсивни записи

- struct Employee {
  char name[100];
  Employee* boss;
  };
- записът се дефинира чрез указател към себе си

  Employee rector = { “Илчев”, NULL },
  dean = { “Великова”, &rector },
  chair = { “Димитров”, &dean },
  assistant = { “Трифонов”, &chair };
- cout << assistant.boss->boss->boss->name;

Абстракция със структури от данни

- Записите позволяват дефиниране на потребителски типове данни и операции над тях
- Идея: изолиране на вътрешното представяне от операциите

Рационални числа

- Логическо описание: обикновена дроб
- Физическо представяне: запис с числител и знаменател
- Базови операции: създаване на рационални числа, намиране на числител и на знаменател
- Аритметични операции: събиране, изваждане, умножение, деление
- Други операции: въвеждане и извеждане

Нива на абстракция

Обектно-ориентирано програмиране

- дефиниране на нива на достъп
- капсулиране на представянето на обект
- тясно обвързване на обекта с операциите над него
- разширяване на възможностите на обект (наследяване)
- различни проявления на един обект (полиморфизъм)