# 01 Структури

Съдържание:

1. [Дефиниране и използване на **структури**. Основни операции.](#_Дефиниране_и_използване)
   1. [Задача 1](#_Задача_1.)
2. [Функции и структури.](#_Функции_и_структури.)
3. [Структури и масиви.](#_Структури_и_масиви)
   1. [Задача 2](#_Задача_2_1)
4. [Вложени структури. Рекурсивно използване на структури.](#_Вложени_структури._Рекурсивно)
   1. [Задача 3](#_Задача_3)
   2. [Задача 4](#_Задача_4)
5. [Изброен тип.](#_Изброен_тип)
   1. Задача за домашна работа.
6. [Обединения.](#_Обединения)

## Дефиниране и използване на структури. Основни операции.

Кои са основните типове данни, които познаваме? int, float, double, char… Променливи от тези типове представят атомарна единица информация. За да се представи по-сложна, съставна информация, С++ използва **структури**. Структурата е колекция от прости променливи, които могат да бъдат от **различен** тип. Масивите също съдържат множество променливи, но **всички** са от **един** и **същи** тип. Елементите на структурата, променливите, в които се съхраняват данните се наричат **член-данни**.

### Дефиниране на структура

Дефиницията на структурата описва начинa, по който е организирана структурата, кои полета ще бъдат включени в нея. Дефиницията на структурата започва с ключовата сума ***struct***, последвана от ***името*** на структурата. Във фигурни скоби се описват членовете на структурата. Дефиницията завършва с **;**.

#### Задача 1.

Да се дефинира структура, която описва триъгълник чрез трите му страни. Да се дефинират функции, които създават и извеждат триъгълник, намира периметъра и лицето му. Да се напише програма, която създава триъгълник по зададени страни, извежда триъгълника, намира и извежда лицето и периметъра му.

|  |
| --- |
| struct Triangle  {  double a, b, c;  }; |

С така дефинираната структура сме създали собствен тип данни, но не дефинираме променливи от типа на структурата. Не се зададеля памет, за разлика от дефиницията на обикновените променливи, при което се осъществява и заделяне на памет.

Как се дефинират **променливи** от типа на структурата?

|  |
| --- |
| Triangle ABC; |

Промелива от типа на структурата се дефинира по същия начин, както и обикновена променлива. Дефинирането на структурата може да се разглежда като създаване на **нов** **тип** данни, дефиниран от потребителя.

Колко **памет** се заделя за такава променлива? 3 х 8B, за всяка от член-данните от тип double.

Размерът на структурата може да не е колкото сумата на полетата в нея. (Подравняване...)

### Основни операции

Как осъществяваме **достъп** до член-данните на променлива от типа на структурата?

Достъпът до член-данните на променлива от типа на структурата се осъществява чрез оператора **.** (*member access operator*).

***<Името на променливата>.<името на член-данната>***

|  |
| --- |
| // инициализиране на член-данните  ABC.a = 5;  ABC.b = 5;  ABC.c = 5; |

Друг начин за инициализиране на променлива от типа на структурата е да използваме следния синтаксис:

|  |
| --- |
| Triangle XYZ = {5, 6, 7}; |

Във фигурните скоби се изброяват толкова стойности, разделени със скоби, колкото са член-данните на структурата, разделени със запетаи.

Стойности на променливата могат да се дадат и чрез оператора за присвояване:

|  |
| --- |
| XYZ = ABC; |

Всяка член-данна на променливата XYZ ще приеме стойността на съответната й член-данна от променливата ABC. Променливите ABC и XYZ са от **един** **и същ** структурен тип.

## Функции и структури.

Какви начини за предаване на параметър на функция познаваме?

* Предаване на параметър **по стойност**;

Когато дадена структура се предава като параметър на функция **по стойност**, за да се свърже формалния с действителния параметър се прави **побитово** **копиране** на член-данните на структурата. Ако структурата има член-данна масив, ще се създаде пълно копие на този масив.

* Предаване на параметър **по указател**;

Когато структурата се предава по указател, подаваме адреса на структурната промелива. В рамките на функцията член-данните на структурата се достъпват с оператора ->.

|  |
| --- |
| // Извеждане на триъгълник  void printTriangle(const Triangle\* tr)  {  cout << "Триъгълник със страни (" << tr->a << ", "  << tr->b << ", " << tr->c << ")\n";  }  // обръщение към функцията  printTriangle(&ABC); |

* Предаване на параметър **по** **псевдоним**.

Псевдонимът дава друго име на вече съществуваща променлива. Достъпът до член-данните на структурната променлива се осъществява с оператора <**.**>.

Възможно ли е структура да бъде върната като резултат от функция?

* Ако резултатът от изпълнението на дадена функция е структура, то...
* Конструктор за копиране...
* Псевдоним или указател, но не към структурна променлива дефинирана в рамките на функцията, защото след приключването на функцията, тази структурна променлива вече няма да съществува...

## Структури и масиви

В следващата задача ще бъдат разгледани възможностите за деклариране на масив от структурни променливи.

#### Задача 2

Да се дефинира функция, която намира разстоянието между две точки в равнината. Като се използва тази функция, да се напише програма, която въвежда координатите на n точки в равнината (2 <= n <= 100), намира и извежда най-голямото разстояние между тях.

Да се дефинира подходяща структура, която представя точка в равнината с декартови координати.

**Какви особености срещаме, когато структурите съдържат член-данни масиви?**

Нека разгледаме структура *Student*, която съдържа следните член-данни: факултетен номер (статичен масив от символи, максимална дължина 8), име (статичен масив от символи, максимална дължина 40), специалност, курс. Ако променлива от тип *Student* се предава на функция като параметър по **стойност**, то на статичните масиви *факултетен* *номер* и *име* ще се създадат копия. Друг е въпросът, ако член-данните не са статични масиви, а указатели към динамична памет. При предаване на такъв параметър по стойност, ще се създадат копия на указателите, които обаче ще сочат същата памет, т.е. има поделяне на памет. Има механизми, които разрешават този проблем и те ще бъдат разгледани *скоро*.

## Вложени структури. Рекурсивно използване на структури.

В дефиницията на структура няма ограничение за типа на нейните член-данни, с едно изключение. В това число член-данните на дадена структура могат да бъдат също структури. Такива структури се наричат **вложени**. *С и С++ не допускат член-данна на дадена структура да бъде структура от същия тип*. Защо? Когато описваме член-данните на дадена структура, дефиницията на тази структура все още не е завършена. Следователно не може да се дефинират променливи от типа на структурата. Това ограничение може да се заобиколи с помощта на указател към структурата, тъй като указателят не дефинира променлива. В такива случаи се говори за **рекурсивно използване на** **структури**.

#### Задача 3

Да се дефинират структурите:

* **Person**, определяща лице по собствено и фамилно име;
* **Client**, определяща клиент като лице, притежаващо банкова сметка с дадена сума.

Да се дефинират функции, които въвеждат и извеждат данни за лице и клиент. Да се напише програма, която:

1. въвежда имената и банковите сметки на множество от клиенти, зададено чрез едномерен масив;
2. извежда имената и банковите сметки на клиентите от множеството;
3. намира сумата от задълженията на клиентите от множеството.

Забележка: задълженията на даден клиент се отбелязват с отрицателен баланс на бамковата сметка.

**Как достъпваме член-данните на вложени структури?**

Последователно прилагаме операторът **.** или->,редът на изпълнение на операторите е отляво надясно.

|  |
| --- |
| Client client;  client.name.firstName = "Иван"; |

#### Задача 4

Да се дефинира структура, която описва дължина (разстояние), като използва английските мерни единици фут и инч. Да се дефинират функции, които въвеждат и извеждат дължината. Да се дефинира функция, която събира две дължини.

Да се създаде структура, която описва размерите на правоъгълна стая (ширина и дължина), като се използва структурата *Дължина*. Да се дефинират функции, които въвеждат и извеждат размерите на стая. Да се напише програма, която пресмята площта на дадена стая (мерната единица е кв.фут).

Забележка:

1 фут = 12 инча; 1 инч = 1/12 фут;

**Възможен ли е следния запис?**

|  |
| --- |
| Distance lhs;  readDistance(lhs);  Distance rhs;  readDistance(rhs);  ***Distance sumDist = lhs + rhs;***  cout << "The sum of the two distances is:\n\t";  printDistance(sumDist); |

Операторът + не може да се използва с променливи от типа на структурата Distance. Това е операция на С++, която може да бъде приложена върху вградените типове данни. След като разгледаме класовете, ще се спрем подробно на възможностите за предефиниране на оператори.

**Битови полета**

За какво можем да използваме БИТОВИте полета в рамките на структура? Примери?

Ако искаме да ограничим паметта, която се използва за дадено поле... Например, ако се опитваме да реализираме структура Person, в която да има възраст. Възрастта може да се запише като трицифрено число. Колко бита са необходими, за да се запише число <= 128? 7?

Ако има още данни от тип int, те ще допълнят останалите битове до 32. Ако се дефинира друго поле от различен тип, то ще започне от следващата дума.

Ако структурата е Student, възрастта няма да надвишава да кажем 32, следователно можем да се ограничим до 5 бита.

Пример:

|  |
| --- |
| struct Date  {  unsigned weekDay : 3; // 0..7 (3 бита)  unsigned monthDay : 6; // 0..31 (6 бита)  unsigned month : 5; // 0..12 (5 бита)  unsigned year : 8; // 0..100 (8 бита)  }; |

Като тип на данните може да се използва int, unsigned, long или изброен тип.

Да се сложат картинки за разпределението на памметта и

## От структури към класове

Структурите обикновено се използват само за съхранение на данни. В действителност в С++ структурите, освен данни, могат да съдържат и функции. (За разлика от С, където структурата съдържа само данни.) Синтактичната разлика между структури и класове в C++ е минилна, така че на теория те са почти взаимно заменяеми. НО голяма част от програмистите използват структурите САМО за съхрание на ДАННИ, докато класовете са тези, които съхраняват и ДАННИ и ФУНКЦИИ за тяхната обработка.

Спецификатори за достъп? В рамките на структурата член-данните обикновено са с публичен достъп, докато в класовете член-данните са скрити (private).

## Изброен тип

Кога можем да използваме изброеният тип? Когато предвартелно знаем, че този тип данни има КРАЙНО множество от стойности (цели числа). С дефинирането и използването на изброения тип програмата значително може да се опрости, а и значително се улеснява четенето на кода.

### Как декларираме изброен тип?

|  |
| --- |
| enum DAYS  {  MON, TUE, WED, THU, FRI, SAT, SUN  };  int main()  {  DAYS day = WED;  cout << day;  return 0;  } |

Използваме ключовата дума **enum** последвана от името на изброения тип. Във фигурни скоби се изброяват допустимите стойности, разделени със запетая. Декларацията завършва с ;. Вътрешно изброеният тип се разглежда като int. Първата стойност от списъка се свързва с 0 по подразбиране, ако не е зададена друга стойност. Към следващите стойности се добавя 1. В примера MON = 0, TUE = 1 и така последователно до SUN = 6.

Променливи от изброения тип могат да се свързват само със стойностите, които са изброени в декларацията на типа. Всеки опит на променлива от изброен тип да бъде присвоено нещо различно от изброените стойности, ще доведе до съобщение за грешка.

|  |
| --- |
| DAYS day1 = 10; // invalid convertion from ‘int’ to ‘DAYS’ |

След като вътрешното представяне на стойностите в изброения тип е int, върху тях могат да се прилагат аритметични операции, както и операции за сравнение.

|  |
| --- |
| DAYS day1 = WED;  DAYS day2 = FRI;  cout << "Разликата е " << day2 - day1 << " дни\n";  if(day2 > day1)  {  cout << "day2 e след day1\n";  } |

Изходът от следният код ще бъде:

|  |
| --- |
| Разликата е 2 дни  day2 е след day1 |

Основният недостатък на изброения тип е, че те не се разпознават от операторите за вход >> и изход <<. Входно-изходните оператори в C++ третират изброеният тип като int, така че изходът ще бъде цяло число. За да се изведе текст, който да съответства на стойността, можем да използваме switch.

|  |
| --- |
| void printDay(DAYS day)  {  switch (day)  {  case MON:  cout << "MONDAY\n"; break;  case TUE:  cout << "TUESDAY\n"; break;  case WED:  cout << "WEDNESDAY\n"; break;  case THU:  cout << "THURSDAY\n"; break;  case FRI:  cout << "FRIDAY\n"; break;  case SAT:  cout << "SATURDAY\n"; break;  case SUN:  cout << "SUNDAY\n"; break;  default: cout << "Невалидена стойност за ден!\n";  }  } |

Можем явно да свържем стойностите в изброения тип с определено цяло число като запишем THU = 4. В този случай FRI автоматично се свързва с 5 и т.н.. Всички стойности са MON = 0, TUE = 1, WED = 2, THU = 4, FRI = 5, SAT = 6 и SUN = 7.

### Задача за домашна работа:

Да се напише програма, която реализира следната игра на карти. Единият от играчите показва на другия три карти, след което ги поставя пред себе си с лицето надолу и ги размества неколкократно. Вторият играч трябва да познае къде се намира определена карта. (Реално това се извършва толкова бързо, че вторият играч почти винаги изгубва позицията на картата и съответно парите си.)

За целта трябва да се дефинира подходяща структура, която описва карта за игра.

***Упътване***: Всяка карта се характеризира с боя и сила. Имаме 4 основни бои (цвята): пика, купа, каро и спатия. Всяка боя се състои от 13 карти, подредени в следния ред: 2, 3, 4, 5, 6, 7, 8, 9, 10, вале, дама, поп и асо.

Включено е просто примерно решение. Как може да бъде подобрено решението? Къде и по какъв начин може да се използва изброен тип? Какви са предимствата от използването на изброен тип?

Ако например се използва изброен тип за боята, знаем точно кои са разрешените стойности, с които можем да инициализираме картата. Всичко извън тези стойности ще даде предупреждение от компилатора.

## Обединения