Тема 1

Структури и обединения. Представяне в паметта. Размер на обекти. Подравняване. Подаване на обекти/инстанции във функции. Създаване на обекти в динамичната памет

1. Структури

Структурите в C++ се използват за групиране на елементи. Елементите, наричани още членове могат да бъдат от различен тип и с различна дължина. Структурата има характеристики, които са вече готови типове данни. Досега сме използвали само примитивни типове данни - int, bool, char, double, но благодарение на структурите и обединенията можем да създаваме нови типове данни с желаните от нас характеристики и функционалности

Пример:

```
struct Box
{
  double height; //   Когато създаваме обект
  double width; //    от структурата, той се
  double legth; //    нарича инстанция
};
```

В паметта член-данните на инстанцията се намират на отделни места. Член-данните са разположени последователно една след друга според реда, в който са декларирани в структурата и всяка от променливите е на адрес кратен на големината ѝ. Промяната на една от член-данните не се отразява върху останалите.

Размерът на инстанцията е не равен на сумата от характеристиките ѝ.

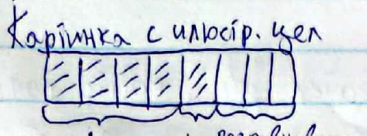
$$sizeof(struct) \neq \sum_{x \in struct} sizeof(x)$$

Компилаторите използват процеса на подравняването

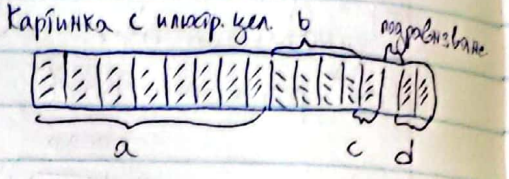
Правила на подравняването:

- всички променливи се разполагат последователно в паметта, в реда в който се декл
- големината на структурата трябва да се дели на размера на най-голямата примитивна член-данна.
- всяка член-променлива трябва да е на адрес, който се дели на големината ѝ, започвайки от адрес 0 в началото на структурата.

Пример:
struct A
{
int i;
bool b;
};
sizeof(A) = 8

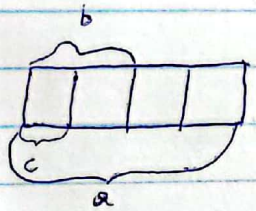


Пример:
struct B
{
double a;
int b;
bool c;
short d;
};
sizeof(B) = 16



За постигане на минимална памет е най-добре да сортираме член.данните в низходящ ред по големината им.

2. Обединения
Обединението като идея е подобна на структурата - използва се за групиране на елементи, но на доста различен принцип. В обединението всички променливи споделят обща памет, те могат да са от различни типове, но във всеки един момент може да се достъпва само една променлива от състава на обединението. Всяка една промяна върху някоя от член-данните се отразява и на останалите от състава на обединението, защото адресът на променливите в обединението е един и същ.
Пример:
union MyUnion
{
int a;
short b;
char c;
};



sizeof(MyUnion) = 4

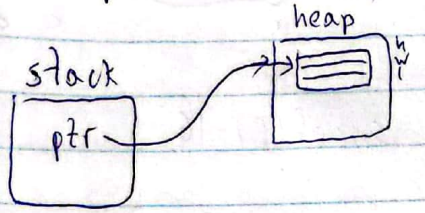
Големината на обединението е равна на големината
променливата с най-голям размер
sizeof(union) = max(sizeof(x)), x ∈ union

3. Създаване на обекти
а) статичен обект
Пример:
Box b = {4, 5.4, 10}; <=> b.height = 4; b.width = 5.4; b.length = 10;
height width length
Ако се изпусне някое поле от инициализиращия списък, то ще бъде произвольна стойност.

Присвояването на стойност на член-данните на обектите се извършва отгоре-надолу по реда, в който са декларирани.

б) динамичен обект.

Box* ptr = new Box(); - създаваме указател "ptr", който сочи към променлива от тип Box в динамичната памет.



Достъпването на елементи може да се случи по два начина:

I н. (*ptr).height = 3; (чрез дереферинг)

II н. ptr->height = 3;

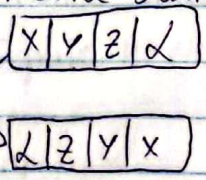
Указателят може да има стойност "nullptr", затова винаги, когато работим с указатели трябва да проверим дали е nullptr. За разлика от указателите, при референциите си гарантираме, че винаги има обект, защото празна референция няма.

Обектите могат да се подават във функции. Препоръчително е да се подават по референция, за да се избегнат излишни копирания. Когато няма да се правят промени по обекта в рамките на функцията, задължително го подаваме по константна референция

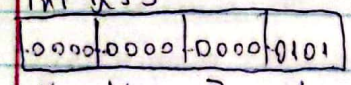
Пример:

```
double calculateVolume(const Box& b)
{
  return b.height * b.width * b.length;
}
```

(*) В паметта байтовете се записват огледално



int a = 5



В паметта байтовете се записват наобратно (a z y x)