Хранение объектов в памяти — конспект темы

От перемены мест слагаемых сумма...

Структура автомобильного номера:

```
struct Plate {
    char c1;
    int num;
    char c2;
    char c3;
    int region;
};
```

Её размер:

`char c1` — 1 байт,

`int num` — 4 байта,

`char c2` — 1 байт,

`char c3` — 1 байт,

`int region` — 4 байта.

Итого: 11 байт.

Используем программные средства, чтобы проверить себя:

Но макрос `sizeof`, который возвращает размер объекта или типа данных, переданного в качестве параметра, укажет размер 16 байт.

```
int main() {
    cout << sizeof(Plate) << endl;
}
```

Все поля в этой структуре должны лежать по «коробочкам» из 4 байт. `c1` займёт не 1 байт, а 4. `c2` и `c3` поместятся в одну «коробочку», но и там возникнут пустые 2 байта. Итого 11 байт для данных + 5 байт пустых.

Макрос `offsetof` позволяет узнать, какое смещение в байтах имеет поле относительно начала объекта:

```
int main() {
    cout << "Sizeof = "s << sizeof(Plate) << endl;
    cout << offsetof(Plate, c1) << endl;
    cout << offsetof(Plate, num) << endl;
    cout << offsetof(Plate, c2) << endl;
    cout << offsetof(Plate, c3) << endl;
    cout << offsetof(Plate, region) << endl;
}
```

Программа напечатает:

```
Sizeof = 16

0
4
8
9
12
```

Выравнивание облегчает процессору поиск. Процессор читает из памяти «словами» по 4 байта. Если половина числа лежит в первых 4 байтах, а вторая половина — во вторых, процессору нужно будет прочитать из памяти дважды, а потом ещё и склеить число из двух половинок.

Правила выравнивания

1. Выравнивание типа по его размеру

Тип `int` выравнивается по байтам, кратным 4. Тип `double` — по байтам, кратным 8. Тип `char` занимает всего 1 байт, поэтому не выравнивается никаким специальным образом.

Байты, оставшиеся неиспользованными, но нужные процессору для выравнивания, называются **padding**.

2. Массивы

Когда структура содержит массив данных, каждый элемент массива представляет собой отдельный элемент. Данные массива располагаются в массиве

последовательно.

3. padding в конце структуры

Если последняя коробочка заполнена не до конца, незаполненные байты остаются частью структуры и учитываются в общем размере. Очевидным это правило становится, когда структуры помещаются в массив.

Выравнивание при наследовании

Макрос `offsetof` невозможно применить к полям базового класса. В таких случаях используйте компилятор clang и специальные флаги компиляции, позволяющие увидеть, как устроены структуры внутри кода:

```
clang -Xclang -fdump-record-layouts plate.cpp
```

Запустив компиляцию с этим флагом, увидите тонны информации. Она нужна не вся. Можно просто автоматическим поиском по тексту найти слово `Outer`.

Плотная упаковка

Структура `Plate` хранит автомобильные номера:

```
struct Plate {
    char c1;
    int num;
    char c2;
    char c3;
    int region;
};
```

Чтобы сэкономить память, которую занимает `Plate`, можно:

- переупорядочить поля. Пусть поля `num` и `region` идут в начале структуры, а однобайтовые `c1`, `c2` и `c3` — в конце;
- использовать для `region` и `num` тип `uint16_t` — для представления чисел от 0 до 1000 достаточно 2 байт;
- использовать директиву `pragma pack` — она сообщает процессору, что выравнивания у структуры быть не должно:

```
#pragma pack(push, 1)
struct Plate {
    uint16_t num;
    uint16_t region;
    char c1;
    char c2;
    char c3;
};
#pragma pack(pop)
```

- определить поля `num` и `region` как битовые:

```
#pragma pack(push, 1)
struct Plate {
    uint16_t num:10;
    uint16_t region:10;
    char c1;
    char c2;
    char c3;
};
#pragma pack(pop)
```

- создать массив символов и вместо самих символов в структуре хранить индексы в этом массиве — для хранения чисел от 0 до 12 будет достаточно 4 бит:

```
const static int N = 12;
const std::array<char, N> Letters = {
    'A', 'B', 'E', 'K', 'M', 'H', 'O', 'P', 'C', 'T', 'Y', 'X'};

#pragma pack(push, 1)
struct Plate {
    uint16_t num:10;
    uint16_t region:10;
    uint8_t c1:4;
    uint8_t c2:4;
    uint8_t c3:4;
};
#pragma pack(pop)
```

- явно сказать компилятору, что вся структура может уместиться в одну переменную типа `uint32_t`:

```
const static int N = 12;
const std::array<char, N> Letters = {
    'A', 'B', 'E', 'K', 'M', 'H', 'O', 'P', 'C', 'T', 'Y', 'X'};

#pragma pack(push, 1)
struct Plate {
    uint32_t num:10;
    uint32_t region:10;
    uint32_t c1:4;
    uint32_t c2:4;
    uint32_t c3:4;
};
#pragma pack(pop)
```

Так размер структуры сократится с 16 байт до 4.