Недостатъци на референциите

- Трябва да се инициализира при декларация
- След като веднъж е била инициализирана, повече не може да се променя

Пойнтър

- Тип данна, която като стойност притежава адрес(клетка)
- Почти всеки пойнтър си има тип и може да притежава адресите само на променливи от същия тип
- Може да съдържа както адреса на някоя lvalue, така и празното пространство (nullptr) или някоя непозволена памет (което е източник на грешки)
- Адресът, който съдържа пойнтърът, може да се променя
- Може да се извършват промени по данните в съответния адрес

Пойнтър - пояснения

- Нарича се пойнтър (pointer), защото все едно сочи към мястото в паметта, чийто адрес съхранява.
- Синтаксис:

<тип> *<име> [= <израз>];

- Пример:

```
int * pointer; //неинициализиран пойнтър
```

Пойнтър – пояснения относно адресите

- Съществува така наречения нулев пойнтър (nullptr), наричан още пойнтъров литерал
- nullptr сочи към адрес “0x00000000“ и има специални свойства, с които ще се сблъскате в курсовете по ООП и СДА

- Оператор & има и още 1 приложение:
  - Когато се използва като префикс, връща адреса на дадената променлива
  - Пример:

```
int a = 5;
int * b = &a;
```

Пойнтър – пояснения относно адресите

- Един пойнтър може да сочи към друг пойнтър (двоен пойнтър)

```
int a = 5;
int * b = &a; //сочи към адреса на a
int **c = &b; //сочи към адреса на b
```

- Тъй като пойнтърът е обект, той също притежава адрес!
- Тази концепция отнема време да се разбере, но точно тя е разликата между занаятчията и програмиста

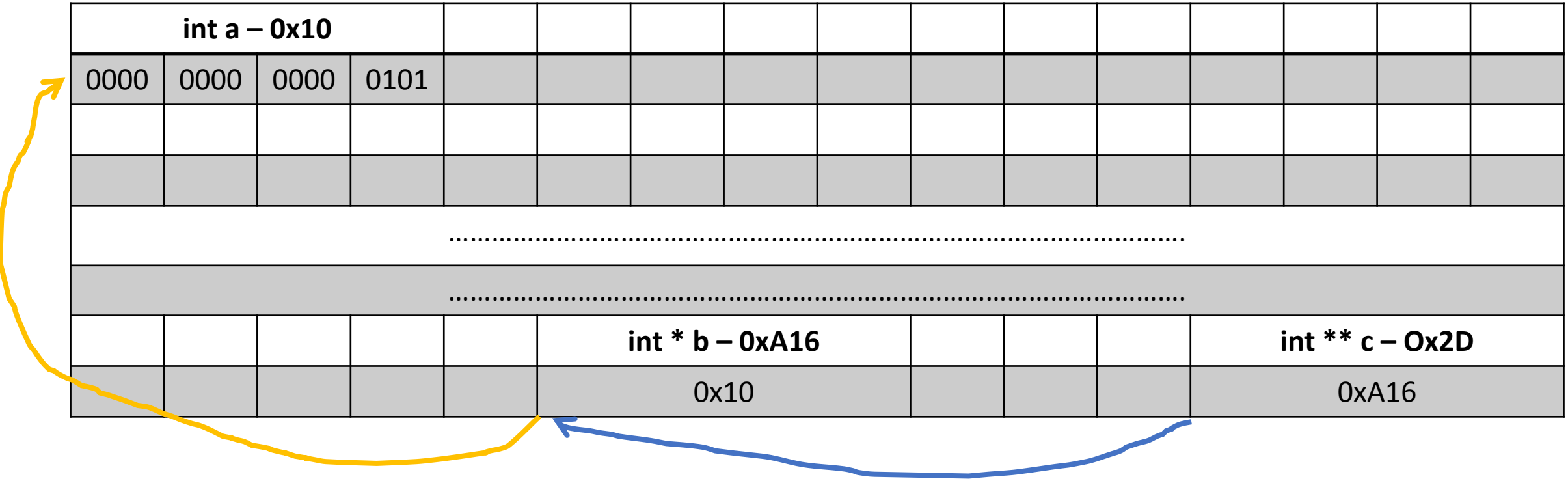
Пойнтър – пояснения относно адресите

```
int a = 5;                  //променлива от тип int
int * b = &a;               //пойнтър към променлива от тип int
(int *)*c = &b;             //пойнтър към пойнтър от тип int
```

- Важно е да се отбележи, че с няма пряка връзка с a
- с единствено съдържа адреса на b и не знае какво се съдържа в него (но може да научи)

Пример

```
int a = 5;
int * b = &a; //сочи към адреса на a
int **c = &b; //сочи към адреса на b
```



Не се предавайте!

- Само още малко суха теория и ще има много примери ☺

Реферирane и дереференцирane

- &<име> — взимане на адреса на променливата<име> (реферирane)
- *<указател> — влизане в дадена клетка от паметта, (дереференцирane)

- Както казахме, пойнтърът съдържа адреса на дадена променлива
- Като цяло, & е операцията за излизане от клетката, а * е операцията за влизане в клетката, а в нормално състояние

Реферирane и дереференцирane - примери

```
bool * engineer= nullptr; //инженер, който не отговаря за самолет
bool plane1 = true;                 //да кажем, че ако е true, то самолетът е ОК
engineer   = &plane1;               //инженерът вече отговаря за самолет1
bool * gremlin = engineer;          //гремлин, който трябва да тормози самолет1
(*gremlin)--;                       // гремлинът повреждa самолета
*engineer = true;                   //инженерът поправя самолета
```

!Важно реферирane и дереференцирane са 3ти по приоритет в таблицата с приоритети, затова трябва да се укаже със скоби, че първо искаме да влезем в клетката и чак след това да действаме!

Precedence	Operator	Description
1	`::`	Scope resolution

2	`++ --`
	type`()` *type*`{}`
	`()`
	`[]`
	`.`
	`->`
		Suffix/postfix increment and decrement
		Function-style type cast
		Function call
		Array subscripting
		Element selection by reference
		Element selection through pointer

	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
		Prefix increment and decrement
		Unary plus and minus
		Logical NOT and bitwise NOT
		C-style type cast

3	`*` `&`	Indirection (dereference) Address-of

	`sizeof`
	`new, new[]`
	`delete, delete[]`
		Size-of
		Dynamic memory allocation
		Dynamic memory deallocation

4	`.* ->*`	Pointer to member
5	`* / %`	Multiplication, division, and remainder
6	`+ -`	Addition and subtraction
7	`<< >>`	Bitwise left shift and right shift
8	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively
9	`== !=`	For relational = and ≠ respectively
10	`&`	Bitwise AND
11	`^`	Bitwise XOR (exclusive or)
12	`|`	Bitwise OR (inclusive or)
13	`&&`	Logical AND
14	`||`	Logical OR

15	`?:`
	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Ternary conditional
		Direct assignment (provided by default for C++ classes)
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR

16	`throw`	Throw operator (for exceptions)
17	`,`	Comma

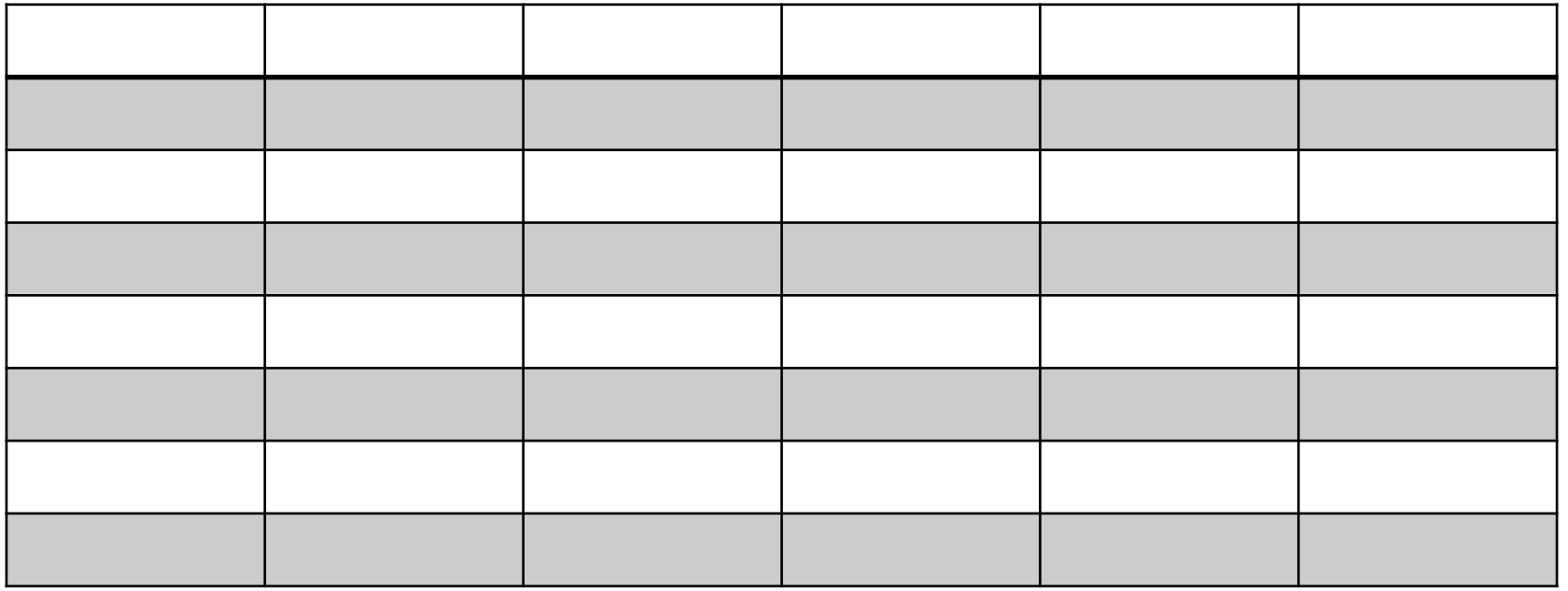
Реферирane и дереференцирane - примери

```
bool plane2 = *engineer; //нов самолет, който е в същото състояние
                         //като този, за който отгаваря инженерът
*gremlin = false;        //гремлинът поврежда самолет 1
 gremlin = &plane2;      //гремлинът отива на самолет 2
*engineer = 1;           //инженерът поправя самолет 1
 *gremlin = !(*engineer); //гремлинът прави обратното на инженера
```

Нека да визуализираме и горния пример

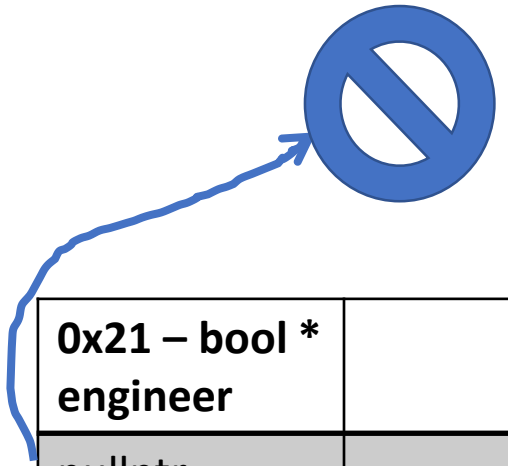
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



Реферирane и дереференцирane - примери

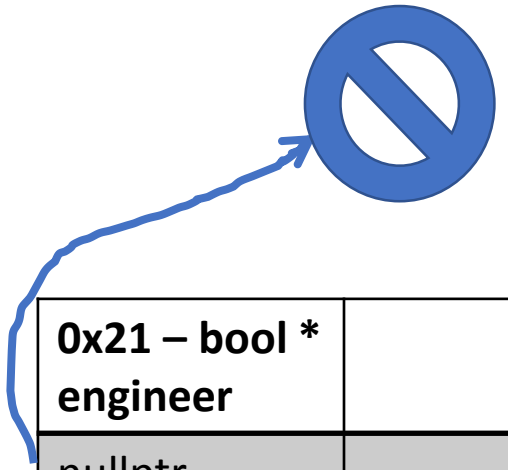
```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



0x21 – bool * engineer					
nullptr					

Реферирание и дереференциране - примери

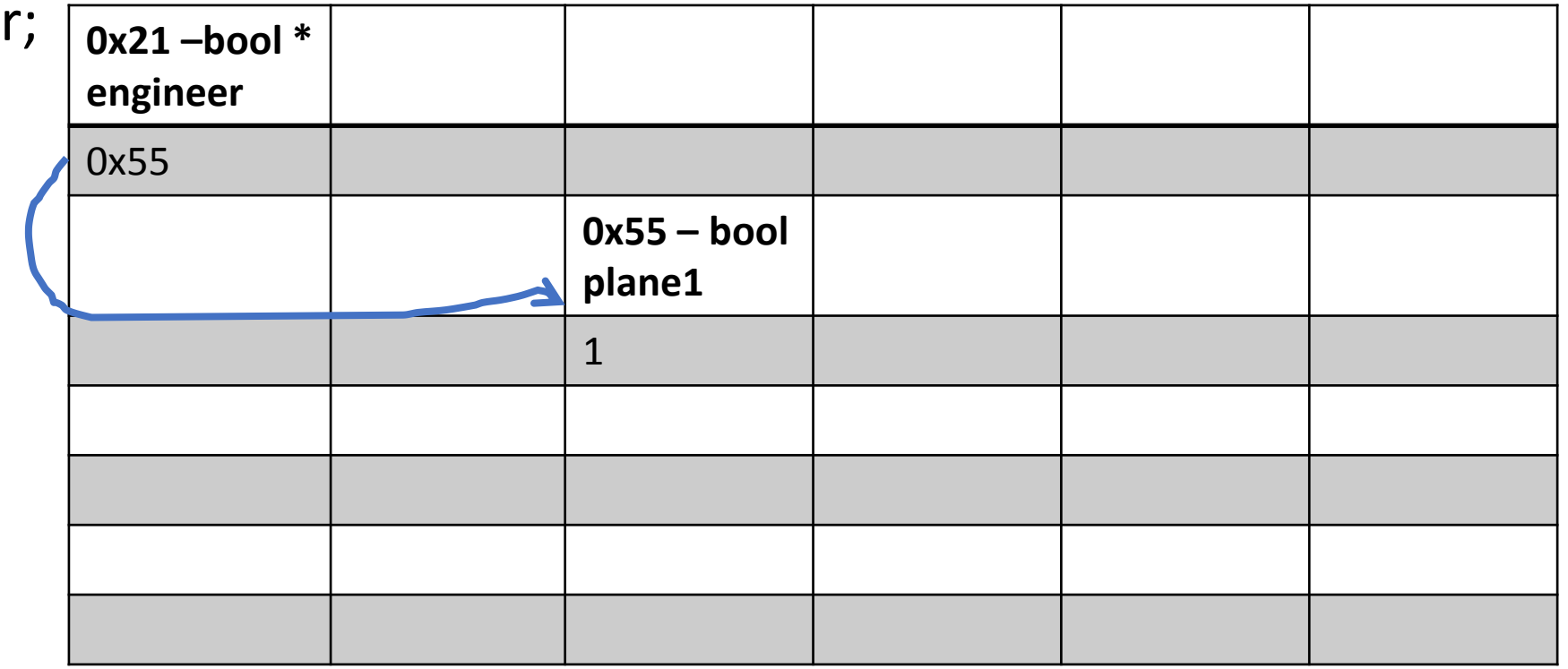
```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



0x21 – bool * engineer					
nullptr					
		0x55 – bool plane1			
		1			

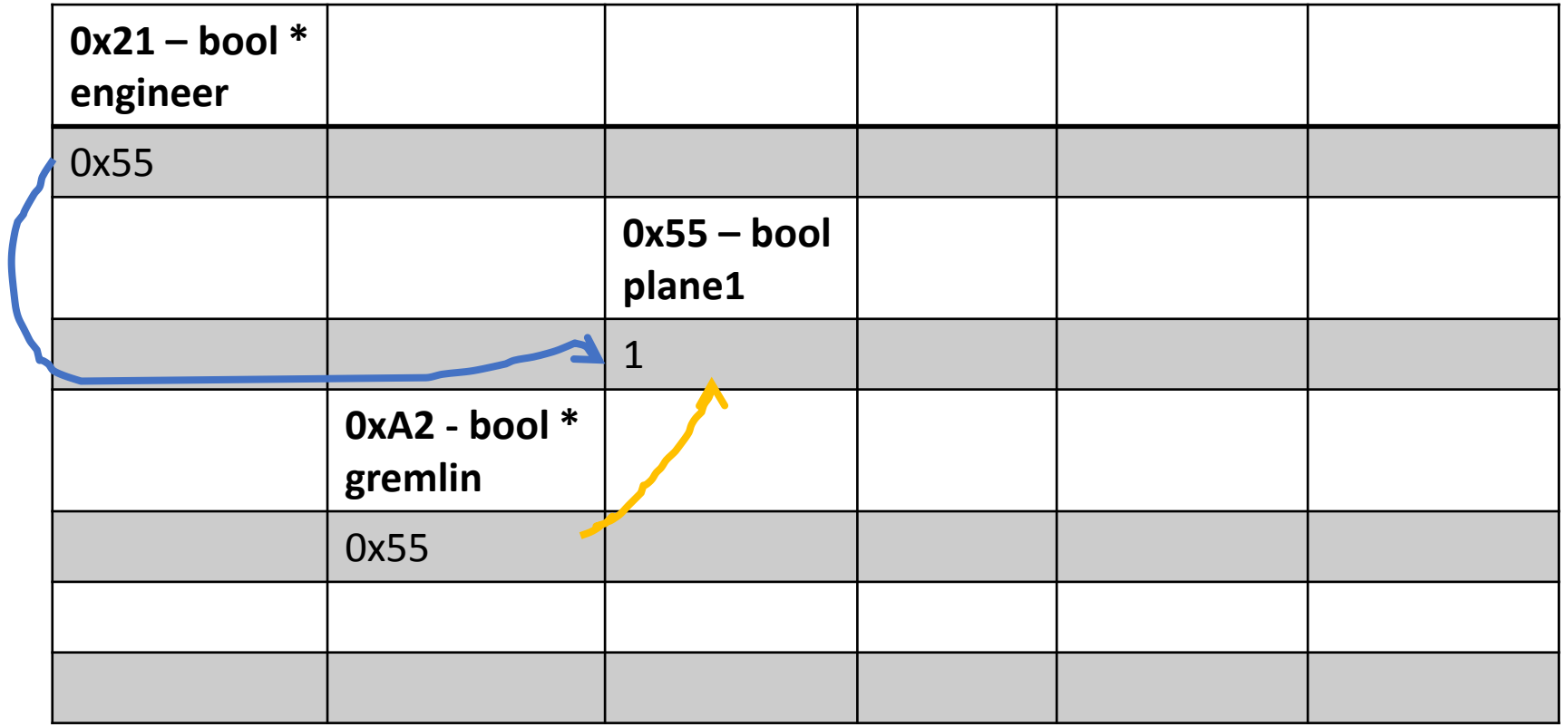
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



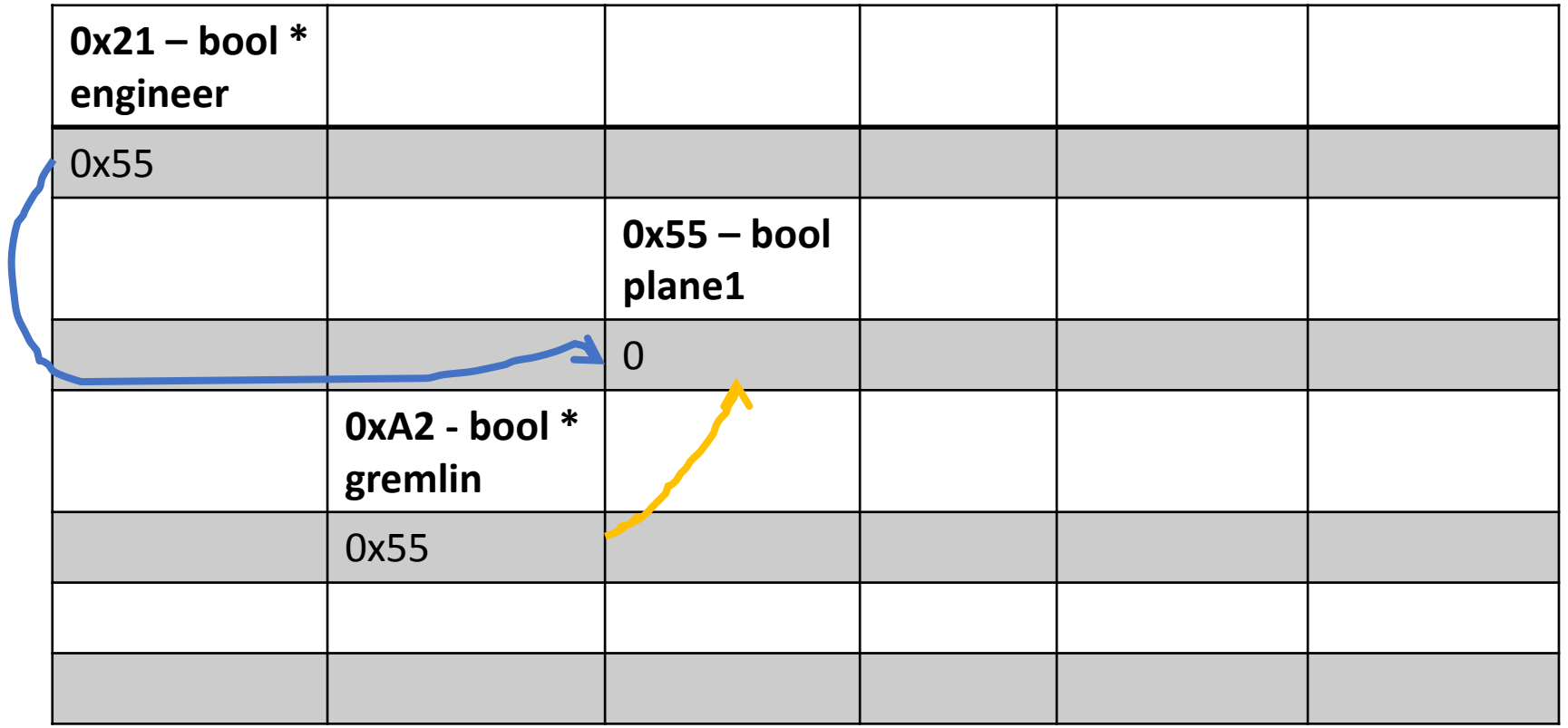
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



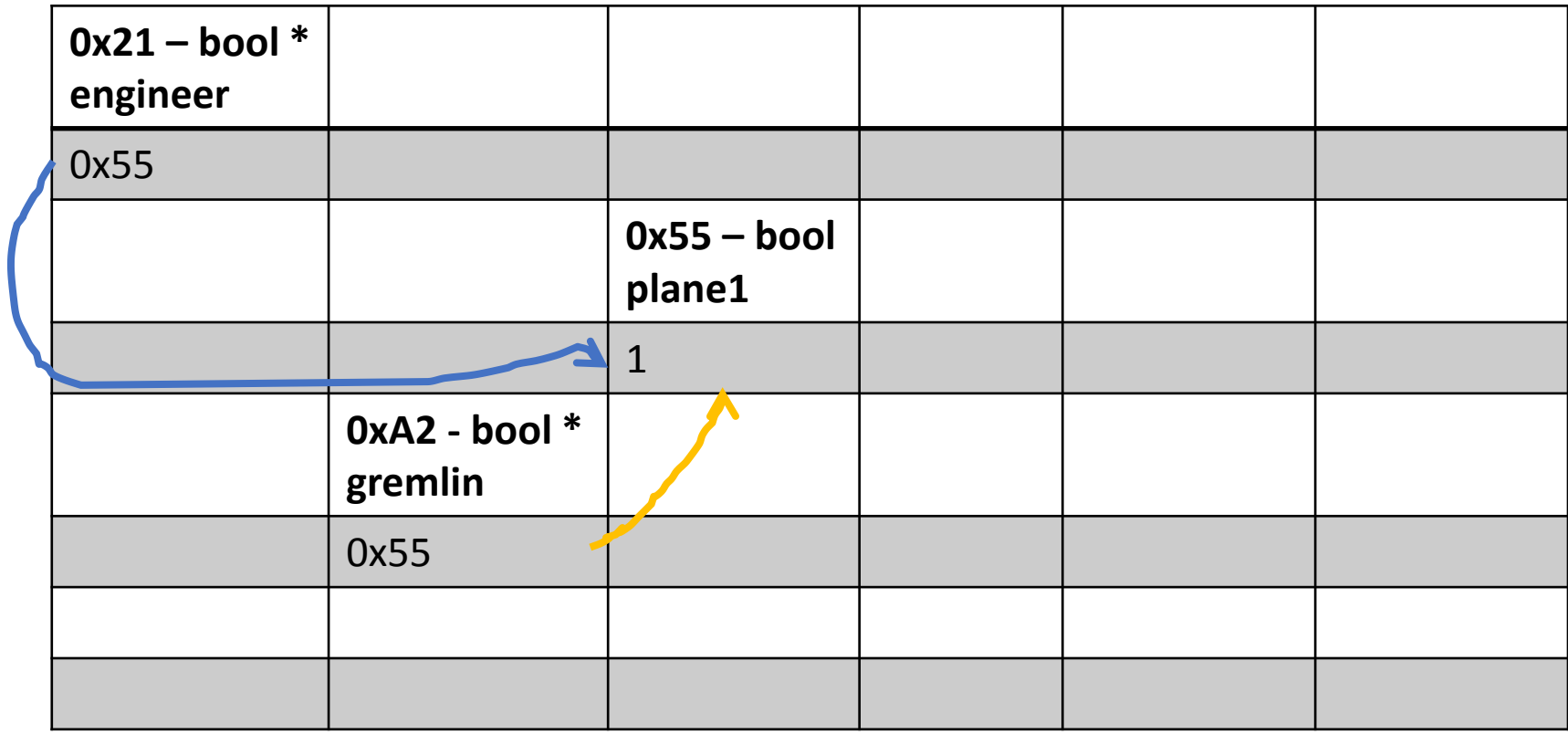
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



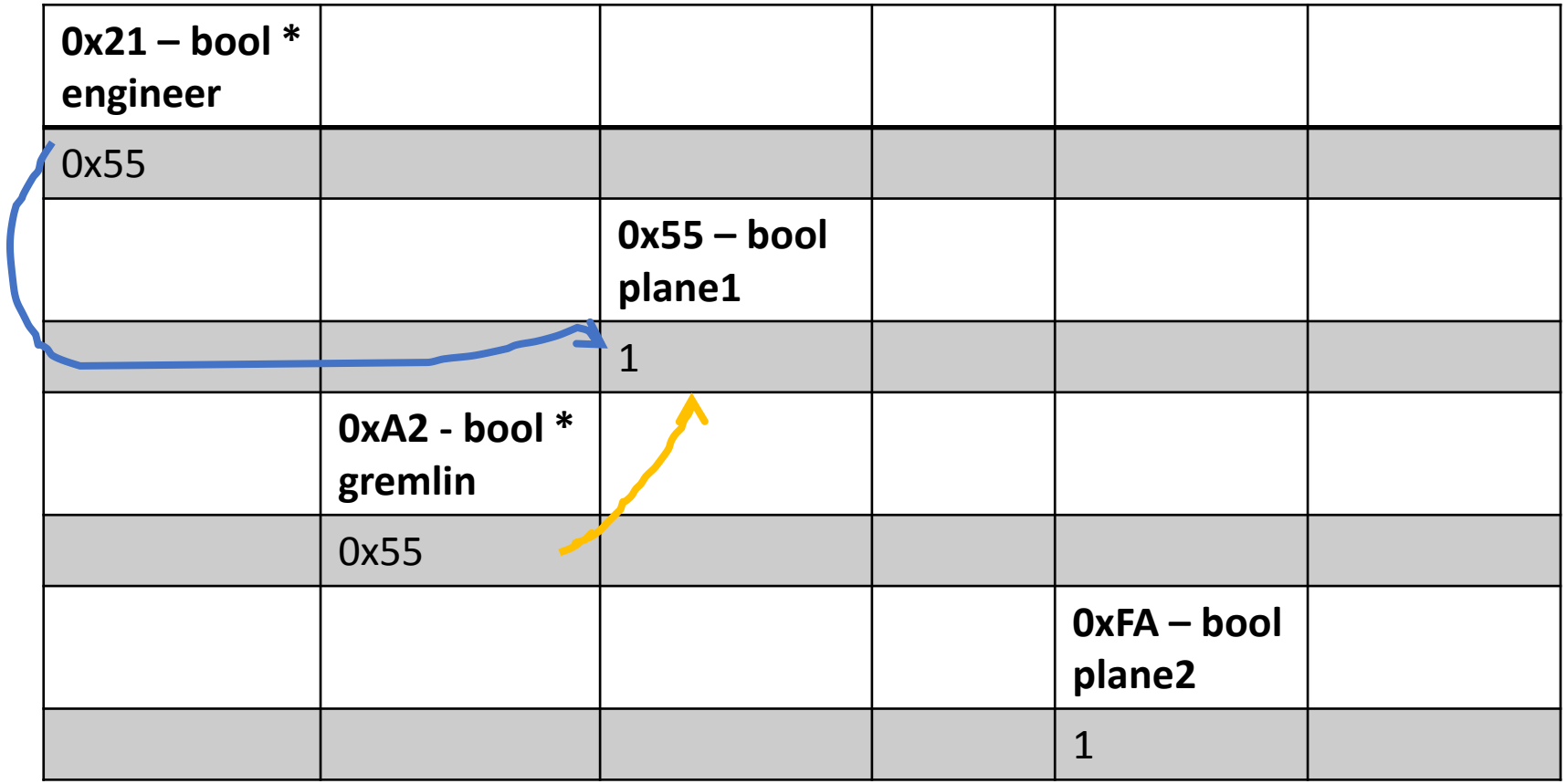
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



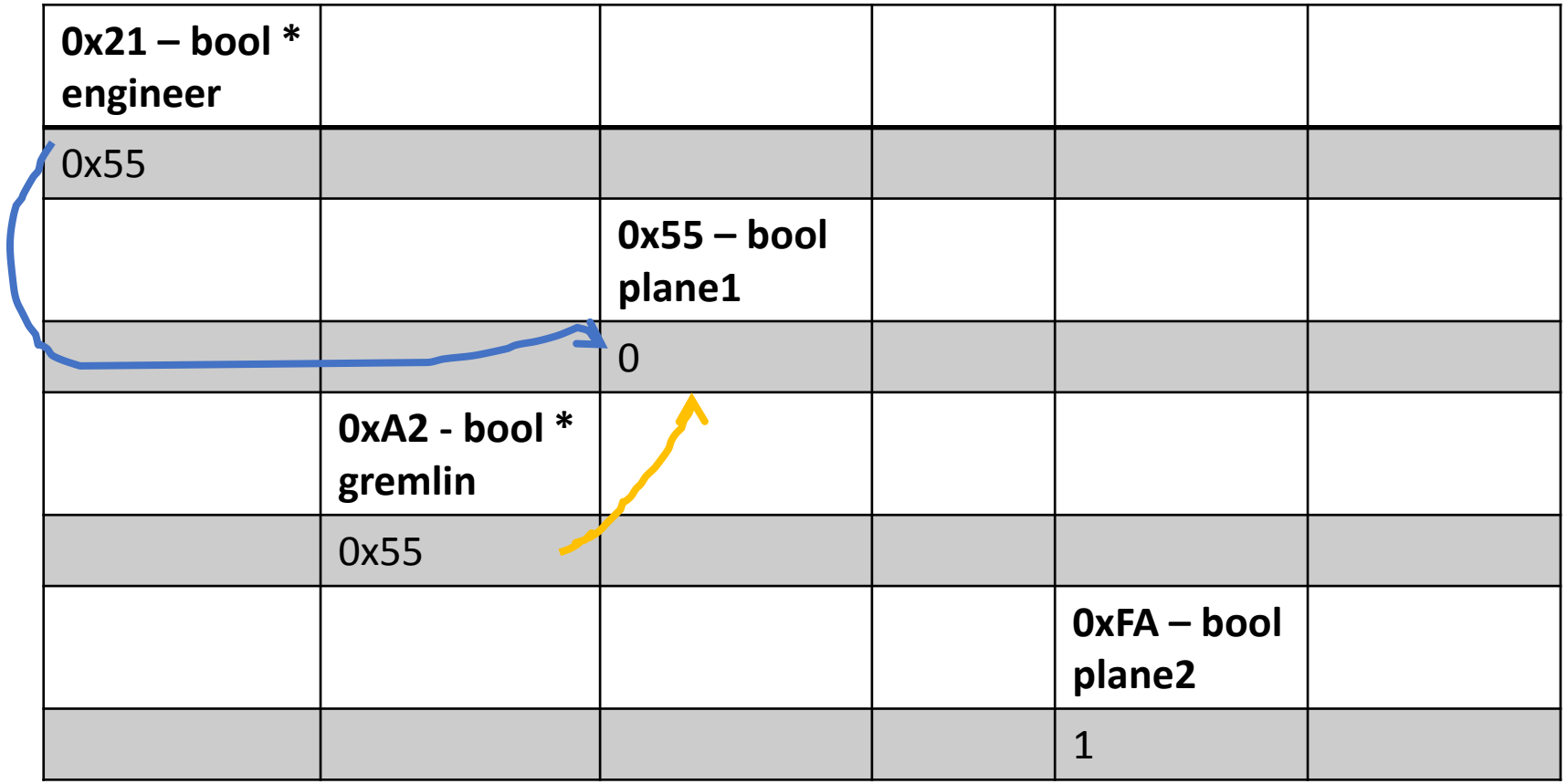
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



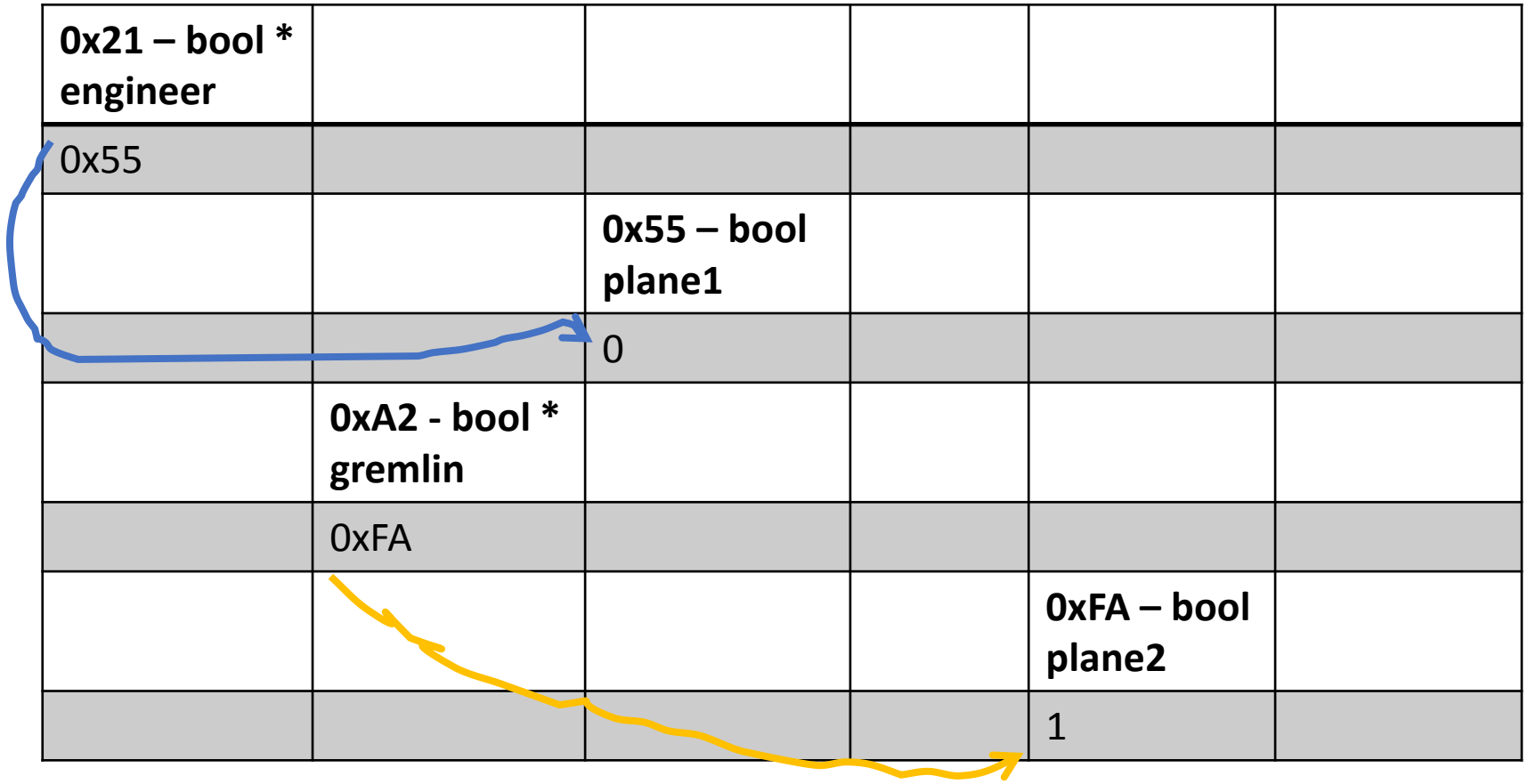
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



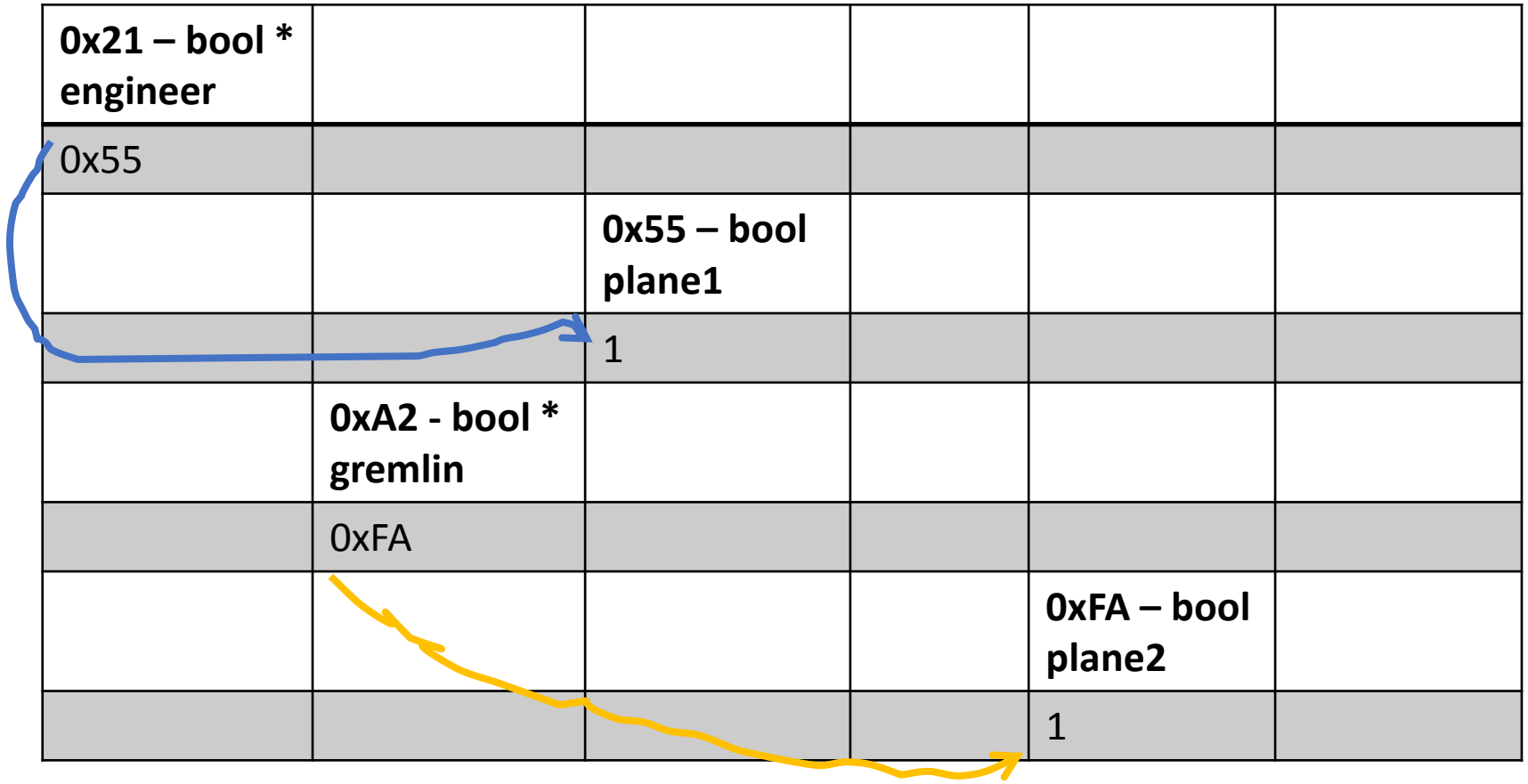
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



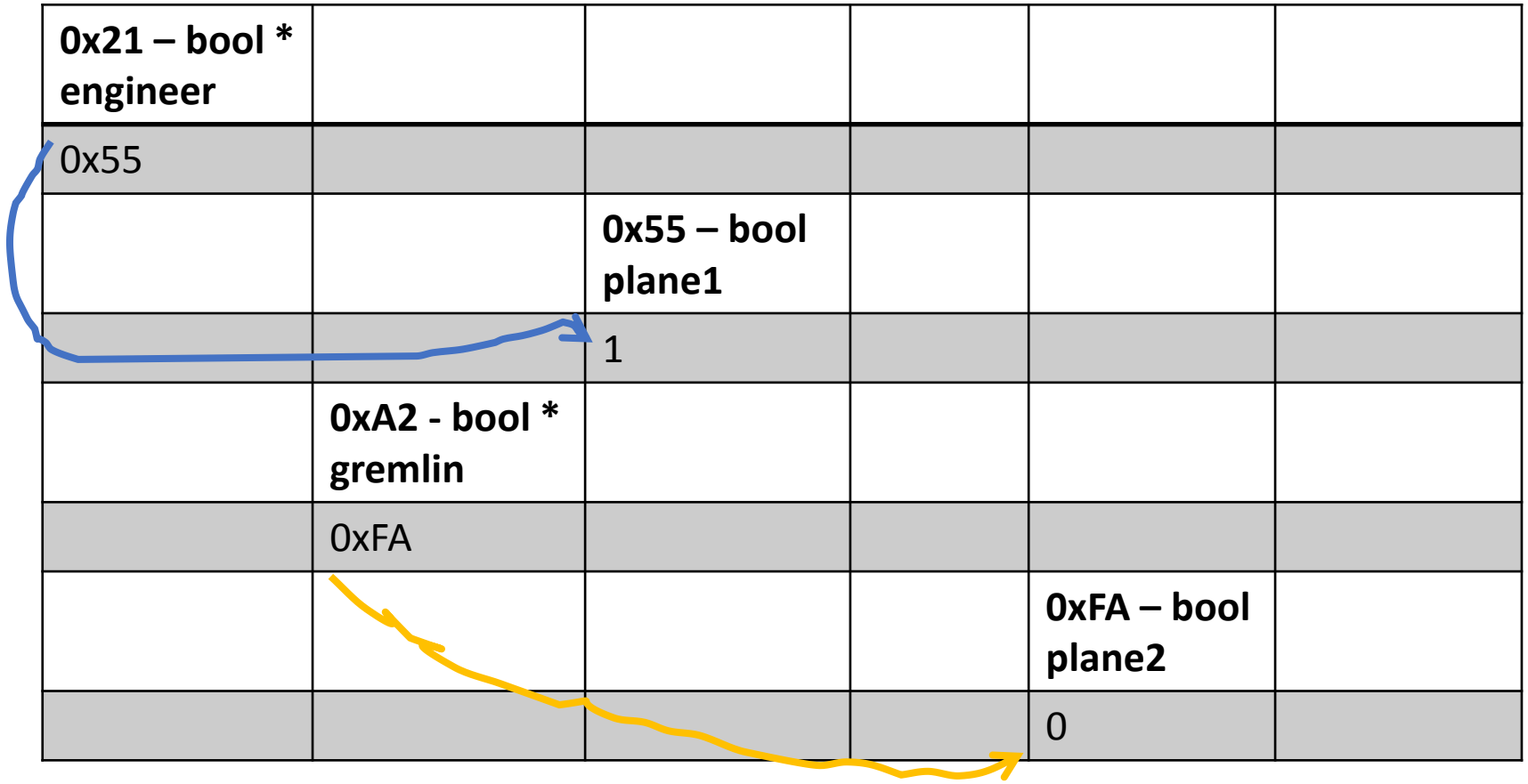
Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



Реферирane и дереференцирane - примери

```
bool * engineer= nullptr;
bool plane1 = true;
engineer = &plane1;
bool * gremlin = engineer;
 (*gremlin)--;
*engineer = true;
bool plane2 = *engineer;
*gremlin = false;
gremlin = &plane2;
*engineer = 1;
*gremlin = !(*engineer);
```



Реферииране и дереференциране

- При реферииране (&) се взима адреса на съответния елемент, затова е задължително да се работи с lvalue (не можем да вземем адреса на константа)
- Адресът съответно е константа и не можем да го променим
- При дереференциране приемаме като параметър някакъв адрес, който е константа
- След като влезем в съответния адрес, имаме достъп до променливата, която е lvalue

Реферирane и дереференцирane

- &<lvalue> връща като резултат <rvalue>!
  - Следните операции са невалидни:
    - &3 – реферирane на константа
    - &x = 1 - присвояване на константа за стойност на референция
- *<rvalue> връща като резултат <lvalue>!

- операциите са дуални една на друга и се унищожават взаимно
  - &(*p) ⇐⇒ p
  - *(&x) ⇐⇒ x

Пойнтъри към константи и константни пойнтъри

- const int * == int const * - пойнтър към константа
- Пойнтърите към константа са същите като обикновените пойнтъри, с тази разлика, че не може да се променя стойността на променливата, към която сочат (дори маниакът да влезе в клетката той не може да променя нищо, а само да казва какво става вътре)

- int * const – константен пойнтър
- Константните пойнтъри са като референциите, но са пойнтъри (маниакът си стои пред клетката, но не може да я сменя с друга)

Други практики, заслужаващи преглед

- const int * const == int const * const – константен пойнтър към константа
- int * * const – константен пойнтър към пойнтър от тип int
- int * const * - пойнтър към константен пойнтър към int
- int const * * - двоен пойнтър към константа от тип int
- int * const * const – константен пойнтър към константен пойнтър към int
- const int * const * const – константен пойнтър към константен пойнтър към константа от тип int

Други практики, заслужаващи преглед

- const int * const == int const * const – константен пойнтър към константа
- (int *) * const – константен пойнтър към пойнтър от тип int
- ((int) * const)* - пойнтър към константен пойнтър към int
- ((int const) *) * - двоен пойнтър към константа от тип int
- ((int) * const) * const – константен пойнтър към константен пойнтър към int
- ((const int) * const) * const – константен пойнтър към константен пойнтър към константа от тип int