

# Добре дошли

- Отидете на [www.menti.com](https://www.menti.com)
- Това е сайт за анкети, който ще използваме активно на тази консултация

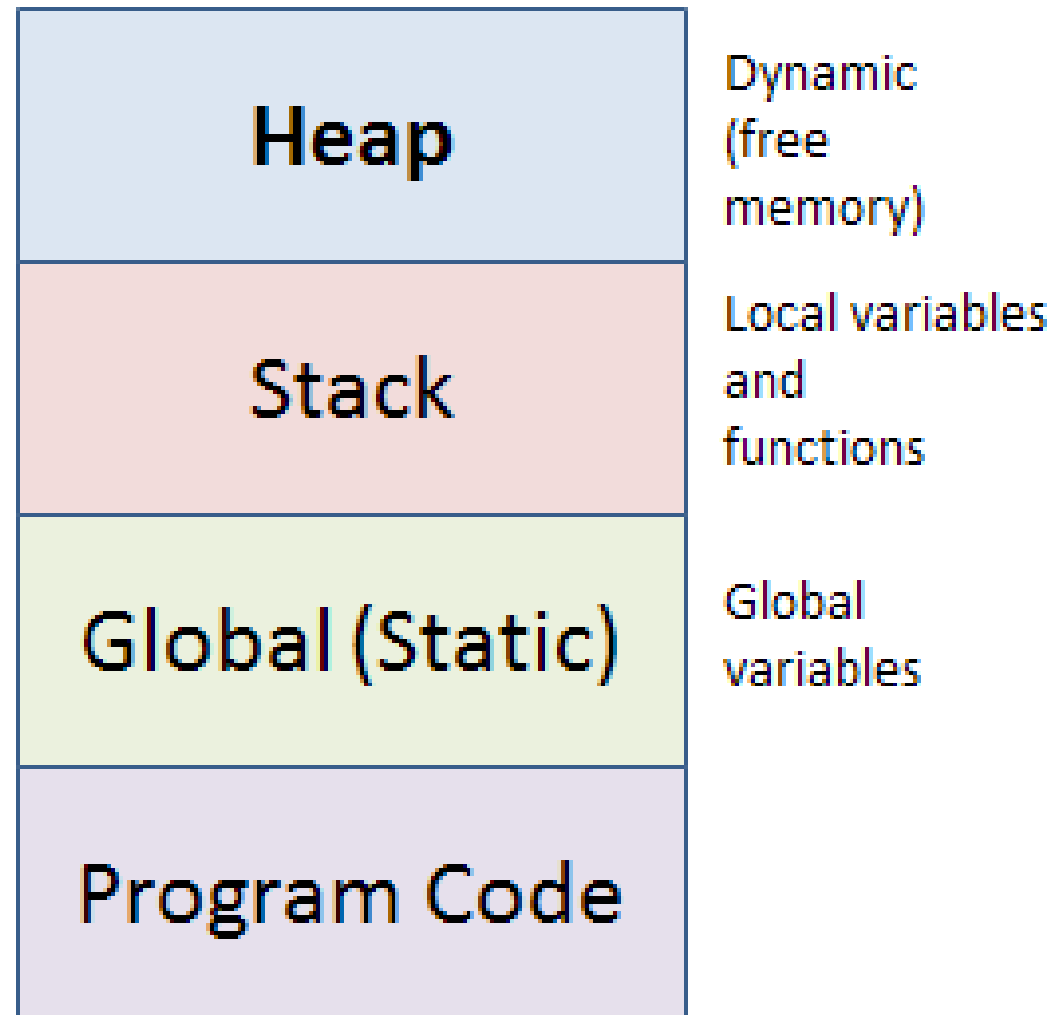
# Консултация по УП за изпит

Изготвена и представена от Мартин Илиев

# Какво покрива тази презентация

- Видове памет
- Работа с динамична памет
- Структури
- Примерни задачи
- Време за въпроси (надявам се да остане такава)

# Видове памет



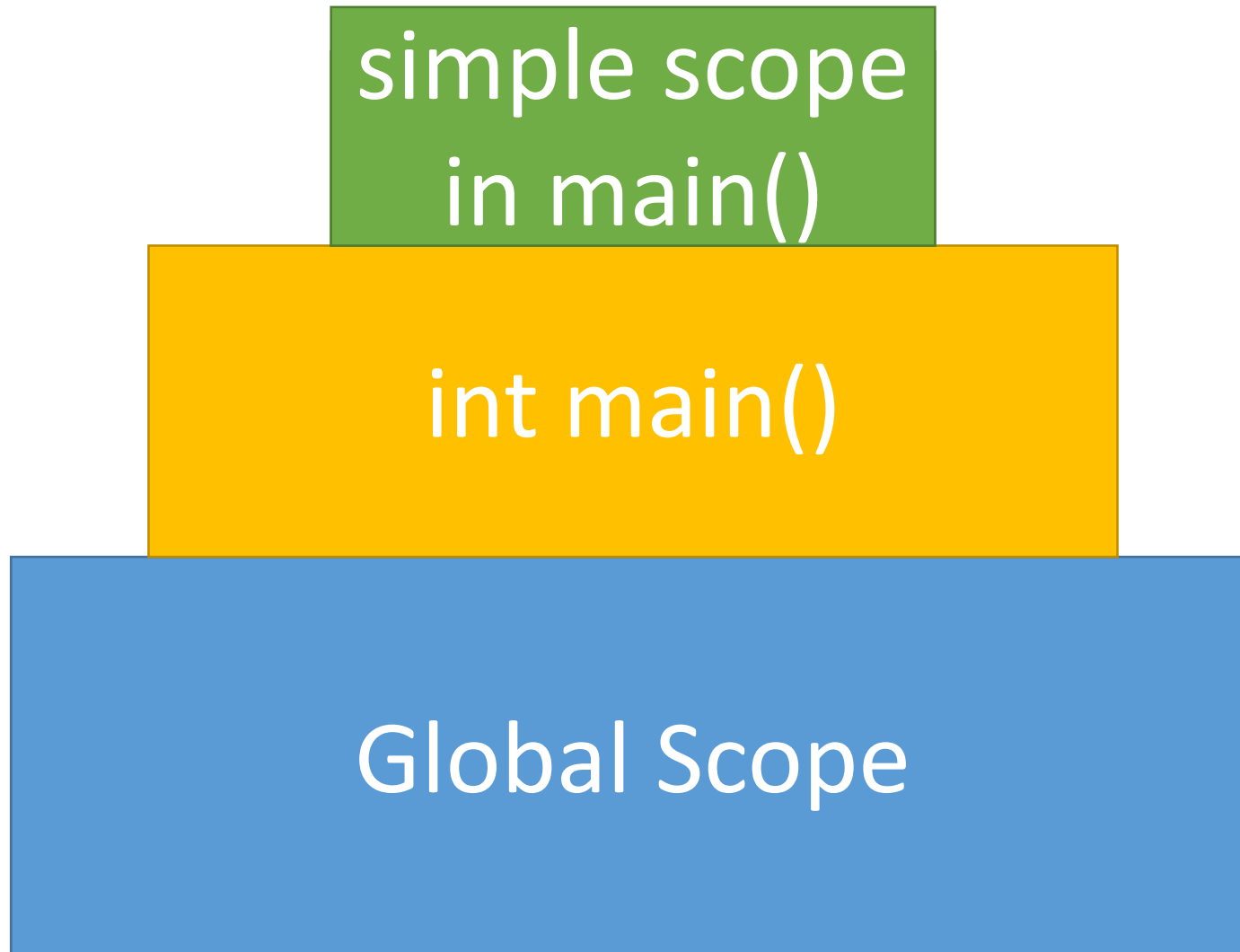
# Статични данни (глобални)

- Данни, които съществуват до края на програмата
- Досега знаете за глобални променливи
- Добра практика е глобалните променливи да са константни

# По-точно обяснение на score

- Дотук за score знаем, че:
  - Пази данните на всички променливи на score-а, в който се намира
  - Ако се създаде нова променлива със запазено име от външен score, то се забравя за старата променлива за този score
  - Когато свърши даден score, всички данни, създадени в него, изчезват
- Освен това:
  - данните, които заделяме в score, използват така наречената стекова памет
  - както сте се сетили, всички функции представляват score, включително и main

# Стек – Купчина



# Стек in a nutshell

- Стек е структура от данни, която няма да разглеждаме сега
- Накратко, в него се вкарват данни и единственият начин да се изкарат данни е да се изваждат отзад напред вкараните данни
- Пример:
  - Голяма колона коли засяда в тясна улица без изход
  - Единствено последната кола може да излезе на заден ход
  - След това само предпоследната може да излезе на заден ход
  - Така след краен брой стъпки и последната ще излезе



Дефиниция за стек според fmi.wiki

# Стек

---

- FILO: First-In-Last-Out

- *Принцип на библията:  
Последните ще бъдат първи*

# Стековата памет на интуитивно ниво

- Майстор Тричко прави ремонт. Задачата му е да смени кранчето за студената вода.
  - 1.Той започва да го сменя, но се обляга на мивката и я изкъртва.
  - 2.Сега задачата му е първо да смени мивката, но докато го прави спуква тръба.
  - 3.Сега задачата му е да оправи тръбата, но за да го направи трябва първо да спре течащата вода.
  - 4.Той спира водата.
  - 3.След това оправя тръбата.
  2. После оправя мивката.
  - 1.Накрая сменя и кранчето за студената вода.

# Стек – Купчина

```
int a = 5;
```

```
int main()
```

```
{
```

```
    char a = 'Q';
```

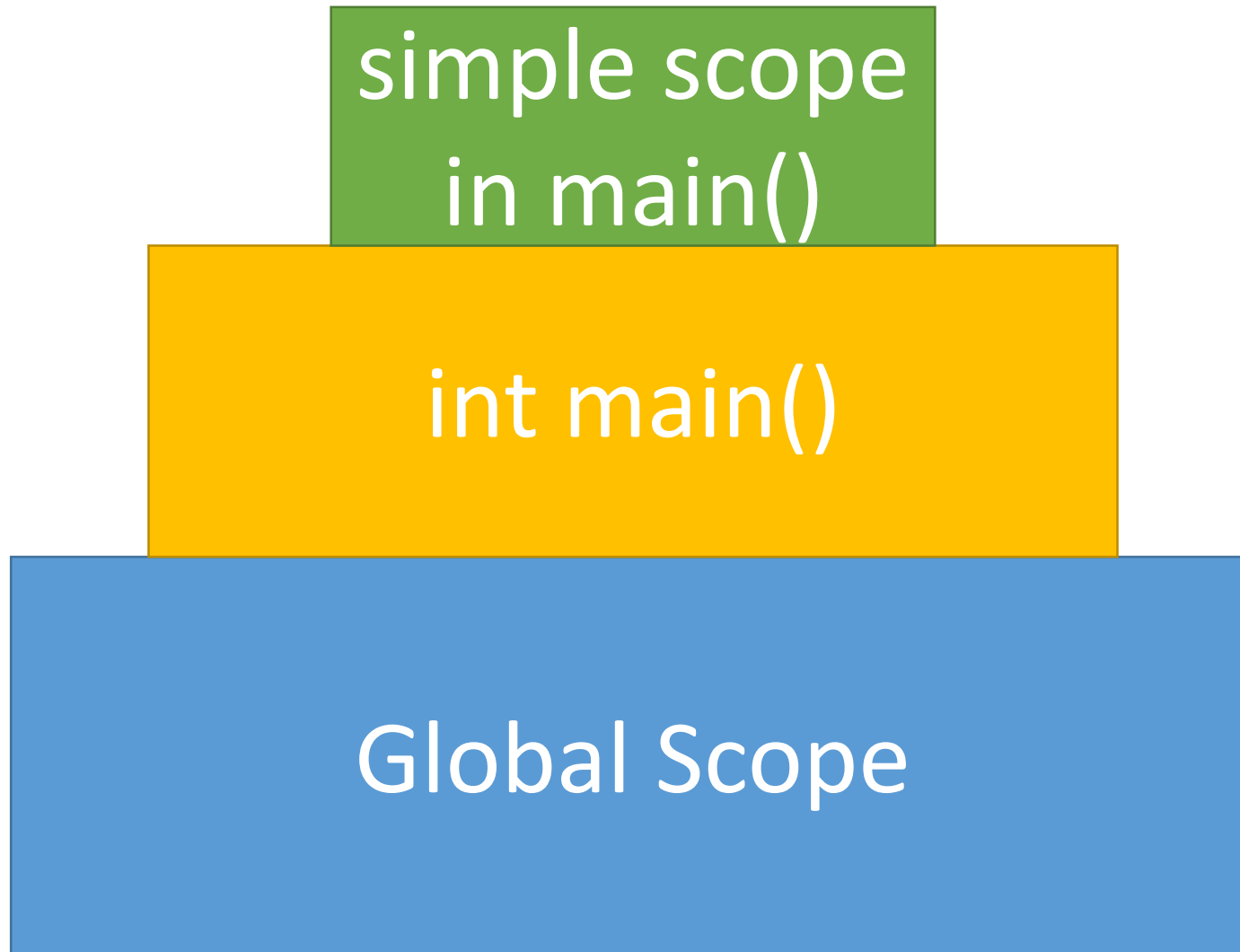
```
    {
```

```
        double a = true;
```

```
    }
```

```
}
```

# Стек – Купчина



# Стек – Купчина

```
int a = 5;
```

```
int main()
```

```
{
```

```
    char a = 'Q';
```

```
    {
```

```
        double a = true;
```

```
    }
```

```
}
```

# Стекова памет

- Заделя се в момента на дефиниция
- Всеки scope знае каква стекова памет е заделил
- При край на scope, той освобождава всичката стекова памет, която е заделил
- Последно заделената стекова памет се освобождава първа
- От горното свойство идва и наименованието на този вид памет

# Стекова памет

- Програмистът няма контрол над управлението на паметта
- Стекова памет не може да се освободи по-рано (преди края на блока)
- Стекова памет не може да се запази за по-дълго (след края на блока)
- До голяма степен работата със стековата памет е предопределена преди началото на програмата

# Статична срещу стекова памет

- И двете имат имена на заделената памет
- При статичната памет, данните съществуват до края на програмата, докато при стековата, до края на scope-а, в който са били създадени



# Какво имаме досега

- Имаме данни, които се запазват по време на компилация
- Тези данни си имат имена
- До голяма степен сме ограничени от езика да извършваме наглед прости операции, като:
  - контрол над паметта, която използваме, в реално време
    - заделяне на памет по време на изпълнение на програмата
    - освобождаване на памет по време на изпълнение на програмата

- Пример:

```
int n;
```

```
std::cin>>n;
```

```
int arr[n];
```

# Динамична памет (heap)

- Може да бъде заделена и освободена по всяко време на изпълнение на програмата
- Областта за динамична памет е набор от свободни блокове памет
- Програмата може да заяви блок с произволна големина
- За нейното управление се грижи операционната система
- Съответно не представлява никаква променлива, към която можем да се обърнем по име

# Задачи за вас #1

- Отидете на [www.menti.com](https://www.menti.com)

# Задача

- Кога ще се изтрият данните на променливата tmp?

```
int foo(const int numb)
{
    int tmp = numb%10;    //A
    if(tmp>5)
        return 1;        //B
    else
        return 3;        //C
    std::cout<<"Izpitat po UP ide\n";
}                          //D
```

Отговор: D, защото return води програмата до края на scope (виж следващите снимки)

Project3 (Debugging) - Microsoft Visual Studio

FileEditViewProjectBuildDebugTeamToolsTestAnalyzeWindowHelp

Quick Launch (Ctrl+Q)

Aston MartinAM

Process: [5208] Project3.exe Lifecycle Events Thread: [3184] Main Thread Stop Debugging (Shift+F5) foo

main.cpp

Project3 (Global Scope) foo(const int numb)

```
4 #include <ctime>
5 #include <algorithm>
6 #include <string>
7 int foo(const int numb)
8 {
9     int tmp = numb % 10; //A
10    if (tmp > 5)
11        return 1; //B
12    else
13        return 3; //C ≤2ms elapsed
14    std::cout << "hi";
15    std::cout << "hi";
16    std::cout << "hi";
17    std::cout << "hi";
18 } //D
19
20 int main()
21 {
22     foo(15);
```

Locals

Name	Value	Type
numb	15	const int
tmp	5	int

Diagnostic T...

Diagnostics sessio...

Events

Process Memory

Summary Events

Events

Show Events

Memory Usage

Take Snapshc

Output

Project3

Project3

Project3

Project3

Project3

Project3

Project3

Loading symbols for ucrtbased.dll

Ln 13 Col 1 Ch 1 INS

Add to Source Control

21:55 18/01/2019

Project3 (Debugging) - Microsoft Visual Studio

FileEditViewProjectBuildDebugTeamToolsTestAnalyzeWindowHelp

Quick Launch (Ctrl+Q)

Aston MartinAM

Process: [5208] Project3.exeLifecycle EventsThread: [3184] Main Thread

Stop Debugging (Shift+F5)foo

main.cpp

Project3

(Global Scope)

foo(const int numb)

```
4  #include <ctime>
5  #include <algorithm>
6  #include <string>
7  int foo(const int numb)
8  {
9      int tmp = numb % 10; //A
10     if (tmp > 5)
11         return 1; //B
12     else
13         return 3; //C
14     std::cout << "hi";
15     std::cout << "hi";
16     std::cout << "hi";
17     std::cout << "hi";
18 } //D ≤2ms elapsed
19
20 int main()
21 {
22     foo(15);
```

100 %

Locals

Name	Value	Type
numb	15	const int
tmp	5	int

Diagnostic T...

Diagnostics sessio...

Events

Process Memory

SummaryEvents

Events

Show Events

Memory Usage

Take Snapshc

Output

Project3

'Project3

'Project3

'Project3

'Project3

'Project3

'Project3

'Project3

Loading symbols for ucrtbased.dll

Ln 18Col 1Ch 1INS

Add to Source Control

Windows Taskbar

21:5518/01/2019

# Задача

- Кога ще се изтрият данните на променливата tmp?

```
int tmp = 5;
int foo(const int numb)
{
    tmp = numb%10;      //A
    if(tmp>5)
        return 1;      //B
    else
        return 3;      //C
    std::cout<<"Izpitat po UP ide\n";
}                      //D
```

Отговор: Нито едно от дадените, защото по така дадената информация, tmp е статична данна

# Задача

Динамичните данни имат сериозен недостатък – нямат имена.  
Как според вас можем да работим с данни, към които няма как дори да се обърнем поименно?

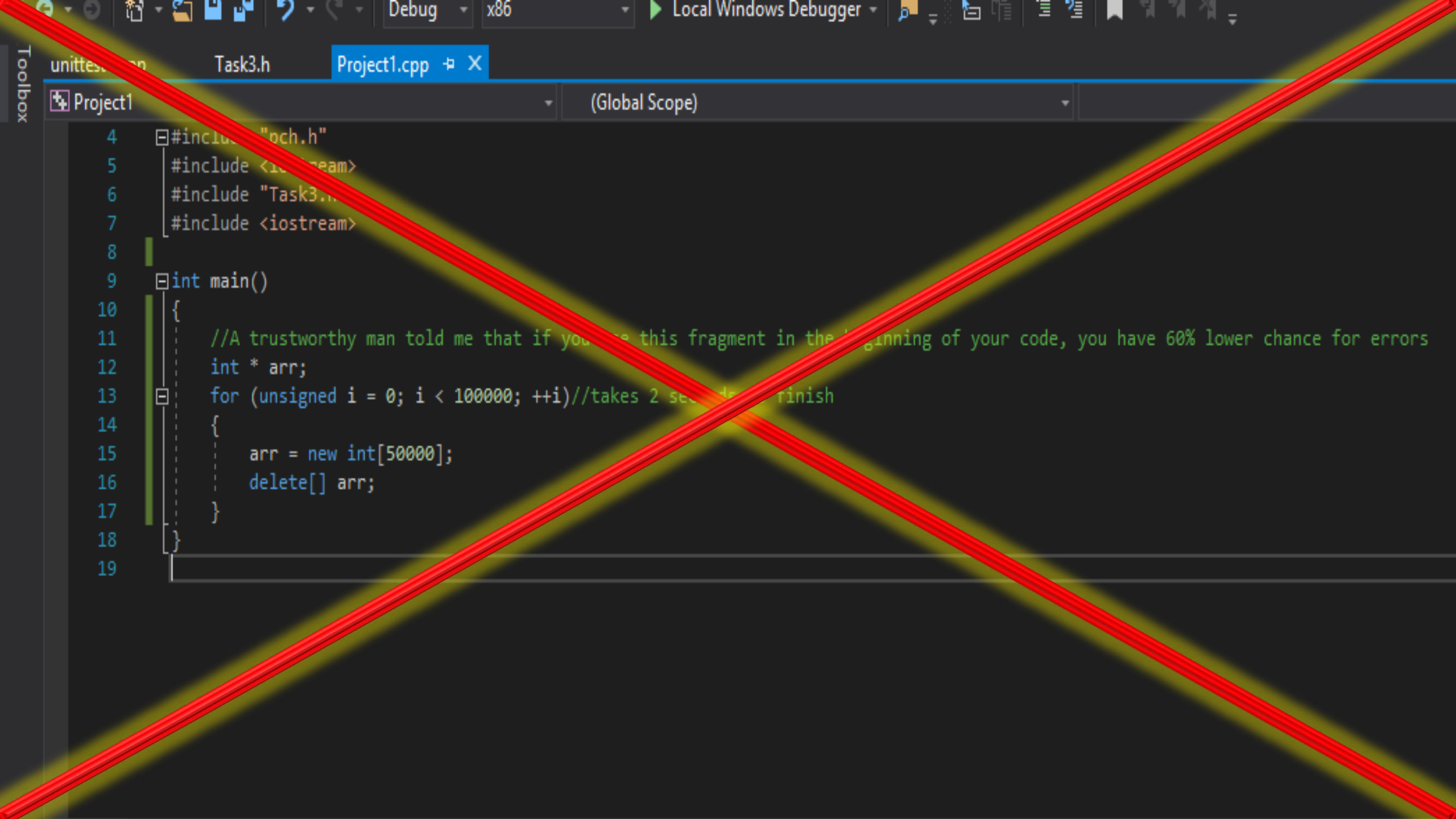
hint: Макар че нямат имена, те все пак имат адреси 😊

Отговор: можем да използваме пойнтьори към адресите на динамично заделените данни



# Работа с динамична памет

- Работата с динамична памет включва:
  1. Заделяне на такава памет
  2. Обработка на данни
  3. Освобождаване на заделената памет
- Като цяло 2. е optional, но реално ние използваме динамична памет точно заради 2.



unittests

Task3.h

Project1.cpp

Project1

(Global Scope)

```
4  #include "pch.h"
5  #include <iostream>
6  #include "Task3.h"
7  #include <iostream>
8
9  int main()
10 {
11     //A trustworthy man told me that if you use this fragment in the beginning of your code, you have 60% lower chance for errors
12     int * arr;
13     for (unsigned i = 0; i < 100000; ++i)//takes 2 seconds to finish
14     {
15         arr = new int[50000];
16         delete[] arr;
17     }
18 }
19
```

# Заделяне на динамична памет

- За заделяне на динамична памет се използват операторите:
  - `new <тип> [(<стойност>)]` – заделя памет за точно един нов обект, инициализира го и връща пойнтьр към него
  - `new <тип>[<число n>]` – заделя памет за n-мерна редица, инициализира всички обекти в нея и връща пойнтьр към първия
- Примери:
  - `char * dChar = new char;`
  - `char * dCharA = new char('A');`
  - `char * dCharArr = new char[6];`

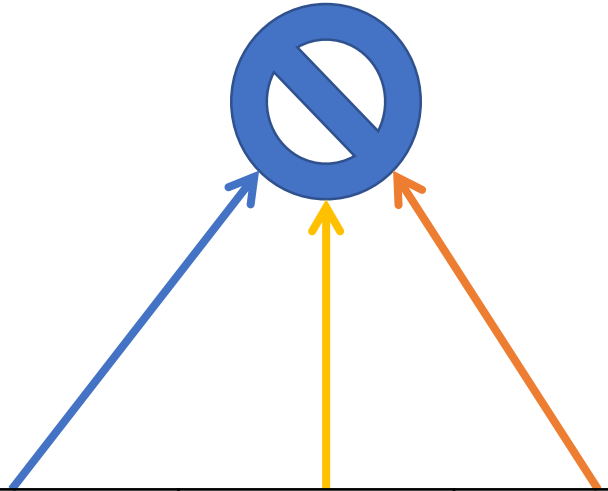
# Заделяне на динамична памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

# Заделяне на динамична памет - визуализация

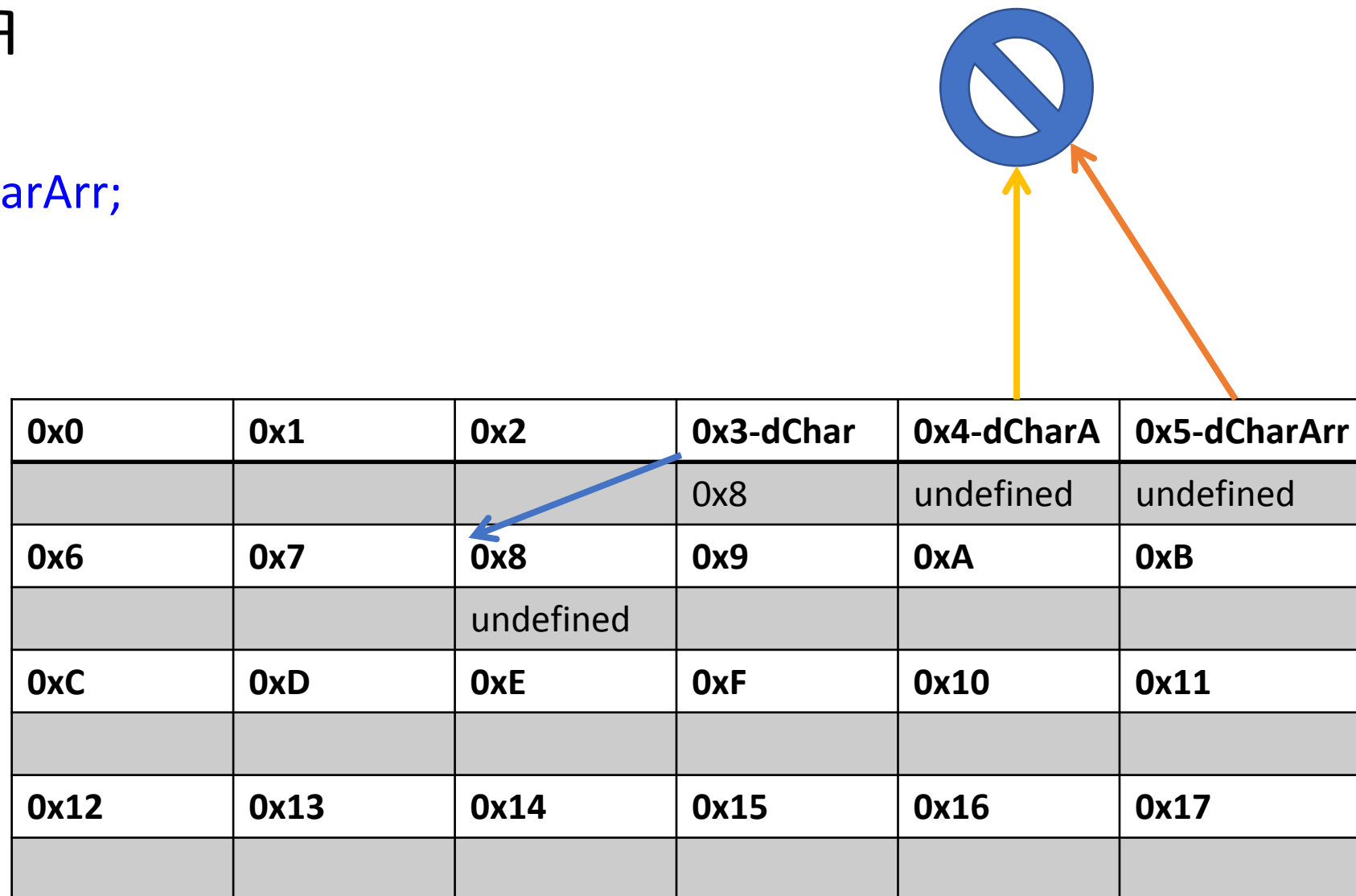
```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```



0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			undefined	undefined	undefined
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

# Заделяне на динамична памет - визуализация

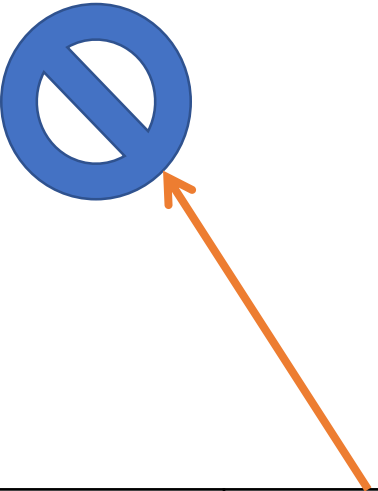
```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```



0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	undefined	undefined
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

# Заделяне на динамична памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```



0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	undefined
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17

# Заделяне на динамична памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined



# Освобождаване на заделена памет

- За освобождаване на динамична памет се използват операторите:
  - `delete <адрес>`
  - `delete[] <адрес>`
- Примери(спрямо предишните примери):
  - `delete dChar;`
  - `delete dCharA;`
  - `delete[] dCharArr;`

# Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined

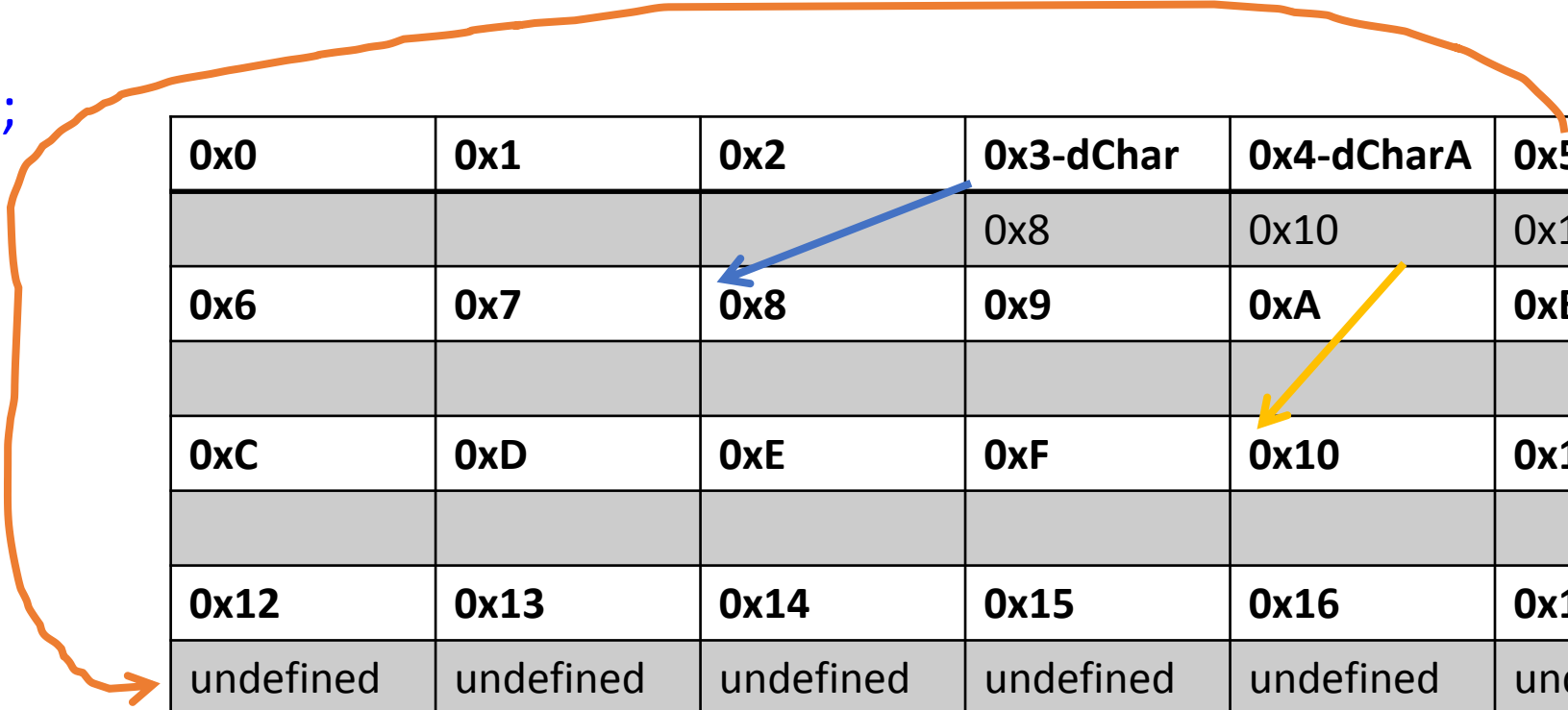
# Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined

# Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```



0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined

# Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

# Освобождаване на заделена памет

- delete операторите могат да освобождават само динамично заделена памет

- Не е позволено освобождаването на стекова памет:

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

- Не е позволено частично освобождаване на памет:

```
int* a = new int[10];  
int * b = a+1;  
delete[] b;
```

# Освобождаване на заделена памет

- За ваше улеснение, може да използвате правилото:
  - new => delete
  - new[] => delete[] *//delete[] си знае колко памет трябва да освободи*
- След освобождаването на дадена памет, тя става недостъпна и обръщането към нея обикновено води до фойерверки
- delete и delete[] върху nullptr са безобидни и не правят нищо
- **Винаги освобождавайте паметта, която сте заделили!!!**

# Освобождаване на заделена памет

- standard (5.3.5/2) :
  - In the first alternative (delete object), the value of the operand of delete shall be a pointer to a non-array object or a pointer to a sub-object (1.8) representing a base class of such an object (clause 10). **If not, the behavior is undefined.**
  - In the second alternative (delete array), the value of the operand of delete shall be the pointer value which resulted from a previous array new-expression. **If not, the behavior is undefined.**



# Обработка на динамични данни

- Особеностите на динамичните данни са:
  - заделяне на памет и освобождаването ѝ се извършва по време на изпълнение
  - липса на име
- Всичко останало си е както и преди, като трябва:
  - да се съобрази, че се обръщаме към пойнтьр от дадения тип, а не просто към обект
  - от сходствата между пойнтьри и масиви следва, че можем спокойно да използваме оператор [] **//припомнете си какво разглеждахме там**

# Обработка на динамични данни

- Възможността да контролираме кога дадена памет да бъде освободена ни дава изненадващо много нови възможности
- Вече функция, връщаща пойнтьър, може да има много повече приложения от преди
- With Great Power Comes Great Responsibility!

# Обработка на динамични данни

- Как се създава матрица NxM ?

```
int ** Matrix (const unsigned n, const unsigned m)
{
    int ** tmp = new int * [n];
    for(unsigned i = 0; i<n; ++i)
        tmp[i] = new int [m];
    return tmp;
}
```

# Обработка на динамични данни

- Как се изтрива матрица NxM ?

```
int ** A = Matrix(3,2);  
for(unsigned i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
bool ** tmp = new bool * [n];
```

```
for(char i = 0; i<n; ++i)
```

```
    tmp[i] = new bool [m];
```

```
return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9	0xA	0xB
10					
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
bool ** tmp = new bool * [n];
```

```
for(char i = 0; i<n; ++i)
```

```
    tmp[i] = new bool [m];
```

```
return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD	0xE	0xF	0x10	0x11
		Undefined	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17

# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
bool ** tmp = new bool * [n];
```

```
for(char i = 0; i<n; ++i)
```

```
    tmp[i] = new bool [m];
```

```
return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	0	Undefined	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17



# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{  
    bool ** tmp = new bool * [n];
```

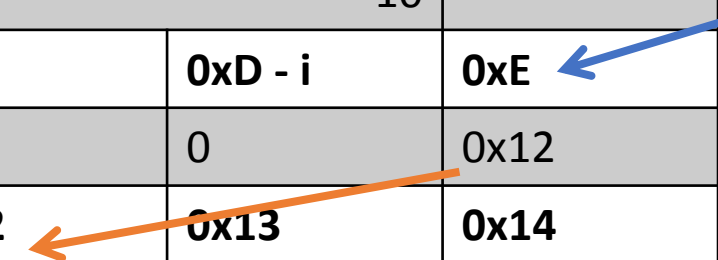
```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	0	0x12	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined				



# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
bool ** tmp = new bool * [n];
```

```
for(char i = 0; i<n; ++i)
```

```
    tmp[i] = new bool [m];
```

```
return tmp;
```

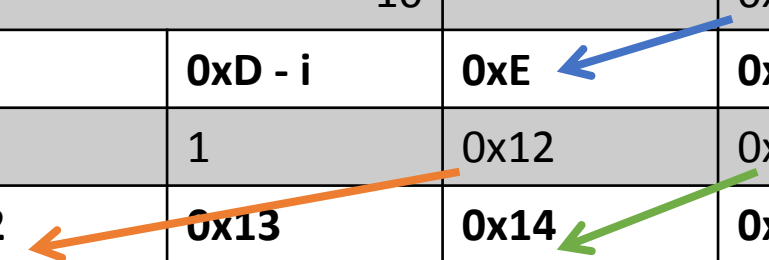
```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	1	0x12	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined				

# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
{
    bool ** tmp = new bool * [n];
    for(char i = 0; i<n; ++i)
        tmp[i] = new bool [m];
    return tmp;
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	1	0x12	0x14	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined		



# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
{
    bool ** tmp = new bool * [n];
    for(char i = 0; i<n; ++i)
        tmp[i] = new bool [m];
    return tmp;
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	10	0x12	0x14	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined		

# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
bool ** tmp = new bool * [n];
```

```
for(char i = 0; i<n; ++i)
```

```
    tmp[i] = new bool [m];
```

```
return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	10	0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
bool ** tmp = new bool * [n];
```

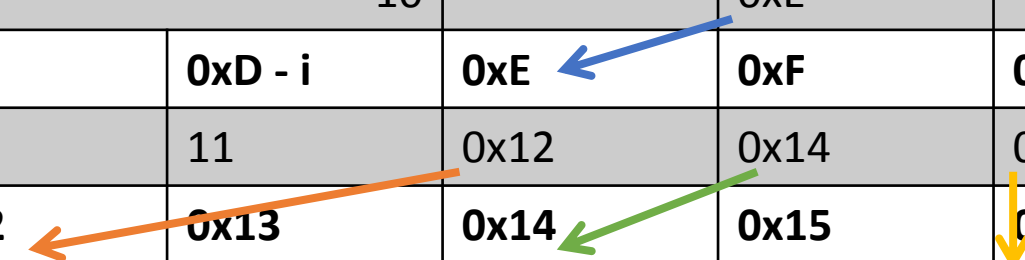
```
for(char i = 0; i<n; ++i)
```

```
    tmp[i] = new bool [m];
```

```
return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	11	0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined



# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

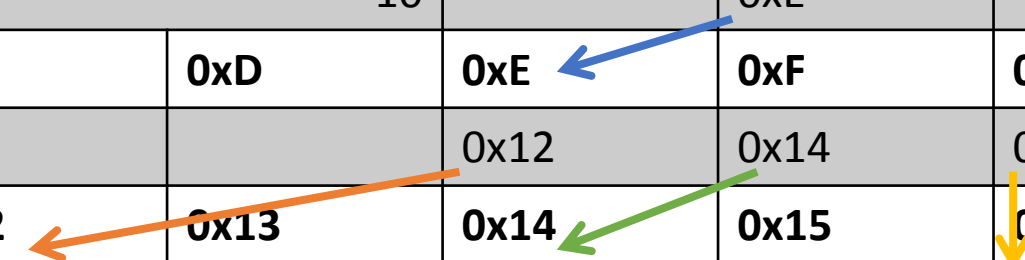
```
{  
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)  
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

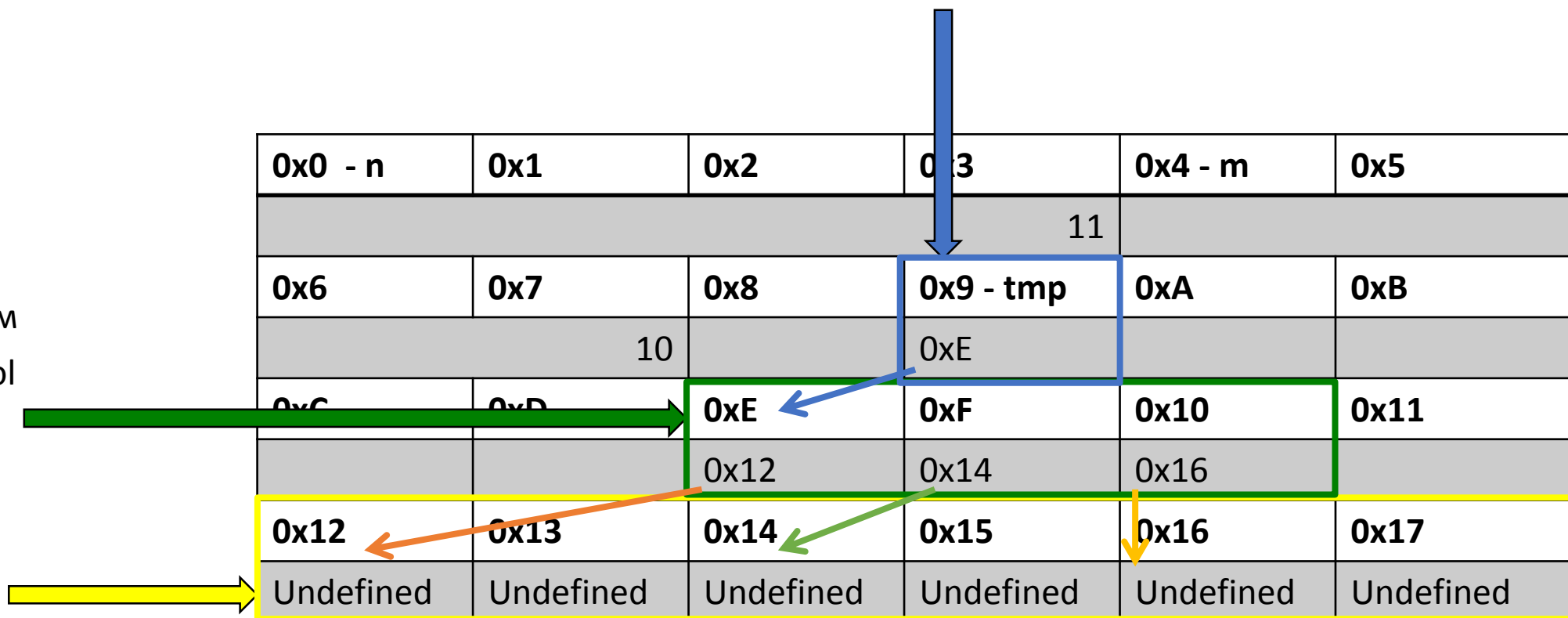


# Обработка на динамични данни - визуализация

Пойнтьър към пойнтьър от тип bool (bool \*\*)

Редица от пойнтьри към  
променливи от тип bool  
(bool \*)

Редица от променливи  
от тип bool (bool)





# Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

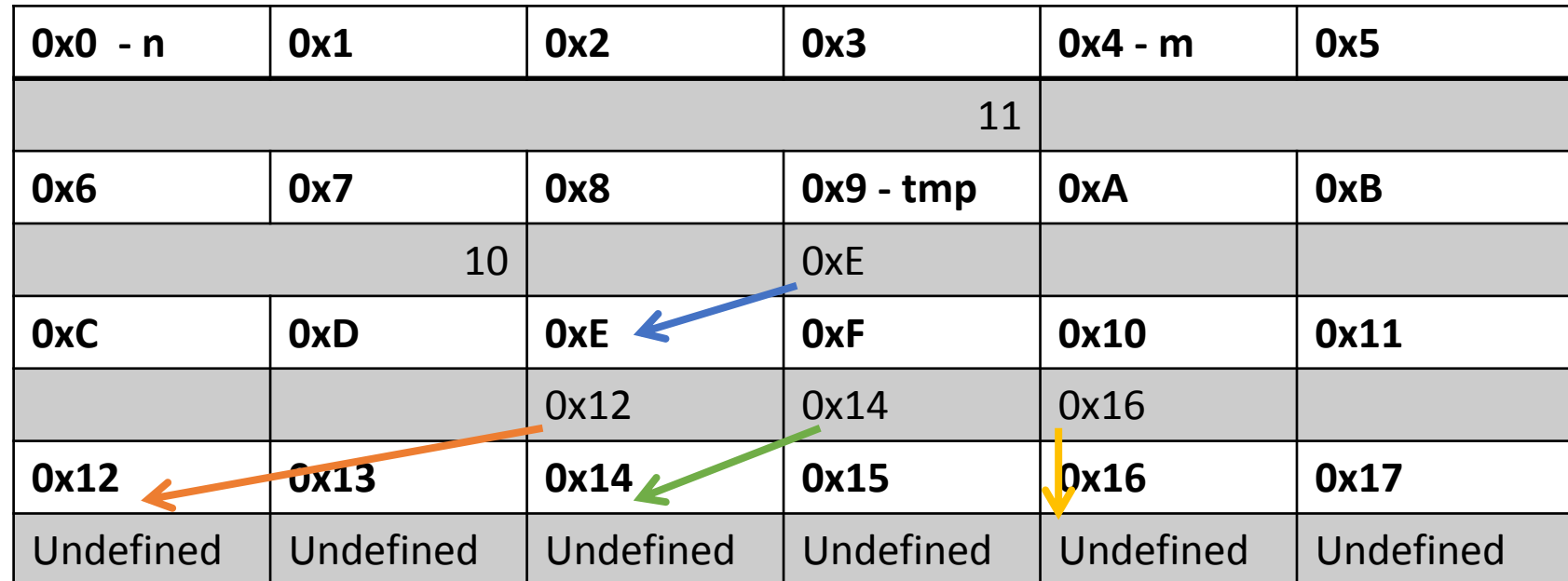
```
{  
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)  
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
11					
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
10			0xE		
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined



# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
```

```
for(char i =0; i<3;++i)
```

```
{
```

```
    delete[] A[i];
```

```
}
```

```
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

The diagram illustrates memory management for a dynamically allocated array of pointers. The memory layout is shown as a table of 6x6 cells. The first row contains indices 0x0 to 0x5. The second row is empty. The third row contains indices 0x6 to 0xB, where 0xB is labeled 'A'. The fourth row contains indices 0xC to 0x11, with 0xE in the 0x5 column. The fifth row contains indices 0x12 to 0x17, with 0x12 in the 0x2 column, 0x14 in the 0x3 column, and 0x16 in the 0x4 column. The sixth row contains 'Undefined' values. Arrows indicate pointer manipulation: a blue arrow points from 0xB to 0xE, an orange arrow points from 0x13 to 0x12, a green arrow points from 0x15 to 0x14, and a yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

Матрица 3X2

0x12	0x13
0x14	0x15
0x16	0x17

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	0
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	0
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	0
0x12	0x13	0x14	0x15	0x16	0x17
		Undefined	Undefined	Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	1
0x12	0x13	0x14	0x15	0x16	0x17
		Undefined	Undefined	Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	1
0x12	0x13	0x14	0x15	0x16	0x17
		Undefined	Undefined	Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.



# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	1
0x12	0x13	0x14	0x15	0x16	0x17
				Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16 to Undefined.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	10
0x12	0x13	0x14	0x15	0x16	0x17
				Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	10
0x12	0x13	0x14	0x15	0x16	0x17
				Undefined	Undefined

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16 to Undefined.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	10
0x12	0x13	0x14	0x15	0x16	0x17

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	11
0x12	0x13	0x14	0x15	0x16	0x17

The diagram illustrates memory addresses and pointers. A blue arrow points from 0xB to 0xE. An orange arrow points from 0x13 to 0x12. A green arrow points from 0x14 to 0x14. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

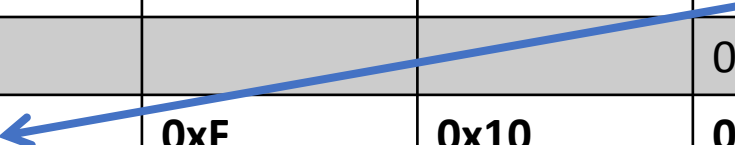
0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17

The diagram illustrates the deallocation of a 3x2 matrix A. The table shows memory addresses 0x0 to 0x17. A blue arrow points from 0xB (labeled 'A') to 0xE. An orange arrow points from 0x12 to 0x13. A green arrow points from 0x14 to 0x15. A yellow arrow points down from 0x16.

# Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);  
for(char i =0; i<3;++i)  
{  
    delete[] A[i];  
}  
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17



# Задачи за вас #2

- Отидете на [www.menti.com](https://www.menti.com)



# Задача

- Какво ще се случи?

```
int* a = new int[10];  
a[5] = 5;  
delete a[0];  
std::cout << a[5];
```

Отговор: Недефинирано поведение, памет заделена с new[] трябва да се трие с delete[]

# Задача

- Какво ще се случи?

```
int* a = new int[10];
```

```
*(a++) = 5;
```

```
*a*=0;
```

```
std::cout<<*a;
```

```
delete []a;
```

Отговор: Недефинирано поведение, памет заделена с new[] се трие с delete[] само от адреса, който е бил върнат от new[]

# Задача

- Какво ще се случи?

```
int* a = new int[10];
```

```
*(a++) = 5;
```

```
std::cout << a[-1];
```

```
delete[] (a-1);
```

Отговор: Ще се изведе 5 и всичко ще е точно

# Почивка

- 10 минути заслужена почивка 😊

# Типове данни

- Още на първата консултация направихме едно условно разграничение на типовете данни:
  - примитивни
  - съставни
- Вече знаем кои са примитивните
- Тогава какво остава за съставни?

# Съставни типове данни

- От самото име следва, че са съставени от нещо, но от какво
- Всеки съставен вид данни се състои или надгражда други типове данни, без значение дали са примитивни, или съставни

⇒ Пойнтъри и масиви

# Структура

- Добре дошли в курса по ООП (spoiler alert)
- Представя обединение от данни
- Винаги има фиксиран брой елементи
- Елементите могат да са от различни типове
- По подразбиране предоставя директен достъп до всеки елемент

# Структура

- Подобно на функциите, при структурата има 2 ключови момента
- Декларация+дефиниция
- Извикване на дефинираното



# Структура

- Декларация

```
struct <име>;
```

- Дефиниция + декларация

```
struct <име>  
{  
    [<тяло>]  
};
```

# Структура

- Също както при функции, съществува и така нареченият forward declaration:

```
struct example;
```

.....

```
struct example  
{  
};
```

# Структура

- Нека разгледаме какво се случва при дефиниция

struct example

{

//казахме, че тук може да има някакво тяло

}; //не забравяйте ; след края на scope на декларация на структура

# Структура

- Нека разгледаме какво се случва при дефиниция

`struct example`

`{`

`//в тялото се записват така наречените член-данни и функции`

`//сега ще говорим само за член-данни, а след време и за член-функции`

`//нека example е съставена от 1 променлива от тип int`

`int member1;`

`};` `//не забравяйте ; след края на scope на декларация на структура`

# Структура

- Нека разгледаме какво се случва при дефиниция

struct example

```
{  
    int member1, member2, member3;  
    char member4;  
    double members5To10[5];  
}; //не забравяйте ; след края на scope на декларация на структура
```

# Структура

- Нека първо преминем към втората част – извикване на структура, а после ще се върнем към дефиниране, за да разширим наученото

- Извикване на структура

<Име на структурата/тип> <наименование>;

- Примери:

example a;

example b;

Какво означава неинициализиран обект?

# Структура

- Нека разгледаме какво се случва при дефиниция

struct example

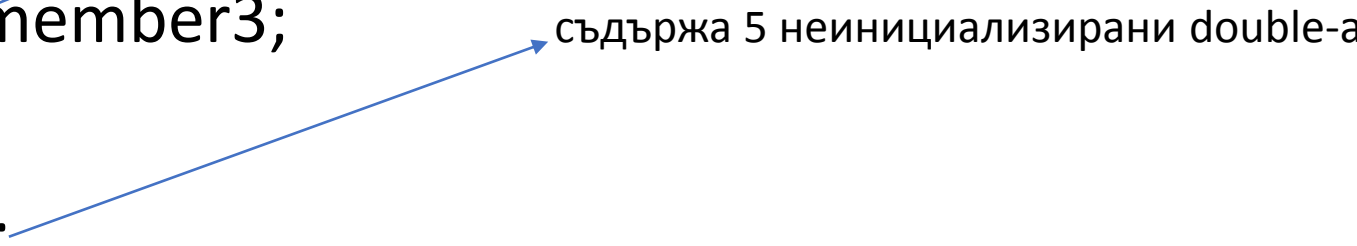
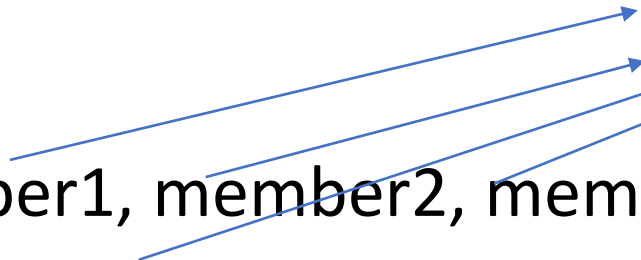
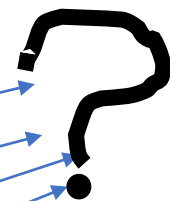
{

int member1, member2, member3;

char member4;

double members5To10[5];

}; **//не забравяйте ; след края на scope на декларация на структура**



# Структура

- Извикване на структура

<Име на структурата/тип> <наименование>;

- При създаване на обект можем и да инициализираме член-данните по 3 начина, като по-използваният ще остане за курса по ООП

<Име на структурата/тип> <наименование> = { <стойност\_за\_член\_1>, <стойност\_за\_член\_2>,.....};

- По този начин не може да се пропускат членове и може да се спре по всяко време



# Структура

- Нека използваме за пример

```
struct example{  
    int member1, member2, member3;  
    char member4;  
    double members5To10[5];  
};
```

.....

example a = {1};	//даваме стойност 1 на member1
example b = {1,2,3};	//даваме стойности на member1-3
example c = {1,2,3,'!'};	//даваме стойности на member1-4
example d = {1,2,3,'!',1,2,3,4,5}	//даваме стойности на member1-5

# Структура

- Нека използваме за пример

```
struct example{  
    int member1, member2, member3;  
    char member4;  
    double members5To10[5];  
};
```

.....

```
example a = {1};
```

Знаем, че member1 има стойност 1, но как да я използваме?

# Структура

- Нека пак имаме структурата example

example a = {1};

- Как да достъпим member1?
- Достъпът в този случай се осъществява с оператор .(точка) + име на член
- `std::cout<<a.member1; //извежда 1`

# Структура

- Какво означава „в този случай“?
- Има случаи, в които обръщението към член данна не се осъществява с оператор .(точка)
- Има случаи, в които нямаме достъп до дадена член данна
- Надявам се някой ден да се реванширам и да ги обясня в презентация на тема ООП

# Структура

- Какво ни дава достъпът до дадена член-данна?
- Достъп до информацията, която съдържа
- Възможност да променяме стойността ѝ ако не е константна
- Пример

`a.member1 = 10;`

# Структура

- Как една член данна може да е константна
- Има 2 начина
  1. Самата член данна да е константна
    - `const int member11`
  2. Цялата структура да е създадена константна
    - `const example b; //лош пример, ще има фойерверки`
- Какви правила трябваше да спазваме при работа с константи?

# Структура

- Когато създаваме константа винаги трябва да задаваме някаква стойност при инициализация.
- Структурите имат начин за справяне с този проблем, освен използването на { } при инициализация
- Спомняте ли си параметрите по подразбиране?
- Също като тях, членовете на структурата могат да имат стойности по подразбиране

# Структура

- За разлика от параметрите по подразбиране на функциите, при структурите не е необходимо само първите  $n$  на брой да имат стойности по подразбиране

- Пример:

```
struct example{  
    int member1 = 2, member2 = 6, member3 = 534;  
    char member4;  
    double members5To10[5] = {1,2,3,4,5};  
};
```



# Структура

- Пример:

```
struct example{
```

```
    int member1 = 2, member2 = 6, member3 = 534;
```

```
    char member4 = '?';
```

```
    double members5To10[5] = {1,2,3,4,5};
```

```
};
```

```
example a;
```

```
//example a = {2, 6, 534, '?', 1, 2, 3, 4, 5};
```

```
example b = {5,4};
```

```
//example b = {5, 4, 534, '?', 1, 2, 3, 4, 5};
```

```
example c = {12,34,1, '%', 5};
```

```
//example c = {12,34,1, '%', 5, 2, 3, 4, 5};
```

# Задачи за вас #3

- Отидете на [www.menti.com](https://www.menti.com)

# Задача

- Валидно ли е следното?

```
struct empty
{
};
int main()
{
    empty e;
    return 0;
}
```

Отговор: Да. Не могат да те използват ако си безполезен, но това не значи, че не можеш да съществуващ

# Задача

Ще се компилира ли този код?

```
struct empty{  
    int a;  
    int b;  
    const int c = b;  
};  
int main(){  
    empty e;  
    return 0;  
}
```

Отговор: Да, защото присвояваме стойност на константата, каква ще бъде тази стойност знае само катерицата Беатрис (undefined behaviour)

# Задача

- Казахме, че структурите могат да съдържат всякакви данни. Тогава ще се изпълни ли това?

```
struct empty
{
    empty A;
};
int main()
{
    empty e;
    return 0;
}
```

Отговор: Не, в C++ не е позволено обект да съдържа обект от същия тип в себе си

# Структури

- Ако една структура съдържа обект от същия тип се получава безкрайна рекурсия (съжалявам, че не остана време за рекурсия)
- Едно коте чело книжка за едно коте, което чело книжка

# Структури

- [illegible]

# Структури

- [illegible]

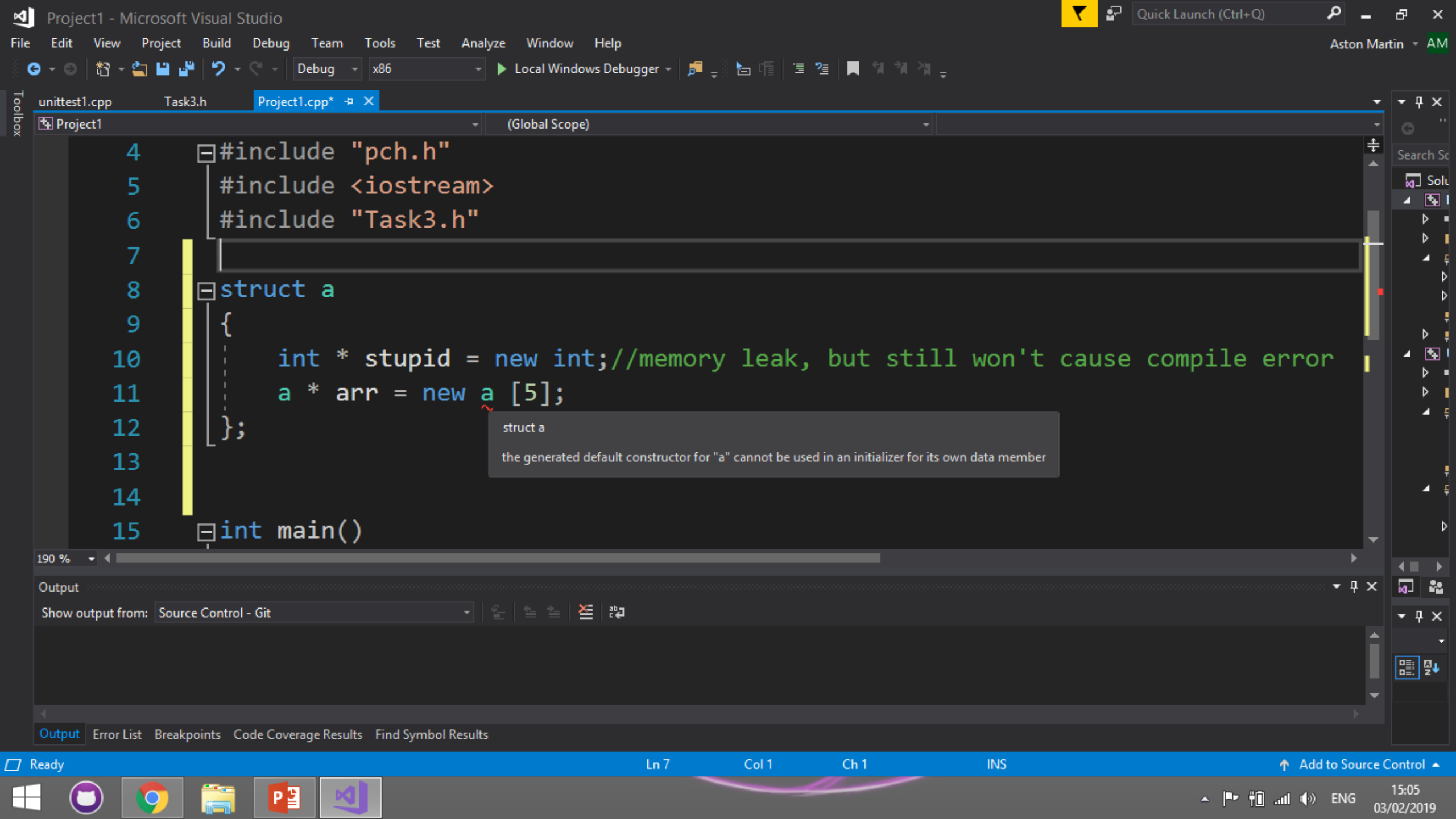


# Структури

- [illegible]

# Структури

- ..... и така докато не свърши свободната памет и не избухнат фойерверки
- Може да съдържа поинтър към обект от същия тип
- Поинтърът не съдържа член данни и прочие, а само адрес, затова няма да се получи рекурсия както преди малко
- Не можете да зададете на такъв поинтър заделяне на динамична памет като параметър по подразбиране



# Структури

- Съществува присвояване на стойностите на структури
- То може да се осъществява само между структури от един и същи тип
- Използва се оператор =
- Буквално се прехвърлят стойностите на всяка член данна 1 по 1

# Структури

example a = {1,2,3,4,5,6}, b = {8,5,3,1,6,7}; //да си представим, че има  
//само 6 член данни от тип int

a = b;

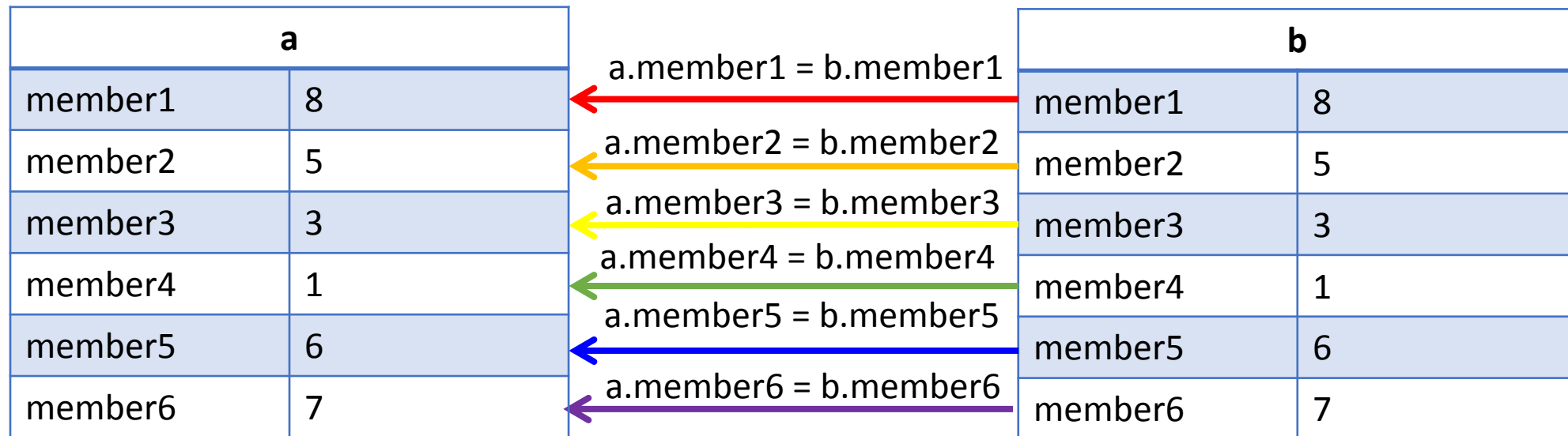
a	
member1	1
member2	2
member3	3
member4	4
member5	5
member6	6

b	
member1	8
member2	5
member3	3
member4	1
member5	6
member6	7

# Структури

example  $a = \{1, 2, 3, 4, 5, 6\}$ ,  $b = \{8, 5, 3, 1, 6, 7\}$ ; *//да си представим, че има  
//само 6 член данни от тип int*

$a = b;$



# Структури

- За структурите важат същите правила за подаване като параметър на функция и връщане като резултат както при примитивните данни
- При подаване като параметър, се създава нов локален обект, на който се присвоява стойността на подадения обект
- При връщане като резултат, се създава нов временен обект, на който се присвоява стойността на това, което връщаме
- Колко от вас видяха [Demo1](#) от миналата консултация

# Задачи за вас #4

- Отидете на [www.menti.com](https://www.menti.com)

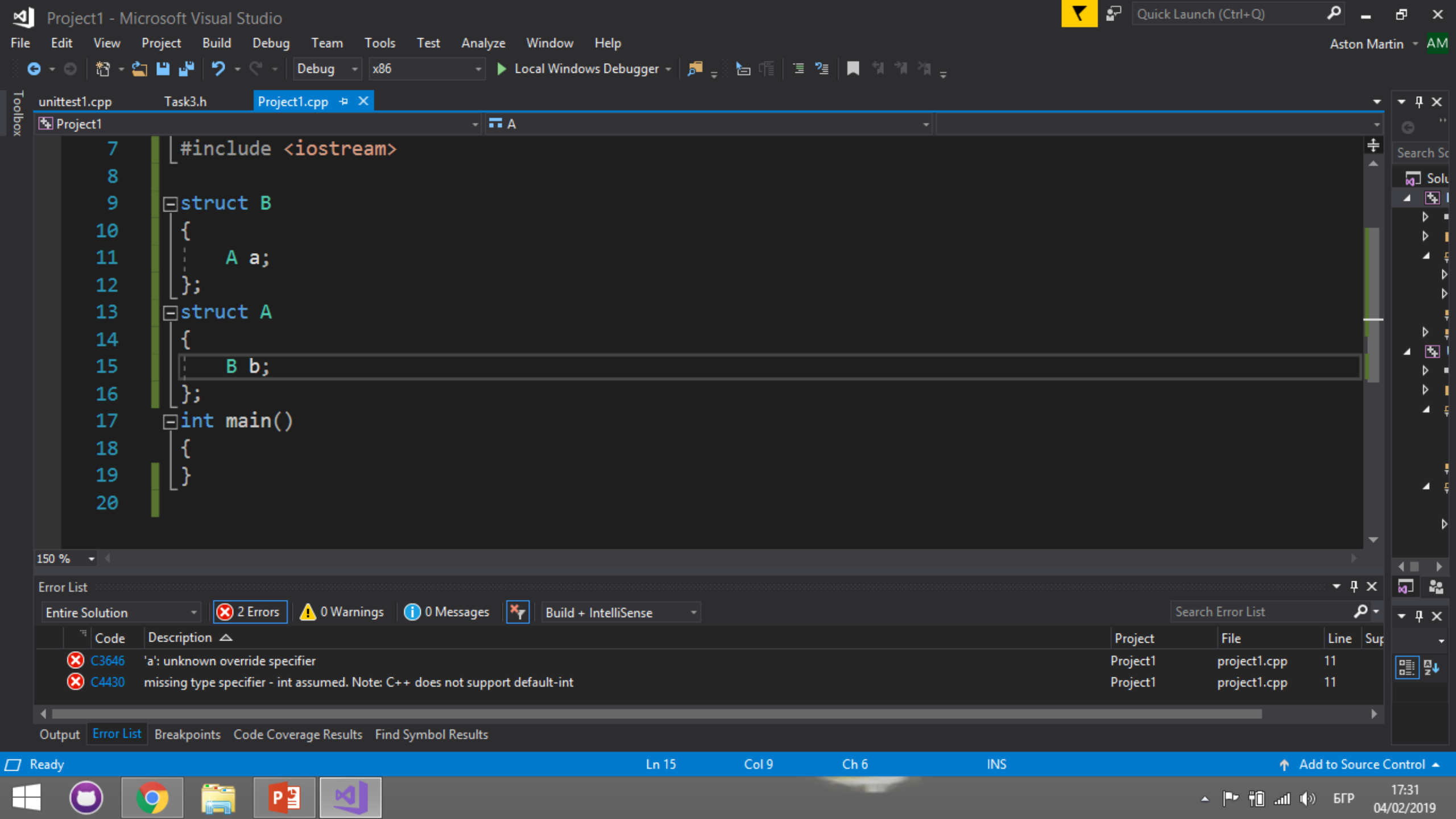


# Задача

- Ще се компилира ли следният код?

```
struct B
{
    A a;
};
struct A
{
    B b;
};
```

Отговор: Не, защото структурата B не знае за съществуването на A



# Задача

- Ще се компилира ли следният код?

```
struct A;
```

```
struct B
```

```
{
```

```
    A a;
```

```
};
```

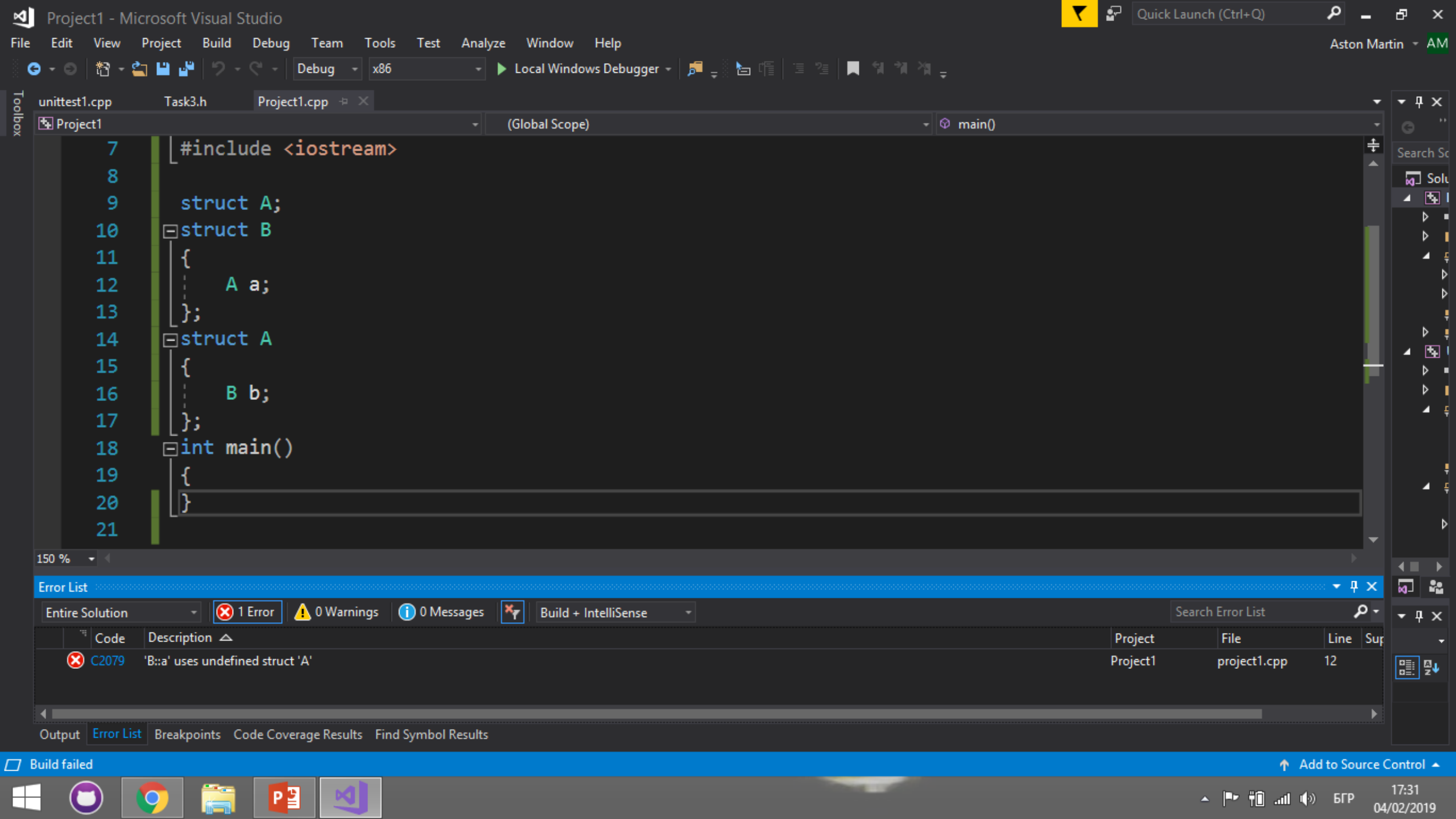
```
struct A
```

```
{
```

```
    B b;
```

```
};
```

Отговор: Не, защото структурата A не е дефинирана, когато структурата B се обръща към нея



# Задача

- Ще се извърши ли поотделно присвояване за всяка член данна, когато се присвоява стойност на поинтър към дадена структура?

Отговор: Не. Поинтърите съдържат просто адрес, те нямат член-данни!

# Почивка

- 10 минути почивка, след което минаваме към задачи 😊

# ИЗТОЧНИЦИ

- Голяма част от информацията е сверена с <https://en.cppreference.com>
- Използвани са дефиниции и описания от материали на доц. Трифон Трифонов
- Авторският код е проверяван на [VisualStudio2017](#)
- <https://study.com/academy/lesson/program-memory-in-c-programming.html> - използвано изображение на слайд 4