Добре дошли

- Отидете на www.menti.com

- Това е сайт за анкети, който ще използваме активно на тази консултация

- Код: /*ПОПЪЛНИ В НАЧАЛОТО*/
- Подредете по приоритет за вас темите, които ще разглеждаме днес

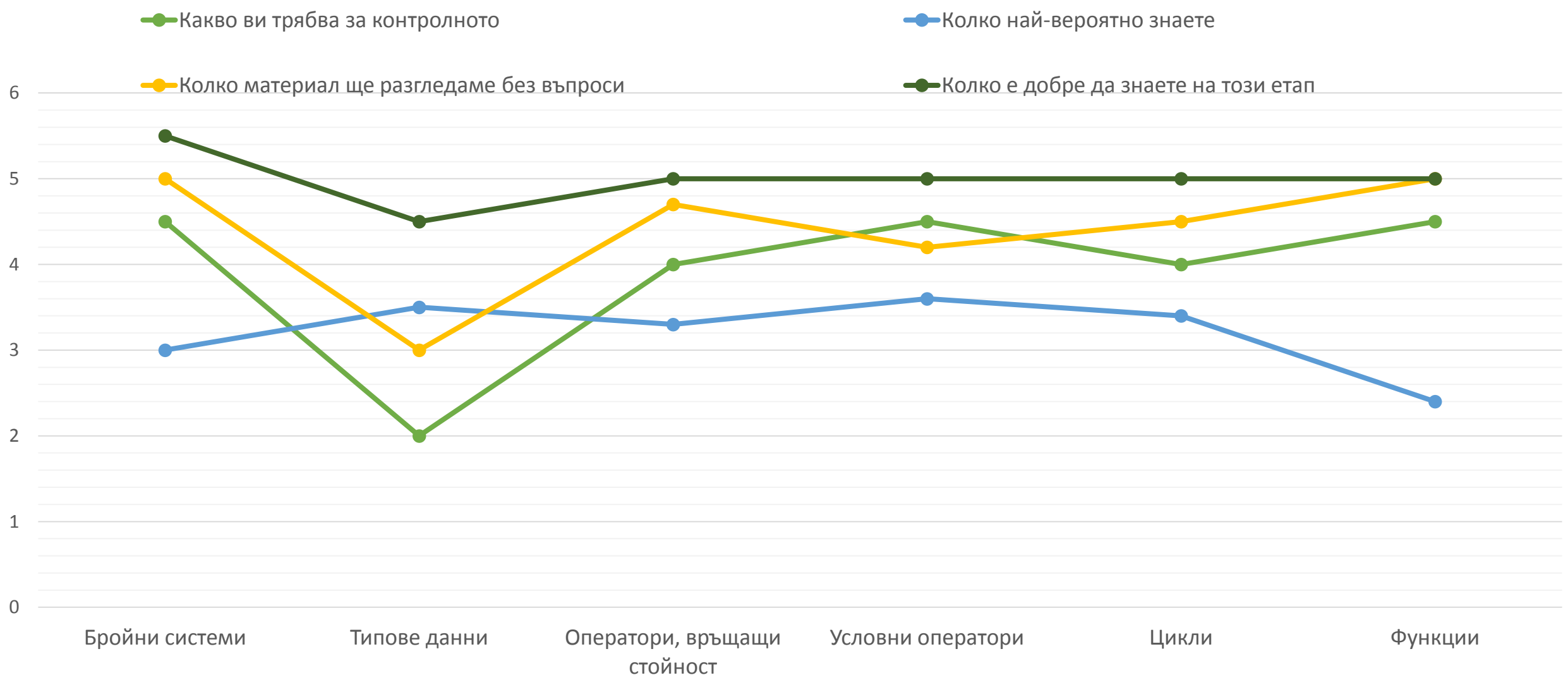
Консултация по УП за теоретично контролно 1

Изготвена и представена от Мартин Илиев

Какво покрива тази презентация

- Бройни системи
- Типове данни
- Оператори връщащи стойност
- Условни оператори
- Цикли
- Функции
- Време за въпроси (надявам се да остане такова)

Анализ на материала



Какво всъщност е бройна система

- Начин за представяне на числата посредством дадена азбука
- Основа:
  - градуси, минути и секунди – 60
  - часове – 24
  - RGB - 16
  - финанси - 10
  - машинен код - 2

Десетична бройна система

- Азбука 0-9
- Основа 10

- Число в десетичен запис, преведено в десетична бройна система

  $43\ 671 = 4x10^4 + 3x10^3 + 6x10^2 + 7x10^1 + 1x10^0$

Двоична бройна система

- Азбука 0-1
- Основа 2

- Число в двоичен запис, преведено в десетична бройна система

  $110010_{(2)} = 1x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 50$

Шестнайсетична бройна система

- Азбука 0-F

  A=10, B=11, C=12, D=13, E=14, F=15

- Основа 16

- Число в шестнайсетичен запис, преведено в десетична бройна система

  $F6A_{(16)} = 15x16^2 + 6x16^1 + 10x16^0 = 3946$

decimal	hexadecimal	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Преминаване в и от десетична бройна система

- Преминаването в десетична бройна вече е показано в предните слайдове

- Преминаване в друга бройна система n
  - Делим на n и запазваме остатъка, докато не получим 0

36/ 2 = 18 (остатък 0)
18/ 2 = 9 (остатък 0)
9 / 2 = 4 (остатък 1)
4 / 2 = 2 (остатък 0)
2 / 2 = 1 (остатък 0)
1 / 2 = 0 (остатък 1)

  - Записваме остатъците в обратен ред

=> 36 = $100100_{(2)}$

Преминавания в бройни системи с основи, когато една от основите е степен на другата

- $100100_{(2)} = 1x2^5 + 1x2^2 = 36$

  36 / 16 = 2 (Остатък 4)
  2 / 16 = 0 (Остатък 2)

- $100100_{(2)} = 24_{(16)}$

- Ами ако имаме $10101010001111010101011010_{(2)}$???

Преминавания в бройни системи с основи, когато една от основите е степен на другата

- $10101010001111010101011010_{(2)}$
- $01010101000111101010101110 10_{(2)}$ -стойността не се променя

- $0101|0101|0001|1110|1010|1011|1010_{(2)}$

- 5 5 1 E A B A

- $=>10101010001111010101011010_{(2)} = 551EABA_{(16)}$

Преминавания в бройни системи с основи, които не са взаимно прости

- $732472_{(8)} = ?_{(2)}$
- 7 3 2 4 7 2
- 111 011 010 100 111 010
- $732472_{(8)} = 111\ \underline{0}11\ \underline{0}10\ 100\ 111\ \underline{0}10_{(2)}$
- !Не забравяйте, че всяко разбиване трябва да съдържа даден брой цифри, затова винаги проверявайте!

Какво са бит, байт, килобайт и т.н.?

- Бит(*binary digit*) - 1 двоична цифра

  $1\ 0\ 0\ 0\ 1_{(2)}$ – число с 5 бита

- Байт – 8 бита

  $1\ 0\ 0\ 1\ 1\ 0\ 0\ 1_{(2)}$ - число с 8 бита или 1 байт

- Килобайт – 1000/1024 байта (спори се)
  - International System of Units – 1kb = 10^3 bytes
  - Microsoft – 1kb = 2^10 bytes

Типове данни

- Усложняват работата на начинаещите програмисти
- Позволяват управление на паметта
- Помагат за проверка на логика
- Определят множество от допустими стойности
- Делят се на примитивни и съставни

Примитивни типове данни

- булев (bool)
- целочислен (int)
- символен (char)
- изборен (enum) – в бонус материалите
- числа с плаваща запетая (float, double)

Размер на примитивните типове данни

*Информацията е валидна за 64 битова ОС, x64 базиран процесор и VC компилатор

- 1 byte – bool, char
- 2 bytes – short (int)
- 4 bytes – int, long(int), float, enum
- 8 bytes – long long(int), double, long double

Boolean

- Ключова дума bool
- Множество от стойности: {0, 1}
- 0 и 1 могат да бъдат заместени с литералите false и true
- false е равностойно на 0
- true е равностойно на 1
- Всяка стойност различна от 0 се смята за true

Цели числа

- Ключова дума int (или само името на модификатора)
- Множество от стойности: $[-2^{31};2^{31}-1]$
- Модификатори
  - short: $[-2^{15};2^{15}-1]$
  - long: $[-2^{31};2^{31}-1]$
  - long long: $[-2^{63};2^{63}-1]$
  - unsigned: $[0;2^{x}-1]$, където (x = 16,32,64)

- По подразбиране целочислените константи са int, за да се третира като long long трябва да го отбележим:
  - 123414234123523L

Числа с плаваща запетая

- Ключова дума double
- Множество от стойности: +/- 1.7e +/- 308 (~15 digits)
- Модификатори
  - long: също като при int нищо не прави под моята архитектура

Числа с плаваща запетая

- Ключова дума float
- Множество от стойности: +/- 3.4e +/- 38 (~7 digits)
- По подразбиране константите с плаваща запетая са double, за да се отбележи, че искаме да се третира като float трябва да го отбележим:
  - 3.14F

Character

- Ключова дума char
- ‘ ‘ – използват се, за да работим директно със символ
- ‘ ‘ съдържат само 1 символ
- ‘\ ‘ – специален символ, няколко символа образуват 1 конкретен
  - ‘\n’, ‘\t’, ‘\\’
- Приложения на ASCII таблицата може да разгледате в бонус материалите

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Константи и литерали

- Константните и литералните стойности в C++ са такива, които не могат да се променят
- Числовите стойности
  - 1 е 1 и това няма как да се промени
- Booleans
  - Истината е истина и лъжата е лъжа
- Символи
  - ‘a’ не може да се промени

Променливи

- Ядрото на информатиката
- Обратно на константите, потребителят избира каква стойността на променливата
- Потребителят избира типът
- Потребителят може да променя стойността

Синтаксис(не важи за enum)

- <тип> <име на променливата>; - декларация
- <тип><име на променливата> = <стойност>; - декларация + инициализация

- !!!Неинициализираните променливи са потенциален източник на грешки, бъдете много внимателни с тях!!!

Именуване на променливи

- Препоръчително от 1 до 31 символа, някои компилатори работят и с по-дълги имена
- Имената трябва да започват с латинска буква, $ или _, след първия символ може да има и цифри
- Има разлика между главни и малки букви
- Някои имена са забранени, защото вече са запазени
- След като едно име бъде употребено то също става запазено за съответното поле

Примери

```
int $php = 5;
double _$_ = 3;
char nayqkiqsimvolever = ‘M’;
float MnOgOqKo = 2.4;
bool _ = false;
long __ = 5234;
long long $$ = 42342;
unsigned gjnueirgeri = 453;
```

Важно!!!

- Всички примери от предния слайд работят

- **Не правете така!!!**

- Това е много лоша практика и може да ви струва скъпо!

Именуване на променливи – добри практики

- Когато пишете код и използвате променливи, които имат важна роля е важно да ги именувате така, че да е ясно каква роля имат!

- Не е добре променливите ви да имат много дълги имена

- Не е препоръчително да пишете на шльокавица, защото един ден кодът ви може да се чете от човек, който не говори български

Примери

```
int age = 5;
double sum = 3;
char coolestSymbol = ‘M’;  //camelCase
float MyDegree = 5.6; //PascalCase
long remaning_slides = 5234; //space replacement
bool isOk= true; //добра практика за именуване на bool
bool hasBlueEyes = false;
```

Представяне на константа в бройна система, различна от десетичната

- За да използвате шестнайсетична бройна система, трябва преди числото да напишете 0x.
  - Например, за да присвоите на променлива a стойност 255, използвайки шестнайсетичното и представяне (FF), трябва да напишете int a = 0xFF.
- За да използвате двоична бройна система, от C++14 съществува и опцията 0b
  - Например, за да присвоите на променлива a стойност 5, използвайки двоичното и представяне (101), трябва да напишете int a = 0b101.

Константни променливи

- const <тип><име на променливата> = <стойност>;
- Стойността се задава при инициализация и не може да се променя след това
- Неинициализирана константа - undefined behaviour

Примери

```
const int number; //неинициализирана константна променлива,
                  //повечето компилатори ще изведат грешка

//правилно създаване на константа
//добра практика е имената на променливи да са изцяло с главни букви
const int MY_NUM = 2;
const char TAB = '\t';
const double PI = 3.14;
```

Задачи за вас #1

- Отидете на www.menti.com

Задача

- Колко памет ще се задели в следния случай?

```
unsigned a, b, c=a;
```

- Използването на неинициализирана променлива е недефинирано поведение!!! => приемаме, че няма да се компилира=>0

Задача

- Ще се компилира ли следният код?

```
const int first = 55;
int second = first;
second = 40;
```

- Отговор: да, защото по никакъв начин не променяме стойността на first

Задача

- Как според вас се определя колко най-малко бита са необходими, за да се представи в двоичен вид дадено число?

- Отговор: Ако числото е положително, намираме най-близката степен на 2, **по-голяма** от даденото число. Ако числото е отрицателно, намираме най-близката степен на 2, **по-голяма или равна** на даденото число и добавяме още 1 бит за знака.
- //1 бит за знака е излишен при положително число, защото търсим оптимално представяне

Супер сте!

- 10 минути почивка

- Ако искате можем да започнем по-рано, за да свършим по-късно. ☺

Оператори връщащи стойност

- Команди, които имат специални изисквания
- Запазен символ или комбинация от символи
- Съществува йерархия на операторите

Типове оператори

1. Comma Operator – в бонус материалите
2. Mathematical Operators
3. Assignment Operator
4. Logical Operators
5. Relational Operators
6. Bitwise Operators
7. Shift Operators
8. Unary Operators
9. Ternary Operator

Приоритет на операторите

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Аритметични оператори

- + събиране 5+3=8
- - изваждане 5-3=2
- *умножение 5*3 = 15
- /деление 15/3=5
- % остатък при деление 5%3 =2

- Резултатите са константи

Относно %

- Изглежда абстрактен, но е изключително важен

- Приложения
  - Намиране на последна цифра 15%10=5
  - Проверка дали дадено число се дели на друго
  - Проверка за четност 23%2=1 => не е четно

- Не може да дели променливи с плаваща запетая

Оператор за присвояване

- <променлива> = <израз>;
- <lvalue> = <rvalue>;

- <lvalue> — място в паметта със стойност, която може да се променя
  - Пример: променлива

- <rvalue> — временна стойност, без специално място в паметта
  - Пример: константа, литерал

Оператор за присвояване

- Дясноасоциативна операция!

a=b=c=d=f=4;

a=(b=(c=(d=(f=4)))) – какво всъщност се случва

(a=b)=(c=(d=f))=4 – грешно

Задачи за вас #2

- Отидете на www.menti.com

Задача

```
int a=1,b=2,c=3,d=4,f=5;
a=b=c=d=f=4;

std::cout<<a;
```

>>4

Задача

```
int a = 5, b = 3;
(a = b) = a + 3;

std::cout <<a<<„, "<<b;
```

>> 6, 3

Задача

```
int a = 5, b = 3;
(a = b) = a + 3;
```

1. a=b е с приоритет заради скобите
2. a вече има стойност 3
3. a+3 = 6
4. a = 6

Смесени оператори за присвояване

- int a=0;
- += събиране a+=8 // a = a + 8
- -= изваждане a-=2 // a = a - 2
- *=умножение a*= 15 // a = a * 15
- /=деление a/=5 // a = a / 5
- %= остатък при деление a%=3 // a = a % 3

Логически оператори

- Работят само с булеви стойности
- && - дали и двете страни са истина
- || - дали поне една от двете страни е истина
- ! - Обратното на булевата стойност вдясно

Таблица на логическите операции

A	B	A&&B	A||B	!A	!B
True	True	True	True	False	False
False	True	False	True	True	False
False	False	False	False	True	True
True	False	False	True	False	True

Начин на работа на && и ||

```
bool error(){
        abort(); //функция, която гърми
        return true;
}
int main(){
false && error();

std::cout << "Did it explode?\n";
}
```

Начин на работа на && и ||

```
bool error(){
        abort(); //функция, която гърми
        return true;
}
int main(){
true && error();

std::cout << "Did it explode?\n";
}
```

Начин на работа на && и ||

```
bool error(){
        abort(); //функция, която гърми
        return true;
}
int main(){
true || error();

std::cout << "Did it explode?\n";
}
```

Начин на работа на && и ||

```
bool error(){
        abort(); //функция, която гърми
        return true;
}
int main(){
false || error();

std::cout << "Did it explode?\n";
}
```

Начин на работа на && и ||

- Ако левият bool на && е false, && връща false, без да проверява десния
- Ало левият bool на || е true, || връща true, беаз да проверява десния

Релационни оператори

- Служат за сравняване на 2 стойности
- == дали двете са равни
- != дали двете са различни
- >   дали лявата е по-голяма от дясната
- <   дали лявата е по-малка от дясната
- >= дали лявата е по-голяма или равна на дясната
- <= дали лявата е по-малка или равна на дясната

- При сравняване на литерал, ще се вземе числената им стойност (виж ASCII таблицата)

Единични оператори

- Работят само с 1 променлива
- Примери:

  ++

  --

  +

  -

- Други, които днес няма да разглеждаме, но скоро ще ви вкарат в ада: &, *, new, delete

++ и --

- Ще разгледаме само ++, всичко при -- е аналогично

- Има два различни оператора ++
  - Prefix (пред името на променливата)
  - Postfix (след името на променливата)

- И двата увеличават стойността на променливата с 1, но връщат различни резултати

Разлика между prefix и postfix

- Prefix връща стойността на променливата след инкрементиране
- Postfix връща стойността на променливата преди инкрементиране

- Prefix връща lvalue
- Postfix връща rvalue

Оператори + и – (отново)

- + и – могат да се използват и само върху един елемент

- Единичните оператори +/- влияят върху знака на числовите стойности

- int a = 50;
- a = -a; //a ще придобие стойност -50
- a = +a; //a ще си запази стойността

Оператор ? :

- Основен синтаксис:

<булева стойност> ? <израз 1> : <израз 2>

- Ако <булева стойност> е true, операторът ще върне <израз 1>, иначе ще върне <израз 2>

- !!!<израз 1> и <израз 2> трябва да са от един и същ вид, иначе ще се получи грешка при компилация

Примери с оператор ? :

- int a = 1 ? 2 : 3; // a ще получи стойност 2, зле написан код
- int a = (0 ? 2 : 3); // a ще получи стойност 3, четлив код
- int a = 5 + (5<4 ? 2 : 3); //a ще получи стойност 8
- Принтиране на по-малка променлива

```
std::cout<<( a < b ? a  : b );
```

Задачи за вас #3

- Отидете на www.menti.com

Задача

```
int a = 5, b=0;
++a += ++b * a;
std::cout<<a;
```

- Ще се компилира, защото ++a е lvalue, а += е оператор за присвояване

Задача

```
int a = 5, b=0;
++a += ++b * a;
std::cout<<a;
```

1. a ще стане 6 заради префикса
2. b ще стане 1
3. b умножено по стойността на a ще даде 6
4. към a ще добавим 6
5. a ще стане 12

Задача

```
int a=5, b=4;
 std::cout<<((2*b-- > --a &&  a-- == b++)?“Algebra":“Geometry");
```

- “Algebra” u “Geometry” очевидно са от един и същ тип => няма основания, за грешка при компилация

Задача

1. a ще стане 4
2. 2*4 > 4 очевидно е вярно
3. b ще стане 3
4. 4==3 очевидно грешно
5. && ще върне false
6. a ще стане 3
7. b ще стане 4
8. Ще се изпише Geometry

Свършихме с операторите!!!

- Почти по средата сме! ☺

Коментари

- Казват на компилатора, да прескочи текста, който заграждат
- Коментарите са в помощ на програмиста и най-вече на тези, които могат един ден да четат кода му
- Слагайте коментари винаги, когато очаквате някой да чете кода ви
- Липсата на коментари води до понижаване на качеството на кода

Коментари

- В C++ има два основни типа коментари

- Еднолинейни

  // - казва на компилатора да не компилира повече на този ред

- Многолинейни

  /* казва на компилатора да не компилира докато не срещне */

Пространства (scope)

- Пространствата се използват за абстрахиране на данни от настоящото поле или за изпълнение на няколко команди с едно повикване
- Синтаксис {……}
- В {…..} важат всички правила на C++
- Всички данни извън {…..} се пренасят в него
- Можем да декларираме нова променлива с име като на такава извън {….} и тогава {……} забравя за старата

Задачи за вас#4

- Отидете на www.menti.com

Задача

```
int a = 0;
{
    int a = 5;//нова променлива
     ++a;
}
```

- std::cout<<a;//първата а не е била променяна => 0

Задача

```
int a = 0;
{
    ++a;//променя оригиналната a
    int a = 5;
}
```

- std::cout<<a;//1

Задача

```
int a = 0;
{
    ++a;
    int a = 5;//нова променлива
    std::cout<<a;//извеждаме новата променлива =>5
}
```

Супер сте!

- 10 минути почивка

- Не се тревожете, най-лошото тепърва предстои! ☺

Условен оператор

- Синтаксис: if (<условие>) <действие1> [else <действие2>]

- Възможно е <действие2> да бъде друг условен оператор

if (<условие1>) <действие1> else if (<условие2>) <действие2> [else <действие3>]

Примери за условен оператор

- int x,y;
- std::cin>>x>>y;

- if(x<y) std::cout<<“x is lower than y”;else;
- Горното е равносилно на if(x<y) std::cout<<“x is lower than y”;

- if(x<y) std::cout<<“x is lower than y\n”;
  else if(x>y) std::cout<<“x is higher than y\n”;
  else std::cout<<“x is equal to y\n”;

Scope като команда на условен оператор

- if(a<b)

{

//Many lines code here

}

Вложени условни оператори

- if (a > 0) if (b > 0) std::cout << 1; else std::cout << 3;

- Какво ще стане ако a е -1?

```
if (a > 0){
        if (b > 0){
                std::cout << 1;
        }
}
else{
        std::cout << 3;
}
```

или

```
if(a>0){
        if(b>0){
                std::cout<<1
        }
        else{
                std::cout<<3;
        }
}
```

Вложени условни оператори

- if (a > 0) if (b > 0) std::cout << 1; else std::cout << 3;

- Какво ще стане ако a е -1?

```
if(a>0){
        if(b>0){
                std::cout<<1
        }
        else{
        std::cout<<3;
        }
}
```

Добри практики при писане на условен оператор

- `if (a > 0) if (b > 0) std::cout << 1; else std::cout << 3;`

```
if (a > 0)
{
        if (b > 0)
        {
                std::cout << 1;
        }
        else
        {
                std::cout << 3;
        }
}
```

Оператор за многозначен избор

- int a = 10;

```
if(a==0)
        std::cout<<“nula”;
else if (a==1)
        std::cout<<“edno”;
else if(a==2)
        std::cout<<“dve”;
……..
else if(a==10)
        std::cout<<“deset”;
```

Оператор за многозначен избор

- Когато искаме да имаме няколко различни изхода в зависимост от стойността на 1 променлива, if ; else невинаги е оптимален

- switch (<променлива>) {

{ case <стойност> : { <действие> } }

[default : { <действие> }]

}

Оператор за многозначен избор

- switch(a)

```
{
        case 1:  std::cout<<“edno”;
        case 2:  std::cout<<“dve”;
        …….
        case 10: std::cout<<“deset”;
        default: std::cout<<“too big”; //може да е навсякъде в тялото и
                                       винаги ще се разглежда последен
}
```

Оператор за многозначен избор

- Какво не е наред с кода от предния слайд?

- При въвеждане на 1 ,например, изходът ще бъде:

ednodvetrichetiripetshestsedemosemdevetdesettoobig

(оцветяването е само с цел да се чете по-лесно)

Оператор за прекъсване

- Ключова дума: break
- Операторът за прекъсване break има две функции
- Когато се използва в switch-statement, при извикването си казва на програмата да спре да изпълнва командите от switch
- Няма смисъл да се слага след последния случай
- Втората функция ще разгледаме по-късно

Оператор за многозначен избор

```
switch(a)
{
        case 1:  std::cout<<“edno”; break;
        case 2:  std::cout<<“dve”; break;
        …….
        case 10: std::cout<<“deset”;break;
        default: std::cout<<“too big”;
}
```

Задачи за вас#5

- Отидете на www.menti.com

Задача

Какво ще се изведе на конзолата?

```
int a = 0;
switch(a)
{
        case 1: std::cout<<“edno”;break;
        default:4==2+2;
        case 2: std::cout<<“dve”; std::cout<<“I wont forget to break”;break;
}
```

Отговор: dveI wont forget to break

Задача

Какво ще се изведе на конзолата?

```
int a = 2;
switch(a)
{
        case 1: std::cout<<“edno”;break;
        case 2: int b = a + 5; std::cout<<b;break;
        default:break;
}
```

- Отговор: грешка при компилация

Задача

- За да се инициализират променливи в случай на switch, трябва да се създаде нов scope, в който да се инициализира новата променлива

```
switch(a)
{
        case 1: std::cout<<“edno”; break;
        case 2: {int b = a + 5; std::cout<<b;} break;
        default:break;
}
```

- ErrorC2360 initialization of 'b' is skipped by 'case' label

Задача

- Ако обаче случаят, в който се инициализира нова променлива е последен, то тогава не се получава грешка при компилация

```
int a = 2;
switch (a)
{
        case 1: std::cout << 1; break;
        default:break;
        case 2: int b = a+5; std::cout << b; break;
}
```

Ще изведе 7

Задача

Какво ще се изведе на конзолата?

```
int a =2;
if(a==2) std::cout<<“dve”;
if(a != 3) std::cout<<“not3”;
else if (a<3) std::cout<<“small”;
else std::cout<<“big”;
```

- Отговор dvenot3

Какво ще се изведе на конзолата?

```
int a =2;
if(a==2) {
        std::cout<<“dve”;
}
if(a != 3) {
        std::cout<<“not3”;
} else if (a<3) {
        std::cout<<“small”;
} else {
        std::cout<<“big”;
}
```

Цикли

- Случвало ли ви се е да ви накажат да напишете „Няма да говоря в час“ 1000 пъти?

- Колко по-лесно е веднъж да напишете:

```
for(unsigned i =0; i<1000; ++i)
        std::cout<<“Няма да говоря в час\n”;
```

Цикли

- Цикъл е операция, която се повтаря N на брой пъти
- Спестява ръчното писане на един и същ код
- Едно от най-мощните оръжия в програмирането

Оператор while

- Синтаксис:

```
while(<условие>)
    <команда>;
```

- Докато <условие> е истина ще се изпълнява <команда>

- Силно препоръчително е в <команда> да се извършва действие, което да превърне <условие> в лъжа!

Оператор while

```
while(true)
        std::cout<<“I love C++\n”; //безкраен цикъл


unsigned i =0;
while(i<1000)
{
        std::cout<<i; //ще изпише числата от 0 до 999
        ++i;
}
```

Композиция do...while

- Синтаксис:

```
do
<команда>;
 while(<условие>);
```

- Първо ще се изпълни <команда>, а после докато <условие> е истина ще се изпълнява <команда>

- Същата идея като while, но гарантирано цикълът ще се изпълни поне веднъж

Оператор break

- Ако break се извика в цикъл, цикълът се прекъсва.

- unsigned i = 0;
- while(true)
  if(i==99)
    break; //ето как можем да счупим безкраен цикъл след 100 стъпки
  else
    ++i;

Оператор continue

- Оператор continue прекратява командата в цикъла

- За разлика от оператор break, оператор continue не прекратява цикъла, а само командата, като цикълът продължава все едно нищо не се е случило

- Пример: принтиране на всички четни числа от 1 до 1000

```
int counter = 0;
while(counter<1000)
{
++counter;
if(counter%2!=0)
        continue;
std::cout<<counter<<‘ ‘;
}
```

Оператор for

- Синтаксис: for(<израз1>; <условие>; <израз 2>)<тяло>

- Работа:

1. Изпълнява се <израз1>
2. Проверява се <условие>
3. Ако <условие> е истина се изпълнява <тяло>, ако не е излизаме от цикъла
4. Изпълнява се <израз2>
5. Връщаме се в 2.

Оператор for

```
for(unsigned i =0; i<1000; ++i)
        std::cout<<“Няма да говоря в час\n”;

for(unsigned i =0; i<1000;++i)
{
        if(i%2 == 0) //ако трябва да се редуват две изречения
                std::cout<<“Няма да говоря в час\n”;
        else
                std::cout<<“Ще слушам класната\n”;
}
  //можеше просто двете изречения да се обединят в едно
```

Вложени цикли

- Реален пример, който използвах наскоро. Исках да си изведа всички числа от 1 до 100 по хиляда пъти.
- Но как да ги изведем без да ги пишем ръчно?

```
for(unsigned i =1; i<=100; ++i)
        for(unsigned j = 0; j<1000; j++)
                std::cout<<i<<‘ ‘;
```

Работещи, но странни и непрепоръчителни употреби на for

- unsigned i = 0;

```
for(;  i<1000; ++i)
        std::cout<<“Няма да говоря в час\n”;
```

- unsigned i = 0;

```
for(;  i++<1000;)
        std::cout<<“Няма да говоря в час\n”;
```

Работещи, но странни и непрепоръчителни употреби на for

- unsigned i = 0;

```
for(;  ;){
        std::cout<<“Няма да говоря в час\n”;
        if(++i==1000)
                break;
}
```

Работещи, но странни и непрепоръчителни употреби на for

```
for(unsigned i =0; i<1000; std::cout<<“Няма да говоря в час\n”, ++i);
```

- Всичките тези примери имат една и съща функция, НО не е добра практика да ги ползвате!!!
- Ако няма да използвате for така както трябва, то помислете дали while не е по-добра алтернатива
- Ако на контролното имате код като горните, просто си обърнете цикъла от for в while

Примерно преобразуване

- for(unsigned i = 0; i<15; std::cout<<i<<“\n”, i+=3, --i);

- Преобразуване:

```
unsigned i = 0;
while(i<15)
{
        std::cout<<i<<“\n”;
        i += 3;
        --i;
}
```

- След преобразуване четимостта е в пъти по-голяма

Задачи за вас#6

- Отидете на www.menti.com

Задача

- Какво ще се изведе на конзолата?

```
int a = 0;
for(unsigned i = 5; i>=0; ++i)
        a+=i;
std::cout<<a;
```

- Отговор: Безкраен цикъл, защото i само ще нараства и никога няма да стане по-малка от 0

Задача

Ще се изведе ли нещо на конзолата?

```
for(unsigned i = 0; i<10; ++i)
{
        for(unsigned j =i; j<50; ++j)
                if(j == 25)
                        break;
    std::cout<<i;
}
```

- Отговор: да, 10 пъти i (0123456789), защото break ще счупи само вложения цикъл, но не и главния

Задача

- Колко пъти ще се изпише “Няма да говоря в час”?

```
unsigned i = 0;
for(; i++ ;){
        std::cout<<“Няма да говоря в час\n”;
        if(i>=1000)
                break;
}
```

Отговор: 0, защото i++ връща стойност 0=>false=>цикълът ще приключи преди да е започнал

Супер сте!

- 10 минути почивка

- Нощуването във ФМИ е тежко само първия път ! ☺

Функции

- Парче код, което изпълнява някакво действие
- Може да връща даден резултат
- Един и същ код може да се използва многократно
- Освобождава място в main()

Синтаксис

- <сигнатура> <идентификатор> ([<формални_параметри>]){ <тяло> }

- <сигнатура> ::= [<тип_резултат> | void]
- void = празен тип, не връща резултат

- Ако типът на резултата се пропусне, подразбира се int
- Ако функция със сигнатура различна от void не връща стойност се получава грешка при компилация

Синтаксис

- <сигнатура> <идентификатор> ([<формални_параметри>]){ <тяло> }

- Ако има формални параметри, то трябва да се специфицира типът им
- Ако параметър е примитивен тип данна или някакъв обект, то се създава нов обект в scope-а на функцията!

Извикване на фунцкия

- <име>([<фактически_параметри>]);

- Извикването на функция всъщност е операция с много висок приоритет
- Типът на фактическия параметър се съпоставя с типа на съответния формален параметър
- Ако се налага, прави се преобразуване на типовете <формален_параметър> = <фактически_параметър>

Връщане на резултат

- return [<израз>];
- Оператор за връщане на резултат на функция
- Типът на <израз> се съпоставя с типа на резултата на функцията ако се налага, прави се преобразуване на типовете
- Работата на функцията се прекратява незабавно
- При сигнатура void, return не връща нищо, а просто прекъсва функцията(не е задължителен)

Примери

- Функция намираща сбора на 5 числа

```
int Sum (int a, int b, int c, int d, int e)
{
	return (a+b+c+d+e);
}
```

или

```
int Sum (int a, int b, int c, int d, int e)
{
	int temp = a+b+c+d+e;
	return temp;
}
```

Примери

```
bool ValidateData(int a)
{
	if(a>=1000)
	{
		return true;
	}
	if (a%2 != 0)
	{
		return false;
	}
}
```

//Грешка при компилиране(undefined behaviour), защото не всички възможни изходи връщат стойност

Q&A

- Q: Примерите дотук изглеждат тривиални и прекалено лесни, за да се наложи да използваме функция. Какво ще стане ако просто си ги въвеждам всеки път?

- A1: Кодът ти ще е претрупан с повтарящи се фрагменти, а това намалява качеството на кода. Некачественият код е за други ВУЗ-ове.
- A2: Ако решиш да промениш нещо ще трябва да пренаписваш кода навсякъде. Това е загуба на време, а и може да е източник на грешки.
- A3: Виж в следващия слайд.

Пример

- Алгоритъм за проверка дали едно число е просто

```
bool IsPrime(int a)
{
        if(a<2)
        return false;
        if(a==2)
                return true;
        for(unsigned i = 3; i*i<a; i+=2)
        {
                if(a%i == 0)
                        return false;
        }
        return true;
}
```

Защо не ползвам else if и else?

Пояснение относно параметрите

- Не са задължителни
- Създават се нови обекти, като при примитивните типове данни това е много бърза операция, но при някои други данни може да е много бавно
- Новите обекти са равни на оригиналните, но не са свързани с тях
- Ако промените временен обект във функцията, оригиналният не се променя

Пояснение относно параметрите

- Подредбата им е от значение
- Добра практика е да са константни ако нямаме намерение да ги променяме
- В тялото на функцията не може да се създават нови променливи с имена на параметри
- Ако вече сте забравили
- Новите обекти са равни на оригиналните, но не са свързани с тях

Функция разменяща стойностите на 2 променливи

```
void swap(double a, double b)
{
        double c = a;
        a = b;
        b = c;
}
```

- Нищо няма да се случи, защото a и b са нови временни обекти.
- Тези променливи не са свързани с променливите, които сме подали като параметри!!!

Overloading

- Възможно ли е да имам функция, която да прави повече от 1 действие в зависимост от подадените параметри
  - Пример Sum(1,2,3), Sum(1,2,3,4), Sum(1.55,1.2)

- Отговор: да, това се нарича function overloading

Декларация на функция

- <декларация_на_функция> ::= <сигнатура>;
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти, но може да бъде дефинирана само веднъж
- Неизпълнените обещания водят до проблеми...
  - ...освен когато никой не разчита на тях

Декларация на функция - пример

- int abs(int); // името на параметъра не е задължително щом не го
  //ползвате без дефиниране, но не е грешка ако го има

```
…….код……
int abs(int a)
{
        return (a>0) ? a : -a;
}
```

- Този похват се нарича forward declaration, ще го разглеждате по ООП

Overloading

- Една функция може да има безброй много overloads
- При извикване на функцията, компилаторът се грижи да намери правилният overload на функцията
- Компилаторът може да направи преобразуване на данните ако се налага
- Ако не намери подходящ се получава грешка при компилиране
- Ако намери повече от 1 подходящ се получава грешка за двусмислие

Примери за overloading

1. void cout(char a){std::cout<<a;}
2. void cout(int a){std::cout<<a;}
3. void cout(char a, int b){std::cout<<a<<‘-’<<b;}
4. void cout(double a, char b){std::cout<<b<<‘-’<<a;}
5. void cout(bool a){std::cout<<a;}
6. void cout(char a, bool b, int c){std::cout<<a<<b<<c;}
7. void cout(const int a){std::cout<<a;}
8. void cout(char a, unsigned b){std::cout<<a<<‘-’<<b;}
9. char cout(char a){return a;}

Примери за overloading

1. void cout(char a){std::cout<<a;} //двусмислие с 9
2. void cout(int a){std::cout<<a;} //двусмислие със 7
3. void cout(char a, int b){std::cout<<a<<‘-’<<b;} //двусмислие с 8
4. void cout(double a, char b){std::cout<<b<<‘-’<<a;}
5. void cout(bool a){std::cout<<a;}
6. void cout(char a, bool b, int c){std::cout<<a<<b<<c;}
7. void cout(const int a){std::cout<<a;} //двусмислие с 2
8. void cout(char a, unsigned b){std::cout<<b<<‘-’<<a;} //двусмислие с 3
9. char cout(char a){return a;} //двусмислие с 1

Как може да се отстранят тези двусмислия

- void cout(char a, int b){std::cout<<a<<‘-’<<b;}
- void cout(char a, unsigned b){std::cout<<b<<‘-’<<a;}

- void cout(char a, int b){std::cout<<a<<‘-’<<b;}
- void cout(unsigned b, char a){std::cout<<b<<‘-’<<a;}

- Важно! За компилатора има значение подредбата на параметрите. Ако спрямо дадената подредба няма отговаряща функция се получава грешка при компилация

Как може да се отстранят тези двусмислия

- Другите 2 двусмислия няма как да се отстранят така
- Може да се промени името на някоя от функциите
- Може една от функциите да има нов параметър, който да не се използва

```
char cout(char a, bool useless){return a;} //лоша практика
```

Параметри по подразбиране

- Възможно е да имате програма, в която 90% от случаите подавате един и същ параметър на дадено място

- C++ позволява да имате стойност по подразбиране за 1 или повече параметри, които не се налага да уточнявате при извикване на функцията

Параметри по подразбиране

- Синтаксис: void Cout(int a, int b = 5){std::cout<<a<<‘ ‘<<b;}

Cout(4); //4 5
Cout(3,6); //3 6

- Параметрите по подразбиране трябва винаги да са в края!!!

- void Cout(int a){}; //ще се получи двусмислие

Параметри по подразбиране

- void Cout(int a, int b = 5, char c = ‘t’){std::cout<<a<<‘ ‘<<b<<‘ ‘<<c;}

Cout(4); //4 5 t

Cout(3,6); //3 6 t

Cout(3, ‘0’);//3 48 t

- ‘0’ има стойност 48 в ASCII => компилаторът го разглежда като int със стойност 48
- Параметрите по подразбиране винаги са в последователността, в която са дефинирани, не могат да се прескачат

Стекова памет

- Извикването на функция е с много висок приоритет
- Но какво става ако се извика функция в тялото на друга функция?
- Коя функция ще е с по-голям приоритет?
- Отговор: Тъй като извикването на функция е с много висок приоритет, ако в тялото на някоя функция извикаме друга, то втората ще е с по-голям приоритет и след като се изпълни ще се върнем в предната.

Стекова памет

- Какво е стек?
  - Съставна структура от данни, за която ви е още рано?
  - Информация за това какво е стек има включена в бонус материалите

- Как да си обясня стекова памет тогава?

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.

Задачи за вас#7

- Отидете на www.menti.com

Задача

- Какво ще се изведе на конзолата?

```
float Rational(int a, int b)
{
        return a/b;
}
std::cout<<Rational(6.4, 1.6);
```

- Отговор: 6, защото 6.4 ще стане се обърне в int и ще стане 6, а 1.6 ще стане 1 => 6/1 = 6

Задача

- Ще се получи ли грешка при компилация и ако да защо?

```
void empty(){};      //дефиниция
void empty();        //декларация след дефиниция
```

- Отговор: не, позволено е да имаме декларации и след дефиницията

Задача

- Какво ще изведе следната програма?

```
void abs(int a)
{
	if(a<0)
		a*=-1;
}
int numb = -5;
abs(numb);
std::cout<<numb;
```

- Отговор: -5, преговорете си частта със създаването на нови обекти

Време за въпроси

- Дано е останало такова

Благодаря ви за вниманието!

- Поне по моя преценка тази презентация съдържа всичко необходимо, за да изкарате максимален брой точки на контролното
- Отделете време за специфичните неща, които могат да ви се паднат
- Успех на контролното!

*Ако тествате код от презентацията имайте предвид, че Power Point преобразува ‘ ‘ в някакъв символ, който не се разпознава от C++, затова ако имате проблеми с кода, просто заменете тези символи с истински ‘ ’

Източници

- Отворени разработки на доц. Трифонов
- Голяма част от информацията е сверена с https://en.cppreference.com
- Авторският код е проверяван на VisualStudio2017

Бонус материали

- Следващите слайдове са като бонус, като материалът в тях все още не е изучаван

Важни неща относно ASCII на този етап

- Може да се извършват математически операции със символи
  (търпение, скоро ще дефинираме и какво са мат. операции)

- За да преобразувате символ число в число, от символа трябва да извадите 48 или символа ‘0’
  - ‘9’ – 7 = 50, защото ‘9’ има числена стойност 57
  - ‘9’ – ‘0’ – 7 = 57 – 48 – 7 = 2

- Главните букви са преди малките
- Разстоянието между малка и главна буква е 2^5 = 32

Изборен тип

- Ключова дума enum
- Стойност $[-2^{31};2^{31}-1]$
- Стойностите на променливите могат да бъдат явни и неявни
  - enum colours {red, blue, yellow}//red = 0, blue = 1, yellow = 2
  - enum subject {DIS = 6, Algebra = 15, DSTR}//DSTR = 16

Задача

- Какво ще изведе на конзолата следният код?

```
enum food {apple = 3, bread = 2, orange};
std::cout<<orange;
```

- 3, няма проблем две члена на 1 enum да имат равни стойности

Задача

- Ще се компилира ли следният код?

```
int main(){
enum age {Medieval, ModernTimes, age};
return 0;
}
```

- Ще се компилира, но няма да може да се създават променливи от тип age, защото компилаторът ще се обръща към члена age, а не към типа на променливата.

Запазени думи в C++

- https://en.cppreference.com/w/cpp/keyword

Оператор ,

- Сложен за обяснение на техническо ниво
- Лесен за обяснение на интуитивно ниво

- Използва се при изреждане

- char a, b, c, d; //деклариране на 4 променливи

- int a=7, b=1, c=4, d=12; //деклариране и инициализиране на 4 променливи

Нещо екзотично

```
int m, n=1;
m = (cout << n, n); //m=1
```

```
int n = 1;
int m = (++n, std::cout << "n = " << n << '\n', ++n, 2*n);
std::cout << "m = " << (++m, m) << '\n’;
```

n=2

m=7

Пояснение

```
int n = 1;
int m = (++n, std::cout << "n = " << n << '\n', ++n, 2*n);
std::cout << "m = " << (++m, m) << '\n’;
```

1. n става 2
2. извеждаме n
3. n става 3
4. n става 6
5. присвояваме стойност 6 на m
6. m става 7
7. Извеждаме m

C++ shell

cpp.sh
online C++ compiler

about cpp.sh

```
1  // Example program
2  #include <iostream>
3  #include <string>
4
5  int main()
6  {
7      int a = 0;
8      ++a = 12 + a++;//ако е = е недефинирано поведения, понеже не се знае кое е с по-голям приоритет, а на vs е друг отговорът- 14
9      std::cout << a;
10 }
11
```

8:19: warning: operation on 'a' may be undefined [-Wsequence-point]

Get URL

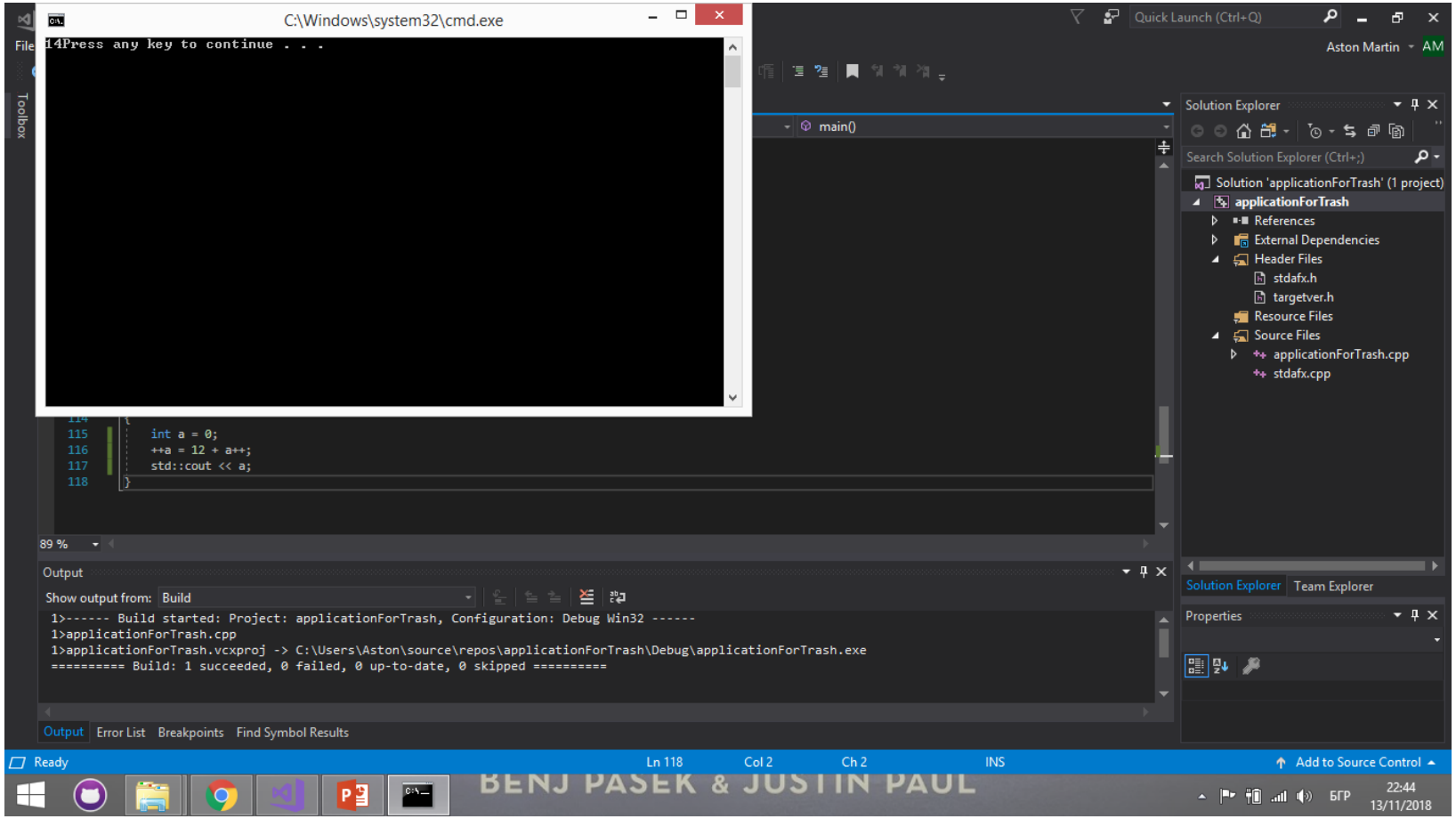
Run

options | compilation | execution

```
13

Program terminated (signal: SIGKILL)
```

https://en.cppreference.com/w/cpp/language/eval_order



```
a && b
a || b
a ? b : c
a , b
```

Undefined behavior

1) Between the previous and next sequence point a scalar object must have its stored value modified at most once by the evaluation of an expression, otherwise the behavior is undefined.

```
i = ++i + i++; // undefined behavior
i = i++ + 1; // undefined behavior (until C++17)
i = ++i + 1; // undefined behavior (until C++11)
++ ++i; // undefined behavior (until C++11)
f(++i, ++i); // undefined behavior (until C++17)
f(i = -1, i = -1); // undefined behavior (until C++17)
```

2) Between the previous and next sequence point, the prior value of a scalar object that is modified by the evaluation of the expression, must be accessed only to determine the value to be stored. If it is accessed in any other way, the behavior is undefined.

```
cout << i << i++; // undefined behavior (until C++17)
a[i] = i++; // undefined behavior (until C++17)
```

Defect reports

The following behavior-changing defect reports were applied retroactively to previously published C++ standards.

DR	Applied to	Behavior as published	Correct behavior
CWG 1885	C++14	sequencing of the destruction of automatic variables on function return was not explicit	sequencing rules added

References

- C++11 standard (ISO/IEC 14882:2011):
  - 1.9 Program execution [intro.execution]
  - 5.2.6 Increment and decrement [expr.post.incr]
  - 5.3.4 New [expr.new]
  - 5.14 Logical AND operator [expr.log.and]
  - 5.15 Logical OR operator [expr.log.or]
  - 5.16 Conditional operator [expr.cond]

УСИ - Google Drive | Facebook | Курс: Увод в софтуерно | (1) Metallica - Where | C++ Shell | Order of evaluation - cp
https://en.cppreference.com/w/cpp/language/eval_order
Zamunda.NET | YouTube | facebook | Solve SI-Practice-5 | Покемон Епизод 74 | Other bookmarks

19) In a shift operator expression E1<<E2 and E1>>E2, every value computation and side-effect of E1 is sequenced before every value computation and side effect of E2
20) In every simple assignment expression E1=E2 and every compound assignment expression E1@=E2, every value computation and side-effect of E2 is sequenced before every value computation and side effect of E1
21) Every expression in a comma-separated list of expressions in a parenthesized initializer is evaluated as if for a function call (indeterminately-sequenced)

Undefined behavior

1) If a side effect on a scalar object is unsequenced relative to another side effect on the same scalar object, the behavior is undefined.

```
i = ++i + 2;       // undefined behavior until C++11
i = i++ + 2;       // undefined behavior until C++17
f(i = -2, i = -2); // undefined behavior until C++17
f(++i, ++i);       // undefined behavior until C++17, unspecified after C++17
i = ++i + i++;     // undefined behavior
```

2) If a side effect on a scalar object is unsequenced relative to a value computation using the value of the same scalar object, the behavior is undefined.

```
cout << i << i++; // undefined behavior until C++17
a[i] = i++;       // undefined behavior until C++17
n = ++i + i;      // undefined behavior
```

Sequence point rules (until C++11)

Definitions

Evaluation of an expression might produce side effects, which are: accessing an object designated by a volatile lvalue, modifying an object, calling a library I/O function, or calling a function that does any of those operations.

A *sequence point* is a point in the execution sequence where all side effects from the previous evaluations in the sequence are complete, and no side effects of the subsequent evaluations started.

Rules

1) There is a sequence point at the end of each full expression (typically, at the semicolon).

2) When calling a function (whether or not the function is inline and whether or not function call syntax was used), there is a sequence point after the evaluation of all function arguments (if any) which takes place before execution of

BENJ PASEK & JUSTIN PAUL

- https://stackoverflow.com/questions/4176328/undefined-behavior-and-sequence-points
- Има готино обяснение за всичките версии

Масиви

- Масивът е съставен тип данни
- Представя крайни редици от елементи
- Всички елементи са от един и същи тип
- Позволява произволен достъп до всеки негов елемент по номер (индекс)

Синтаксис

- <тип> <идентификатор> [[<константа]] [= { <константа> [, <константа>] }] ;
- Примери:
  - bool b[10];
  - double x[3] = { 0.5, 1.5, 2.5 }, y = 3.8;
  - int a[] = { 3 + 2, 2 * 4 }; ⇐⇒ int a[2] = { 5, 8 };

- За всички фенове на Java и C#

bool[10] b; е невалиден израз

Операции за работа с масив

- Достъп до елемент по индекс: <масив>[<цяло_число>]
- Примери: x = a[2]; (rvalue) a[i] = 7; (lvalue!)
- Броенето на индексите започва от 0
- Внимание: няма проверка за коректност на индекса!

Операции за работа с масив

- Няма присвояване a = b
- Няма поелементно сравнение a == b винаги връща false ако a и b са различни масиви, дори и да имат еднакви елементи
- Няма операции за вход и изход std::cin >> a; std::cout << a;
- std::cout << a; извежда адреса на a (не важи за символен низ)

Символен низ

- Описание: Символен низ наричаме последователност от символи последователност от 0 символи наричаме празен низ

- Представяне в C++: Масив от символи (char), в който след последния символ в низа е записан терминиращият символ ’\0’

Относно ‘\0’

- Първият символ в ASCII таблицата, с код 0
- Използва се като прекъсвач(терминатор) от много функции за символни низове, за да се определя края на низа

- Може да се сложи в средата на масив от символи

char a = {‘H’, ‘e’, ‘l’, ‘l’, ’\0’, ’o’}; //символният низ е “Hell”

Символен низ

- Примери:
- char word[] = { ’H’, ’e’, ’l’, ’l’, ’o’, ’\0’ };
- char word[6] = { ’H’, ’e’, ’l’, ’l’, ’o’ };
- char word[100] = "Hello";
- char word[5] = "Hello"; //валиден масив е, но не е символен низ
- char word[6] = "Hello";
- char word[5] = { ’H’, ’e’, ’l’, ’l’, ’o’ };

stack

- Стекът е линейна структура от данни в информатиката, в която обработката на информация става само от едната страна наречена връх. Дъното не е и не трябва да е достъпно. Стековете са базирани на принципа „последен влязъл пръв излязъл“.

- Стекът теоретично може да събере безкраен брой обекти, но на практика само краен брой, ограничен от количеството памет. Обектите могат да се поставят и да се четат (вадят) единствено от горната страна на стека. Стекът има три операции:
- push (добавяне) – поставя нов обект върху стека
- pop или pull (изваждане/изтегляне) – вади най-горния (последно добавения) елемент от стека
- peek (надникване) – показва най-горния елемент от стека без да го изважда

stack – начин на работа