

Лекция 3.

Абстракция на данните. Базови абстракции на данните

*Как парични суми,
бройки, текстове и т.н. се
представят в програмите?*

Данни. Компютърен модел на данните

- **Данни** – това е общо понятие за всичко това, с което оперират изчислителните машини
- *Модели на данните* – процесът на изграждане на адекватен компютърен модел на данните включва създаването на три основни взаимосвързани и взаимозависими модела:
 - абстрактен модел на данните
 - знаков модел на данните
 - физически модел на данните

Абстрактен модел на данните

Абстрактният модел (АМ) на данните определя присъщите им свойства и функционалност, като игнорира детайлите на тяхното физическо представяне

Абстракцията на данни е разделянето на логическите свойства на данните от тяхното конкретно приложение

- Средствата за абстрахиране с цел описание на данните в ЕП се развиват по естествен начин в съответствие с нуждите от такива средства:
 - Формални математически модели (математически структури): реална алгебрична система, вектори, матрици, множества, съжително и предикатно смятане и т.н.
 - Абстрактни модели данни, които са били създадени или уточнени специално за нуждите на информатиката: масив, запис, файл, стек и т.н.
- Видове АМ: прости (*базови абстракции на данните*) и съставни (*структурирани абстракции на данните*) според своята структурност

Знаков модел на данните. Тип на данните в ЕП

Знаков модел (ЗМ) – моделът на данните на ниво език за програмиране, т.е. знаково описание на съответно избраните абстрактни модели, основава се на концепцията за *тип на данните*

- **Тип данни (*data type*)**– същност:
 - основна концепция, заложена в представянето на данните в ЕП
 - основна семантична компонента на ЕП, която обхваща същността на данните
 - осигурява представяне на данните по начин възможно по-естествен за човека
 - улеснява обработката на данните
 - напр. типа на реалните числа (реален тип данни)
- **Тип данни – определение:** Всеки *тип данни* е съвкупност от три множества:
 - **множество от стойности**, които представляват константите на типа, т.е. набор от стойности, които могат да се присвояват на променливи от този тип
 - **множество от операции**, които могат да се извършват над неговите елементи (за реалния тип данни това са събиране, изваждане, умножение и т.н.)
 - **множество от релации** валидни над множеството от стойности на типа (за реалния тип данни това са равно, наредба и т.н.)

Освен това в ЕП за някои типове се поддържа и множество от т.нар. стандартни функции и процедури

Система от типове в ЕП. Силно и слабо типизиране

Система от типове на езиците за програмиране е съвкупността от правила използвани за изграждане и организиране на съвкупността от типове

- **Правила:**

- всеки тип данни трябва да бъде предварително дефиниран
- всеки елемент данни принадлежи на точно един от съществуващите в езика типове данни
- на всеки елемент на данните трябва да бъде присвоен тип
- всеки оператор или операция в езика изисква операнди от определени типове и дава резултат от определен тип



В зависимост от строгостта на спазване на тези правила ЕП се класифицират от **силно типизирани** (Ada, Algol, Modula-2 и Pascal) към **слабо типизирани** (LISP, APL, C++)

- **Предимства:**

- надеждност – може да се провери правилната употреба на типа
- улеснена четливост
- улеснено писане – програмистът може да мисли за числа с плаваща запетая, а не да мисли за числа в един момент, а в следващия да мисли за битове и байтове (абстракция)
- преносимост и изменяемост – ако представянето се промени, не се налага променяне на приложните програми

Физически модел на данните

Физическият модел (ФМ) на данните е начинът, по който стойностите на даден тип данни се съхраняват в паметта на компютъра – място и правила за съхранение, количество заемана памет

- **Същност:**

- Правилното съхраняване на стойностите от различен тип в повечето случаи е **грижа на компилатора** на ЕП
- Пример: C++ разполага стойностите на променливите, декларирани в главната функция на дадена програма (автоматични променливи) в т.нар. стек (*stack*, *stack frame*). Компилаторът резервира съответното количество памет за различните величини (константи и променливи) при срещане на техните декларации


- **Ограничения:**

- Тъй като всеки "жив" ЕП е реализиран на компютър (физическо устройство), то това неизбежно означава, че физическият модел съответства на знаковия и абстрактния модел, но при известни ограничения
- Пример: Множествата на реалните и целите числа са безкрайни, но се реализират като крайни в определен диапазон

Абстракции на данните: метод на изучаване

- Абстрактен модел – определение, приложение и реализация в ЕП
- Физически модел в езика C++
- Знаков модел на данните в езика C++ :
 - Синтаксис за деклариране на типа и на величини от този тип
 - Множество от стойности на типа
 - Операции
 - Релации
 - Стандартни функции


Базови абстракции на данните

 Най-често са абстрактни модели, които се използват масово и в други области на човешкото знание, като например алгебра на реалните числа, алгебра на целите числа и т.н.

 В ЕП те се реализират чрез *прости* (примитивни) *типове данни*

Прости типове данни в ЕП

- **Определение:** *Простите* типове (*simple data types*) са такива, чийто стойности са неделими (атомарни)
- **Видове според начина им на дефиниране:**
 - обикновено са реализирани като *стандартни* (вградени, предварително-дефинирани) *типове* в ЕП – целочислени, реални, символни, логически и т.н.
 - *дефинирани от потребителя* (ЕП предлагат средства, чрез които потребителят може сам да дефинира свои собствени типове)
- **Константите** от прост тип – могат да се задават директно по начина близък до общоприетия:

102 (целочислена)	-2.16 или 0.01E-3 (реални)
true и false (логически)	'a' 'z' '%' 'я' (символни)
- **Видове според природата им** – *посочващи* и *скаларни*
Посочващите типове се отличават от всички останали по това, че стойностите им са адреси от паметта на компютъра
 Тук ще бъдат разгледани само скаларните прости типове

Прости скалярни типове данни в ЕП

- **Определение** – *скалярните* (*scalar*) типове имат две основни свойства:
 - стойностите им са крайно множество и представляват един единствен компонент т.е. може да представят една стойност
 - между стойностите им е определена строга наредба (за тях са определени всички операции за сравнение $<$, $>$, $=$, и т.н.)
- **Приложение:** Когато програмата трябва да обработва прости данни, между които има наредба те могат да се моделират с някой скаларен тип данни. Например парични суми, мерки, поредни номера, азбука и т.н.
- **Видове:** Скалярните типове се делят на две групи според свойството изброимост на множеството от стойностите им, което е определящо по отношение на тяхното приложение при моделиране на данните – *дискретни* и *недискретни*

Прости дискретни типове данни в ЕП

- **Определение** – *дискретните* (*discrete, ordinal*) скаларни типове се отличават от недискретните по това, че стойностите им притежават следните допълнителни свойства:
 - стойностите са подредени линейно и поради това може да им се съпостави пореден номер (числата 0, 1, 2, 3, ... 0 съответства на най-малката стойност, 1 – на следващата и т.н.)
 - за всяка константа от множеството стойности освен за най-голямата е известно коя е следващата, а за всяка константа освен за най-малката е известно коя е предходната
 - стойностите от дискретен тип могат да се преобразуват от един в друг дискретен тип
- **Приложение:** Поради свойствата си, дискретните типове могат да се прилагат във всички случаи, когато е необходимо да се моделират крайни линейно подредени множества данни, да се обхождат елементи според линейната им наредба, да се номерират крайни множества от стойности и т.н.

Система от типове в C++

- **Характеристики:**

- Естествената *система от типове* на C++ не е много богата, както тези на някои други езици за програмиране
- C++ е *по-силно типизиран* от езика C

- **Правила:**

- Всеки тип данни трябва да бъде деклариран, за да може да се използва в програмата – *предварително деклариран* (стандартен) или *декларирани от потребителя* по специални синтактични правила
- *Простите типове* (с малки изключения) в езика са предварително декларирани, като са им съпоставени стандартни имена: `int`, `float`, `double`
- *Синонимни имена* (синоними) на типовете могат да се декларират явно чрез ключовата дума `typedef` (това не е декларация на нов тип):
 - **синтаксис** – не може да се декларират в рамките на дефиниция на функция:
`typedef <декларация на тип> <име-синоним>;`
 - **пример:**
`typedef unsigned long ulong; //синоним ulong`

Деклариране на променливи в C++

Типът на всяка променлива, която се използва в програмата се декларира задължително, чрез **синтаксиса**:

<декларация на тип> <списък имена на променливи>;

Където *<декларация на тип>* е:

- указване името на типа

```
int Myint;
```

```
ulong ul; // Equivalent to "unsigned long ul;"
```

- директна декларация на типа:

```
struct mystructtag
```

```
{ int i; float f; char c;} var_ms;
```

```
typedef struct mystructtag
```

```
{ int i; float f; char c;} mystruct;
```

```
mystruct ms; // Еквивалентно на "struct mystructtag ms;"
```



В някои случаи се допуска използването на т.н. **анонимни типове** (без имена), но при спазване на доста ограничения

Дефиниране на променливи в C++

Дефиницията на променлива може да включва освен елементите от декларацията на една променлива и указание каква памет да бъде заделена за тази променлива, както и присвояване на начална стойност на променливата (инициализация)

- **Синтаксис:**

<декларация на тип> <име на променлива> = <инициализиращ израз> незад ,
...;

- **Декларация на типа :**

- всеки валиден тип от езика;
- `auto` (🔊 от версия C++11) – “**автоматично**” определяне на типа на променливата, но е инициализирана.

- **Примери:** `int x = 10; auto mark = 6;`

- **Правила:**

- при задаване на инициализиращият израз се спазват различни ограничения в зависимост от типа памет, който използва променливата
- в някои случаи ако променливата не е инициализирана, то тя се инициализира автоматично обикновено със стойност 0 (🔊 да не се разчита)

Дефиниране на константни стойности в C++

Типът на константите може да бъде:

- определен от компилатора автоматично

```
cout << "Angle : "; // константа от тип низ
```

```
cout << 180.0; // реална константа
```

- явно деклариран:

- **синтаксис:**

```
const <тип> <име> = <израз>;
```

- **пример:**

```
const double Neutral_pH = 7.0; /* pH на водата */
```

```
const int CREDITMARK = 65;
```

```
const char QUERY = '?';
```

```
const double PI = 3.1415926535898;
```

```
const double SOMENUM = 1.235E75;
```

```
const double DEGREES_TO_RADIANS = PI / 180.0;
```

- **предимства:**

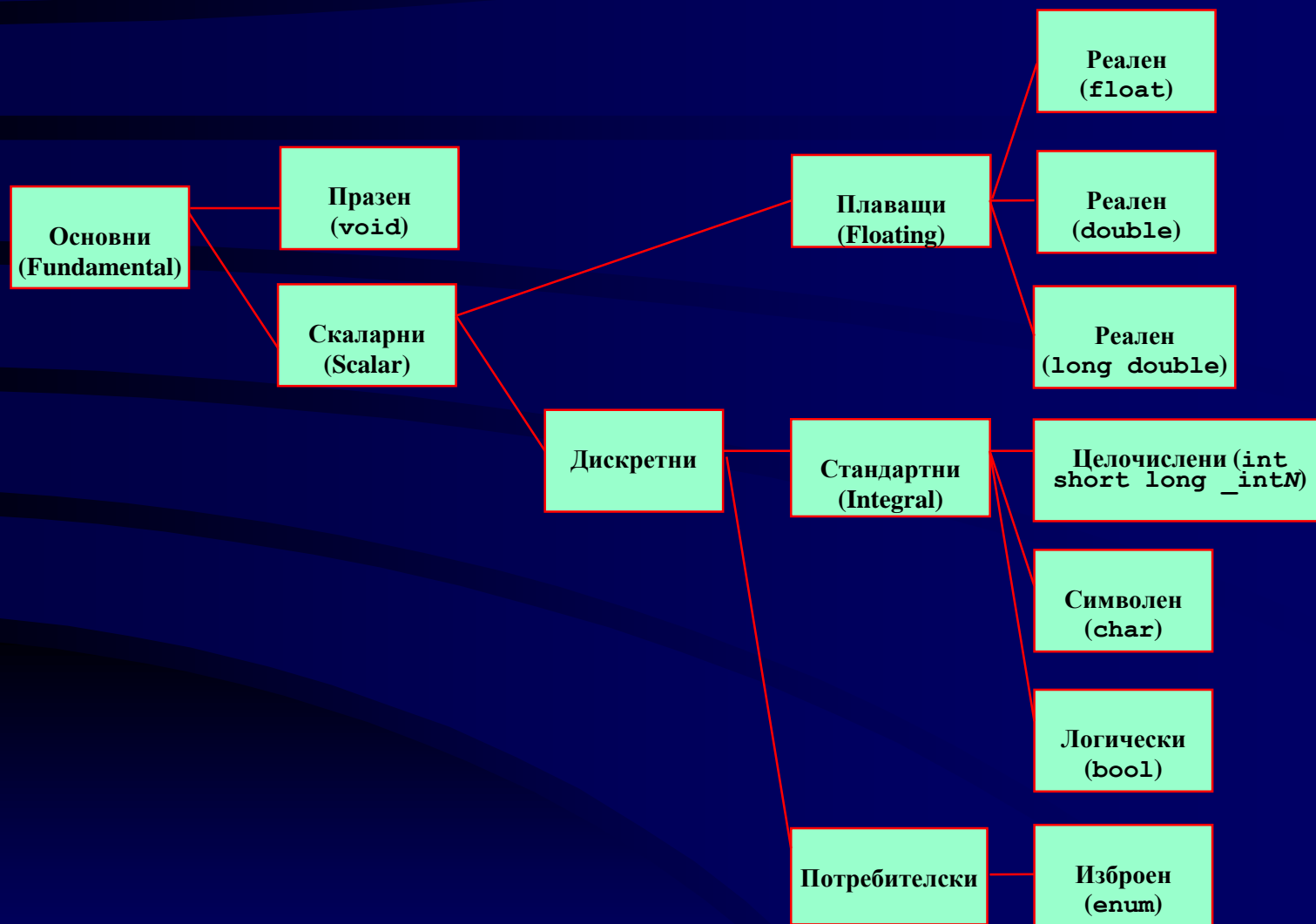
- повишава четливостта на програмата
- улеснява евентуални модификации на програмата



За дефиниране на константи може да се използва и директивата на препроцесора

#define (за деклариране на макроси): #define g 32.0


Прости типове данни в C++: класификация




Прости типове данни в C++ – знаков модел

Ако игнорираме указателите, съществуват само *три основни категории*:

- **целочислени типове** (*integral*) – предназначени за обработка на цели числа (положителни и отрицателни в десетична, също и в осмична и шестнайсетична бройна система – започват съответно с 0 и 0x)
- **типове с плаваща запетая** (*floating*) – предназначени за обработка на числа, които могат да имат дробна част
- **празен, недействителен тип** (`void`) – описва празно множество от стойности

 Не могат да се декларираат променливи от тип `void` (използва се за деклариране на резултата на функции, които не връщат стойност или за деклариране на указатели от произволен или неизвестен тип – “пораждащи”)

 В рамките на първите две категории различните типове се различават само по обхвата на числата, които те представят, което зависи от големината на представянето (броя на използваните байтове/битове) и дали представянето е явно или неявно (`char`, `bool`, `enum`)

Целочислени типове в C++ – знаков модел

- Деклариране и обхват:

Т и п	К р а т к о	Ф и з и ч е с к о п р е д с т а в я н е
<code>int</code>		Цяло число в диапазона $\pm 2,000$ милиона
<code>short int</code>	<code>short</code>	Цяло число в диапазона -32768 to $+32768$
<code>long int</code>	<code>long</code>	Цяло число в диапазона $\pm 2,000$ милиона
<code>__intn</code>		Цяло число с фиксиран размер (в битове). <code>n</code> може да бъде 8, 16, 32 или 64
<code>char</code>		Символ – буква, цифра или друг символ от дефинираното множество от символи (в Microsoft C++, е ASCII)
<code>bool</code>		Логическите стойности <code>true</code> и <code>false</code>

- Към декларацията може да се добави ключовите думи `signed/unsigned` – със/без знак (с изключение на тип `bool`)

- Примери:

```
short aCount; // екв. на short int aCount; signed short aCount
char aGenderFlag; // M = male, F = Female
long FailCount, ECount, DCount, CCount, BCount, ACount;
bool Condition=true;
```

Целочислени типове в C++ – множество от стойности и физически модел

- Множество от стойности – цели числа в определен интервал, могат да се задават в различна бройна система

- `int d = 42;`
- `int o = 052;`
- `int X = 0X2A;`
- `int b = 0b101010; // C++14`

- Физическо представяне:

Т и п	Р а з м е р
<code>char, unsigned char, signed char</code>	1 byte
<code>short, unsigned short</code>	2 bytes
<code>int, unsigned int</code>	4 bytes
<code>long, unsigned long</code>	4 bytes
<code>bool</code>	1 byte

Типове с плаваща запетая в C++ – знаков модел

- АМ на реалния тип не се отличава от математическото понятие за реална алгебрична система. Този тип не е дискретен, т.е. както е известно от математиката множеството от стойностите му не е изброимо, защото за дадена реална стойност не можем да кажем коя е непосредствено по-малка или по-голяма стойност (напр. дали след 0,1 следва 0,2 или 0,12, или 0,102, или 0,1000005?)
- Деклариране и обхват:

Т и п	Ф и з и ч е с к о п р е д с т а в я н е
<code>float</code>	Реално число (ниска точност)
<code>double</code>	Реално число (стандартна точност)
<code>long double</code>	Реално число (стандартна точност)

- Примери:
`float money;`
`double pi;`

Типове с плаваща запетая в C++ – множество от стойности и физически модел

- Множество от стойности – числа с плаваща запетая с единична, двойна или „разширена“ точност

- Физическо представяне:

Т и п	Р а з м е р
<code>float</code>	4 bytes
<code>double</code>	8 bytes
<code>long double</code>	8 bytes

integral	16	signed	$\pm 3.27 \cdot 10^4$	-32767 to 32767
		unsigned	0 to $6.55 \cdot 10^4$	0 to 65535
	32	signed	$\pm 2.14 \cdot 10^9$	-2,147,483,647 to 2,147,483,647
		unsigned	0 to $4.29 \cdot 10^9$	0 to 4,294,967,295
	64	signed	$\pm 9.22 \cdot 10^{18}$	-9,223,372,036,854,775,807 to 9,223,372,036,854,775,807
		unsigned	0 to $1.84 \cdot 10^{19}$	0 to 18,446,744,073,709,551,615
floating point	32	IEEE-754	$\pm 3.4 \cdot 10^{\pm 38}$ (~7 digits)	$\pm 3.402,823,4 \cdot 10^{38}$
	64	IEEE-754	$\pm 1.7 \cdot 10^{\pm 308}$ (~15 digits)	$\pm 1.797,693,134,862,315,7 \cdot 10^{308}$

Прости стандартни типове в C++ – множество от операции и релации

Като следствие от свойствата на скаларните типове, в езика C++ за всички прости стандартни типове могат да се прилагат следните релации и операции:

- ♦ всички сравнения `==`, `!=`, `<`, `>`, `<=`, `>=`
- ♦ **пре/пост-операция за нарастване** (*pre/post increment operators*) – добавя 1

към аргумента си

```
int count=5;
```

```
++count; /* PRE Increment, добавя 1 към count */
```

```
count++; /* POST Increment, добавя 1 към count */
```

- ♦ **пре/пост-операция за намаляване** (*pre/post decrement operators*) – изважда 1 от аргумента си (🔊 освен за тип `bool`)

```
int count=5;
```

```
--count; /* PRE Increment, изважда 1 от count */
```

```
count--; /* POST Increment, изважда 1 от count */
```

- ♦ **логически операции:**

`&&` (логическо "и"), `||` (логическо "или"), `!` (логическо "не")

- ♦ **извеждане** (`cout`) и **въвеждане** (`cin`), като за стойности от тип `bool` се използва техният целочислен еквивалент

Прости стандартни типове в C++ – аритметични операции

О п е р а ц и я	З н а к	П р и м е р	Стойност на sum / money преди	Стойност на sum / money след
У м н о ж е н и е	*	sum = sum * 2;	4	8
Д е л е н и е	/	sum = sum / 2;	4	2
Д е л е н и е	/	money = money / 2;	4.2	2.10000
С ъ б и р а н е	+	sum = sum + 2;	4	6
И з в а ж д а н е	-	sum = sum - 2;	4	2
Н а р а с т в а н е prefix	++	++sum;	4	5
Н а м а л я в а н е prefix	--	--sum;	4	3
Н а р а с т в а н е postfix	++	sum++;	4	5
Н а м а л я в а н е postfix	--	sum--;	4	3
М о д у л	%	sum = sum % 3;	4	1



Операцията % е определена само за целочислени аргументи

Пример:

```
int i=1, i1=1, j, k;
j=++i; // j е 2, i е 2
k=i1++; // k е 1, i1 е 2
```


Прости стандартни типове в C++ – стандартни функции в `math.h`

Стандартни функции за извършване на някои **математически изчисления** с простите типове са декларирани в заглавния файл `math.h`:

Routine	Use
<code>abs</code>	Return absolute value of <code>int</code>
<code>acos</code>	Calculate arccosine
<code>asin</code>	Calculate arcsine
<code>atan, atan2</code>	Calculate arctangent
<code>ceil</code>	Find integer ceiling
<code>cos</code>	Calculate cosine
<code>cosh</code>	Calculate hyperbolic cosine
<code>div</code>	Divide one integer by another, returning quotient and remainder
<code>exp</code>	Calculate exponential function
<code>fabs</code>	Find absolute value
<code>floor</code>	Find largest integer less than or equal to argument
<code>fmod</code>	Find floating-point remainder
<code>labs</code>	Return absolute value of <code>long</code>

Routine	Use
<code>ldiv</code>	Divide one <code>long</code> integer by another, returning quotient and remainder
<code>log</code>	Calculate natural logarithm
<code>log10</code>	Calculate base-10 logarithm
<code>modf</code>	Split argument into integer and fractional parts
<code>pow</code>	Calculate value raised to a power
<code>rand</code>	Get pseudorandom number
<code>sin</code>	Calculate sine
<code>sinh</code>	Calculate hyperbolic sine
<code>sqrt</code>	Find square root
<code>srand</code>	Initialize pseudorandom series
<code>tan</code>	Calculate tangent
<code>tanh</code>	Calculate hyperbolic tangent

Прости стандартни типове в C++ – стандартни функции в `ctype.h`

Стандартни функции за обработка на **символния прост тип** са декларирани в заглавния файл `ctype.h`

(1) **is**-функции:

Routine	Character Test Condition
<code>isalnum</code>	Alphanumeric
<code>isalpha</code>	Alphabetic
<code>__isascii</code>	ASCII
<code>isctrl</code>	Control
<code>isdigit</code>	Decimal digit
<code>islower</code>	Lowercase
<code>isprint</code>	Printable
<code>ispunct</code>	Punctuation
<code>isspace</code>	White-space
<code>isupper</code>	Uppercase

(2) **to**-функции:

Routine	Description
<code>tolower</code>	Converts <i>c</i> to lowercase if appropriate
<code>toupper</code>	Converts <i>c</i> to uppercase if appropriate

Дискретни типове дефинирани от програмиста в C++ – знаков модел на изброен тип

Изброен тип – дискретен тип, чието множеството от стойности е линейно подредено крайно множество, което се задава чрез пряко изброяване на имената на тези стойности

- **Синтаксис** за деклариране на типа:

```
enum <име>_незад { <изброяващ списък>_незад }
```

Където <изброяващ списък> е:

<име> = <константен израз>_незад , ...

- **Физически модел:** 4 байта
- **Множеството от стойности:** съвкупността от идентификаторите, изброени в скобите при деклариране на типа. На всяка стойност от даден изброен тип се съпоставя целочислена стойност:
 - по подразбиране: 0, 1, 2, ... по реда на изброяването им
 - чрез явно задаване в *изброяващия списък*, като стойностите могат да се повтарят. Ако цялото число, съответно на някоя изброена стойност не е зададена, то е с 1 по-голямо от целочислената стойност на предишното име от списъка

- **Примери:**

```
enum Suit { Diamonds, Hearts, Clubs, Spades } s;  
enum Suit {Diamonds = 5,Hearts,Clubs = 4,Spades=-1};  
enum Color {Red=5, Orange=Red+1, Yellow, Green, Blue};
```

Изброен тип в C++ – операции и релации

- Стойностите от изброен тип се считат еквивалентни на съответните им целочислени стойности. Поради това за тях са валидни всички **операции** валидни и за **целочислените типове**
- Стойностите от изброен тип са подредени според наредбата на съответните им целочислени стойности. Поради това за тях валидни всички **операции за сравнение**
- При **извеждане** (напр. с `cout`) на стойности от изброен тип се извежда техният целочислен еквивалент
- Стойностите от изброен тип не могат пряко да се въвеждат (напр. с `cin`)
- **Примери:**

```
s = sin(s);  
cout<<(Orange<Yellow+1)<<endl;
```

Изрази. Изрази в C++

- Под **израз** (*expression*) в ЕП се разбира правило за изчисляване на някаква стойност от точно определен тип. Изразът се състои от един или повече операнда, свързани помежду си с някои от допустимите операции в езика
- При срещане на израз в една програма се извършва изчисление на израза и се формира една единствена стойност
- При **конструиране на изрази** е необходимо да се съчетават правилно операциите и операндите
- Изразът в езика C++ представлява:

Последователност от операнди и операции, която се използва за една от следните цели:

- изчисляване на стойност въз основа на операндите (константи, променливи, изрази, израз в кръгли скоби): `-2.16`, `money`, `2*a-3`, `2/(a+3)`, `i++`
- обозначаване на функции или други обекти: `sqrt(x)`, `myFunc('s', 2)`
- генериране на т.нар. “страничен ефект”, т.е. всяко действие различно от изчисляване на стойността на израза (напр. промяна на стойността на променлива, чрез присвояване): `money = 100`, `money += 100`, `i++`



Изразите, които се използват за последните две цели ще бъдат разгледани по-подробно по-късно

Изчисляване на изрази. Приоритет на операциите в C++

- При изчисляване на изразите се прилагат следните правила:
 - в един израз операциите с по-висок приоритет се изчисляват преди тези с по-нисък
 - в случай на няколко последователни операции с един и същ приоритет, изчисленията се извършват отляво надясно
 - изразите със скоби се изчисляват първи
- Операциите изброени по намаляване на техния приоритет са:

О пер а тор	С м и с ъ л
()	Function call
()	Conversion
++	Postfix increment
--	Postfix decrement
++	Prefix increment
--	Prefix decrement
*	Dereference
+	Unary plus
-	Arithmetic negation (unary)
!	Logical NOT
*	Multiplication
/	Division
%	Remainder (modulus)
+	Addition
-	Subtraction

О пер а тор	С м и с ъ л
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equality
!=	Inequality
& &	Logical AND
	Logical OR
e1 ? e2 : e3	Conditional
=	Assignment
*=	Multiplication assignment
/=	Division assignment
% =	Modulus assignment
+=	Addition assignment
- =	Subtraction assignment

Съвместимост и проверка на типовете в ЕП

Съвместимостта (еквивалентност, *type compatibility*) на типове това е свойството им да могат да се заместват като аргументи на изрази, функции, оператори и други

- **Определение:** Ще казваме, че типът **T1** е съвместим с типа **T2** ако, в случай когато се изисква аргумент от типа T1, на негово място може да се зададе аргумент от типа T2

Проверката на типа (*type checking*)

- **Определение:** Процесът, при който транслаторът проверява дали всички конструкции в програмата имат смисъл по отношение на типовете на константите, променливите, процедурите и другите единици
- **Видове:** варира от строга до либерална в различните ЕП
- **Начини на извършване:** *статична* (по време на компилирането, *static*) и *динамична* (по време на изпълнение, *dynamic*)

Преобразуване на типовете в ЕП

Преобразуването на типовете (*type conversion*) позволява изрази от действителния тип T_a да се използват в контекста на очакван тип T_e

- **Начини на преобразуване:** *явно* (програмиста пише собствен код за преобразуване на типове); *неявно* (ЕП предоставя специална поддръжка за преобразуване типове)
- **Начини на задаване на преобразуване в ЕП:**
 - *безусловно преобразуване* (неявно, принудително, коерция, *coercion*) – преобразуването автоматично се извършват от процесора (ако такова е дефинирано)
 - *специални функции* (напр. `trunc` и `round` са две различни преобразувания от `real` в `integer` в езика Pascal)
 - *претипизиране* (явно, определяне на типа, *type casting*) – чрез указване на типа, към който става преобразуването
- **Видове преобразувания от семантична гледна:**
 - *Преобразуванията без загуби* запазват информацията, съдържаща се в стойността (напр. целочислен в низ; плаваща запетая в двойна точност; знаков в целочислен)
 - *Преобразуванията със загуби* могат да загубят информация (напр. плаваща запетая в целочислен; целочислен в знаков)

Преобразуване на типовете в C++

Правилата на C++ са доста либерални при задаване на смесени изрази. Реализирани са безусловното преобразуване, специални функции за преобразуване и претипизиране

Преобразуване на типовете се извършва в следните случаи:

- Когато стойност от един тип се присвоява на променлива от друг тип или когато някоя операция преобразува типа на своите операнди преди да бъде извършена
- Когато стойност от един тип явно се преобразува към друг тип
- Когато стойност се предава като аргумент на функция или когато се връща като резултат на функция

Преобразуване на типовете в C++:

Безусловно преобразуване

- **Правила:** В един израз или подизраз всички компоненти се преобразуват безусловно до типа с най-широк диапазон, който присъства в израза по специални правила и зависи от компилатора (вж. примера)

Long double (най-широк)

Double

Float

Unsigned long int

Long int

Unsigned int

Int (най-малък)

- **Пример:**

```
float    fVal;        double  dVal;  int    iVal;
unsigned long ulVal;
dVal = iVal * ulVal; /* iVal се преобразува в unsigned long
                    * Резултатът от умножението се преобразува
                    * до double */
dVal = ulVal + fVal; /* ulVal се преобразува в float
                    * Резултатът от събирането се преобразува
                    * до double */
```



По-безопасно е да се прилага стриктно преобразуване, като се използва някой от другите два

Преобразуване на типовете в C++: Претипизиране

- **Синтаксис:** (<име на тип>) <израз>
- **Семантика:** Резултатът на <израз> се преобразува към типа <име на тип>. Преобразува от кой да е скаларен тип към кой да е друг скаларен тип по правилата, които се използват и при безусловното преобразуване
- **Пример:**

```
float x = 3.1;  
int i;  
i = (int)x; // стойността на i е 3  
x =(float)2;
```
- **Случаи, в които е допустимо претипизирането:**

Тип на резултата	Тип, който се преобразува
Целочислени	Всеки целочислен, с плаваща запетая или тип указател
С плаваща запетая	Всеки аритметичен тип
Тип указател или <code>void *</code>	Всеки целочислен, <code>void *</code> или тип указател
Структура, обединение или масив	—
Тип <code>void</code>	Всеки тип

Преобразуване на типовете в C++:

Специални функции за преобразуване

- **Синтаксис:** <име на тип> (<израз>)
- **Семантика:** Резултатът на <израз> се преобразува към типа <име на тип>. Преобразува от кой да е скаларен тип към кой да е друг скаларен тип по правилата, които се използват и при безусловното преобразуване
- **Примери:**

```
float f=float(2);
```

```
enum Days
{
    Sunday, Monday, Tuesday, Wednesday,
    Thursday, Friday, Saturday
};
```

```
int i;
Days d = Thursday;
```

```
i = d;      // неявно преобразуване
```

```
cout << "i = " << i << "\n";
```

```
d = 6;      // Грешка !!!
```

```
d = (Days)6; // Претипизиране
```

```
d = Days( 4 ); // Функция за преобразуване на типа
```

🔊 Някои от стандартните функции за обработка на стандартните прости типове, които са декларирани в заглавния файл `math.h` също служат за преобразуване на типовете (вж. слайд 24)

Често допускани грешки при използване на простите типове в C++

- В компютъра числата се представят в краен брой битове, което може да доведе до поне два проблема:

- целочислените стойности са твърде големи, за да бъдат представени
- реалните стойности са представени приблизително (препоръчително е типът `float` да не се използва)



Във всички тези случаи, ако е възможно компилаторът извежда предупредителни съобщения (*warning*)

- **Пример:**

```
//пример за грешка от закръгляне
double price=3e14;
double price_=price-0.05;
double disc=price-price_;
//трябва де е 0.05
cout<<disc<<"\n";//извежда 0.0625 !!!
```

- Операцията “целочислено деление” може да предизвика проблеми, защото в случай, че и двата аргумента са цели, то в резултата дробната част се игнорира ($10/4$ е 2)