



Универзитет „Св. Кирил и Методиј“ во Скопје  
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

# Контрола на тек 2

Структурно програмирање

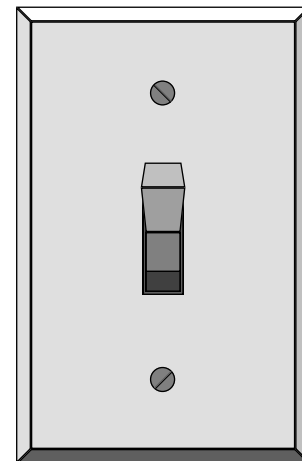
ФИНКИ 2024

# Избор од повеќе можности

## switch - case

```
switch (izraz)  
{  
    case konstanta1: blok_naredbi1;  
    case konstanta2: blok_naredbi2; break;  
    . . .  
    case konstantan: blok_naredbin;  
    default:        naredbi;  
}
```

*izraz* мора да резултира во **int** или **char** податочен тип



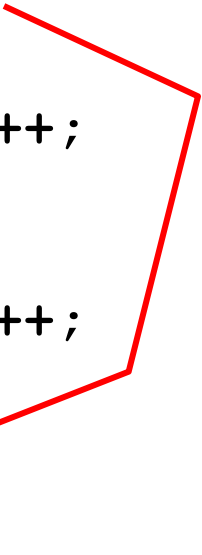
# Избор од повеќе можности

## switch – case


- Не смее да има два или повеќе case изрази со иста вредност.
- Програмата продолжува со наредбите зад case наредбата со вредноста на пресметаниот израз од switch наредбата.
- Се извршуваат следните наредби сè додека не се најде на наредбата break или до крајот на switch-case блокот.
- Ако нема case наредба со соодветна вредност се извршуваат наредбите од default блокот.
- Ако не е наведен default блок на наредби, не се случува ништо - се продолжува со наредбите зад switch-case блокот.

# switch – case без break

```
switch (option){
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```



```
switch (option){
    case 'A':
        aCount++;
    case 'B':
        bCount++;
    case 'C':
        cCount++;
}
```



# default case

```
switch (option) {
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
    default:
        otherCount++;
        break;
}
```

- Ако имате 75 поени, која оценка ќе ја добиете? Што ќе испечати програмата?

```
switch (int(score) / 10) {  
    case 10:  
    case 9: cout << "Grade = A" << endl;  
    case 8: cout << "Grade = B" << endl;  
    case 7: cout << "Grade = C" << endl;  
    case 6: cout << "Grade = D" << endl;  
    default: cout << "Grade = F" << endl;  
}
```

- Ако имате 75 поени, која оценка ќе ја добиете? Што ќе испечати програмата?

```
switch (int(score) / 10) {  
    case 10:  
    case 9: cout << "Grade = A" << endl; break;  
    case 8: cout << "Grade = B" << endl; break;  
    case 7: cout << "Grade = C" << endl; break;  
    case 6: cout << "Grade = D" << endl; break;  
    default: cout << "Grade = F" << endl;  
}
```

# Решете

- Напишете програма која на влезот очекува број 1, 2 или 3.

Ако се внесе

- ☐ 1 - печати 1
- ☐ 2 - печати 2 3
- ☐ 3 - печати 3
- ☐ било кој друг број – печати “Greska”



# Решение

```
#include <iostream>
using namespace std;

int main() {
    int num;

    cout << "Внеси број помеѓу 1 и 3: ";
    cin >> num;
    switch (num) {
        case 1:
            cout << "1 "; break;
        case 2:
            cout << "2 ";
        case 3:
            cout << "3"; break;
        default:
            cout << "Greska\n";
    }

    return 0;
}
```

# Што ќе отпечати следнава програма?

```
#include <iostream>
using namespace std;
int main() {
    int j = 0;
    while (j < 6)
    {
        switch (j)
        {
            case 0: j++;
            case 1: j++; break;
            case 2:
            case 3: j += 2; break;
            default: j = j - 1;
        }
        cout << "Vrednosta na j e " << j++ << endl;
    }
    return(0);
}
```

Vrednosta na j e 2  
Vrednosta na j e 5

како ќе работи  
програмата без ова ++?

# Напишете програма...

... која пресметува вредност на едноставен аритметички израз (без приоритети, само цели броеви):

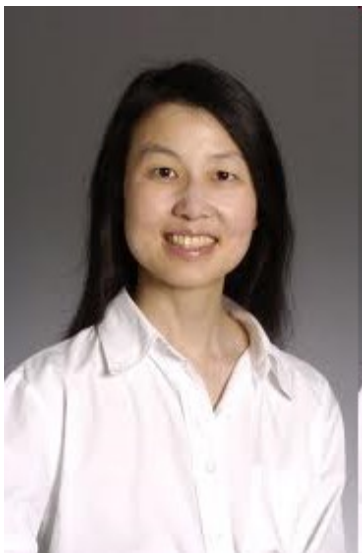
```
#include <iostream>
using namespace std;
int main() {
    char oper = '+';
    int broj, resenieto = 0;
    do {
        cin >> broj;
        switch (oper) {
            case '+': resenieto += broj; break;
            case '-': resenieto -= broj; break;
            case '*': resenieto *= broj; break;
            case '/': resenieto /= broj; break;
        }
        cin >> oper;
    } while (oper != '=');
    cout << "Resenieto e " << resenieto << endl;
    return 0;
}
```

15+2\*2-4/3+1=  
Resenieto e 11



# Алгоритми

# Алгоритамско размислување

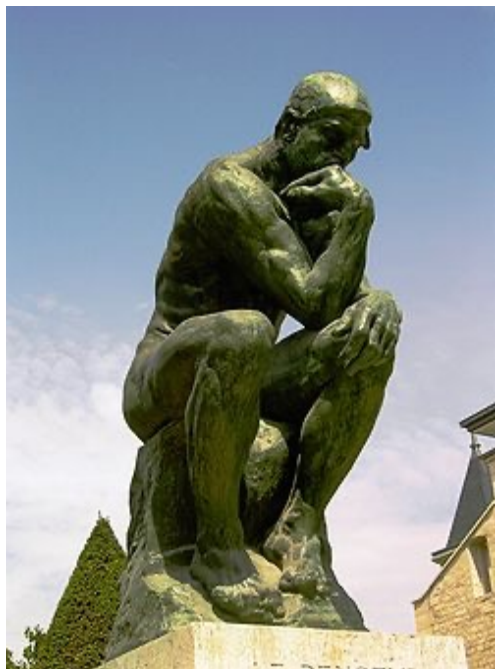


Алгоритамското (пресметковното) размислување (англ. computational thinking) е фундаментална вештина за секого, не само за информатичарите. Кон **читањето**, **пишувањето** и **аритметиката** треба да се додаде и алгоритамското размислување како неопходна вештина за секое дете.

...

Jeannette M. Wing  
Computational Thinking  
CACM, vol. 49, no. 3, pp. 33-35, 2006

# Што е „Computational Thinking“?



**Мисловните процеси вклучени во формулирање на проблем и изразување на негово решение (решенија) на тој начин што човек или машина ќе може ефективно да ги спроведе.**

Има 2 чекора кај пресметковното размислување: наоѓање на соодветни апстракции за доменот на проблемот и потоа формулирање на алгоритми за решавање проблеми со примена на најдените апстракции

# Алатки за почетничко програмирање

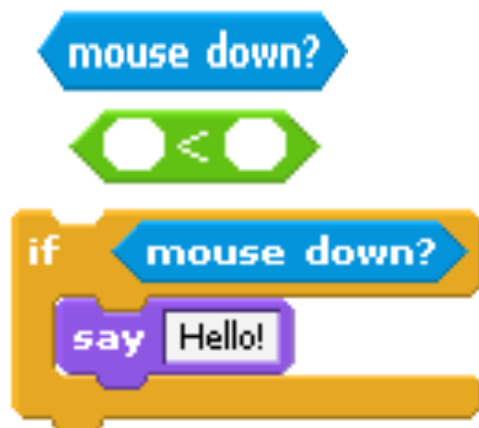
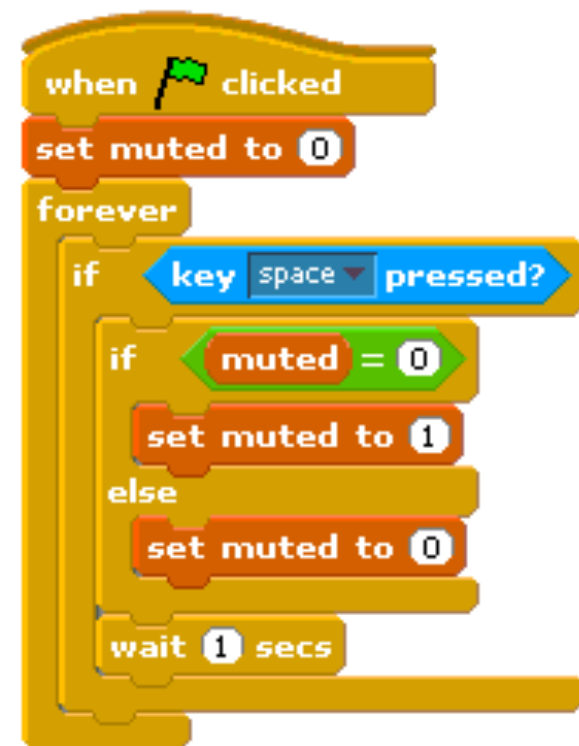
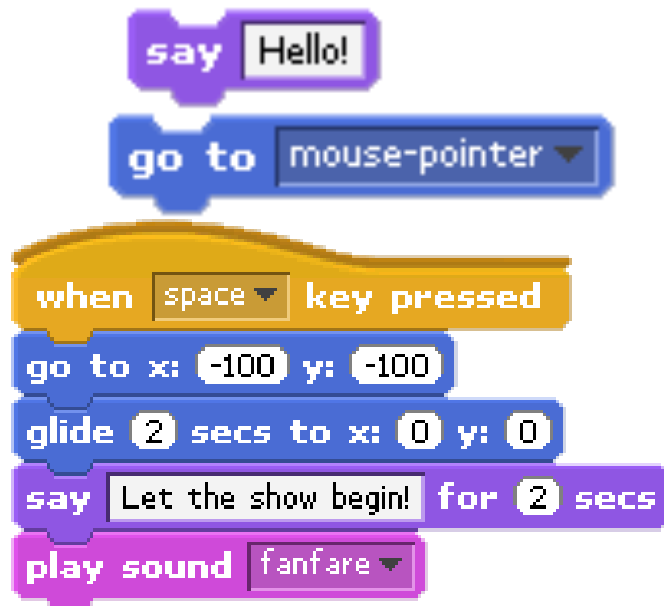
- Постојат бројни алатки продуцирани во последните неколку години, дел од кои целат дури и кон ученици на 7 годишна возраст

- Примери:

- [Swift Playgrounds](#) е бесплатна iPad апликација која треба на децата да им помогне со основите на програмирањето. Апликацијата преку команди има за цел да ги научи децата на неопходната логика.
- Scratch (сега Scratch 3.0) е програмска околина развиена на MIT. Веќе десетина години се употребува како алатка за почетничко програмирање. Линк: <https://scratch.mit.edu>
- ScratchJr – базирана на Scratch, но со идеја да е соодветна дури и за деца кои се уште не знаат да читаат/пишуваат
- Редица ‘видео игри’ кои поттикнуваат алгоритамско размислување. Пример: CodeMonkey ([www.playcodemonkey.com](http://www.playcodemonkey.com)), Beta ([www.betathegame.com](http://www.betathegame.com)), codecombat.com, Hack ‘n’ Slash ([hacknslashthegame.com](http://hacknslashthegame.com)), DigitMile ([digit.mile.mk](http://digit.mile.mk) - развиена во Македонија)...



# Примери од Scratch





# Алгоритми - дефиниции

- Chamber's on-line dictionary “A set of prescribed computational procedures for solving a problem; a step-by-step method for solving a problem.”
- Knuth, “The Art of Computer Programming” ... “ An algorithm is a finite, definite, effective procedure, with some input and some output.”
  - **Finiteness:** *"An algorithm must always terminate after a **finite number of steps**"*
  - **Definiteness:** *"Each step of an algorithm must be **precisely defined**; the actions to be carried out must be rigorously and unambiguously specified for each case"*
  - **Input:** *"...quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects"*
  - **Output:** *"...quantities which have a specified relation to the inputs"*
  - **Effectiveness:** *"... all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using paper and pencil"*

# Алгоритми

## ■ Алгоритам –

- ☐ постапка која се состои од конечно множество на точно дефинирани дејства (операции),
- ☐ операции применети врз влезните податоци, по строго пропишан редослед, кои доведуваат до излезни резултати

## ■ алгоритамски чекори - дејства од кои се состои еден алгоритам

## ■ Зависно од општоста на чекорите алгоритамот може да биде:

- општ
- детален

# Пример за алгоритам

Да се подредат три броја  $a$ ,  $b$  и  $c$  по големина.

## Општ алгоритам:

чекор-1: Задавање на три броеви.

чекор-2: Подредување на првиот и вториот број по големина.

чекор-3: Подредување на првиот и третиот број по големина.

чекор-4: Подредување на вториот и третиот број по големина.

чекор-5: Печатење на броевите.

## Подетален алгоритам:

чекор-1: Задавање на броевите  $a$ ,  $b$  и  $c$ .

чекор-2: ако  $a > b$  тогаш  $a \leftrightarrow b$

чекор-3: ако  $a > c$  тогаш  $a \leftrightarrow c$

чекор-4: ако  $b > c$  тогаш  $b \leftrightarrow c$ .

чекор-5: Печатење на броевите  $a$ ,  $b$  и  $c$ .

# Детален алгоритам

чекор-1: Задавање на броевите  $a$ ,  $b$  и  $c$ .

чекор-2: ако  $a > b$  тогаш

чекор-2-1:  $rot \leftarrow a$

чекор-2-2:  $a \leftarrow b$

чекор-2-3:  $b \leftarrow rot$

чекор-3: ако  $a > c$  тогаш

чекор-3-1:  $rot \leftarrow a$

чекор-3-2:  $a \leftarrow c$

чекор-3-3:  $c \leftarrow rot$

чекор-4: ако  $b > c$  тогаш

чекор-4-1:  $rot \leftarrow b$

чекор-4-2:  $b \leftarrow c$

чекор-4-3:  $c \leftarrow rot$

чекор-5: Печатење на броевите  $a$ ,  $b$  и  $c$ .

# Што е правилен алгоритам?

**Правилен алгоритам е оној кој ги исполнува условите:**

**Во блок-дијаграмите**

- има само една влезна линија
- има само една излезна линија
- за секој јазол постои пат од влезната до излезната линија кој минува низ него.

**Значи: правилен алгоритам не смее да има недостапни сегменти.**

# Пример

Дали овој алгоритам  
е ПРАВИЛЕН?

```
алгоритам СоГрешка;  
почеток  
    читај m, n;  
    ако  $m \geq n$   
        тогаш  
            печати m, '≥', n;  
    инаку  
        ако  $m < n$   
            тогаш  
                печати m, '<', n;  
            инаку  
                печати m, '=', n;  
    крај_ако { $m < n$ }  
    крај_ако { $m \geq n$ }  
    печати 'Каде е грешката ?';  
крај
```

# Пример

Дали овој алгоритам  
е ПРАВИЛЕН?

Има недостапен сегмент

```
алгоритам СоГрешка;  
почеток  
    читај m, n;  
    ако  $m \geq n$   
        тогаш  
            печати m, '≥', n;  
    инаку  
        ако  $m < n$   
            тогаш  
                печати m, '<', n;  
            инаку  
                печати m, '=', n;  
    крај_ако { $m < n$ }  
    крај_ако { $m \geq n$ }  
    печати 'Каде е грешката ?';  
крај
```

# Пример

Дали овој алгоритам  
е ПРАВИЛЕН?

Има недостапен сегмент

```
алгоритам СоГрешка;  
почеток  
    читај m, n;  
    ако  $m \geq n$   
        тогаш  
            печати m, '≥', n;  
    инаку  
        ако  $m < n$   
            тогаш  
                печати m, '<', n;  
            инаку  
                печати m, '=', n;  
        крај_ако { $m < n$ }  
    крај_ако { $m \geq n$ }  
    печати 'Каде е грешката ?';  
крај
```



# Пример

Дали овој алгоритам  
е ПРАВИЛЕН?

Има недостапен сегмент

Што треба да се исправи за да биде  
правилен?

```

алгоритам СоГрешка;
почеток
    читај m, n;
    ако m ≥ n
        тогаш
            печати m, '≥', n;
    инаку
        ако m < n
            тогаш
                печати m, '<', n;
            инаку
                печати m, '=', n;
        крај_ако {m < n}
    крај_ако {m ≥ n}
    печати 'Каде е грешката?';
крај
    
```

# Пример

Дали овој алгоритам  
е ПРАВИЛЕН?

Има недостапен сегмент

Што треба да се исправи за да биде  
правилен?

ако  $m > n$

алгоритам СоГрешка;  
почеток

читај  $m, n$ ;

ако  $m \geq n$

тогаш

печати  $m, ' \geq ', n$ ;

инаку

ако  $m < n$

тогаш

печати  $m, ' < ', n$ ;

инаку

печати  $m, ' = ', n$ ;

крај\_ако  $\{m < n\}$

крај\_ако  $\{m \geq n\}$

печати 'Каде е грешката ?';

крај

# Правилно градење алгоритми

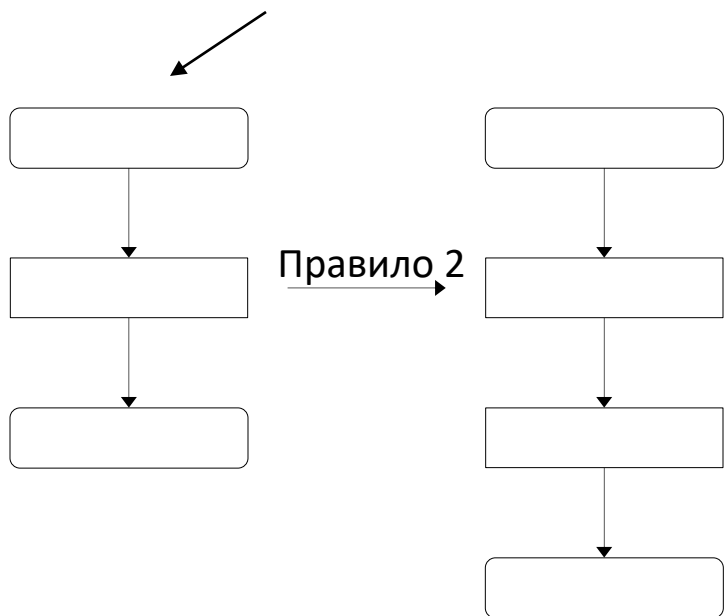
## ■ Правила

1. Започнете со наједноставниот блок-дијаграм
2. Секој правоаголник (процес) може да се замени со два последователни правоаголника (процеси)
3. Секој правоаголник (процес) може да се замени со која било контролна структура (секвенца, if, if/else, while, do/while, for)
4. Правилата 2 и 3 може да се применат во кој било редослед и поголем број пати.

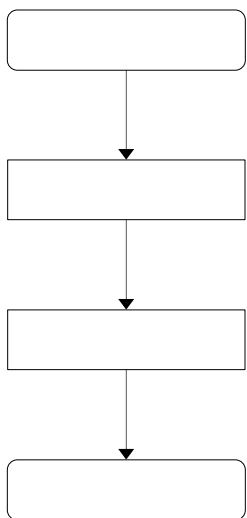


## Правило 1

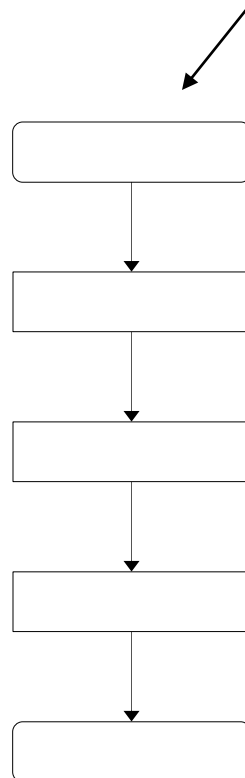
(започни со  
наједноставниот  
Блок-дијаграм)



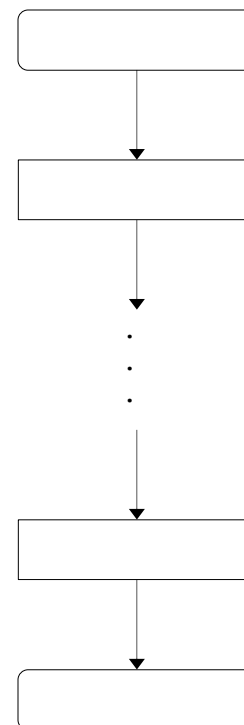
Правило 2



Правило 2



Правило 2



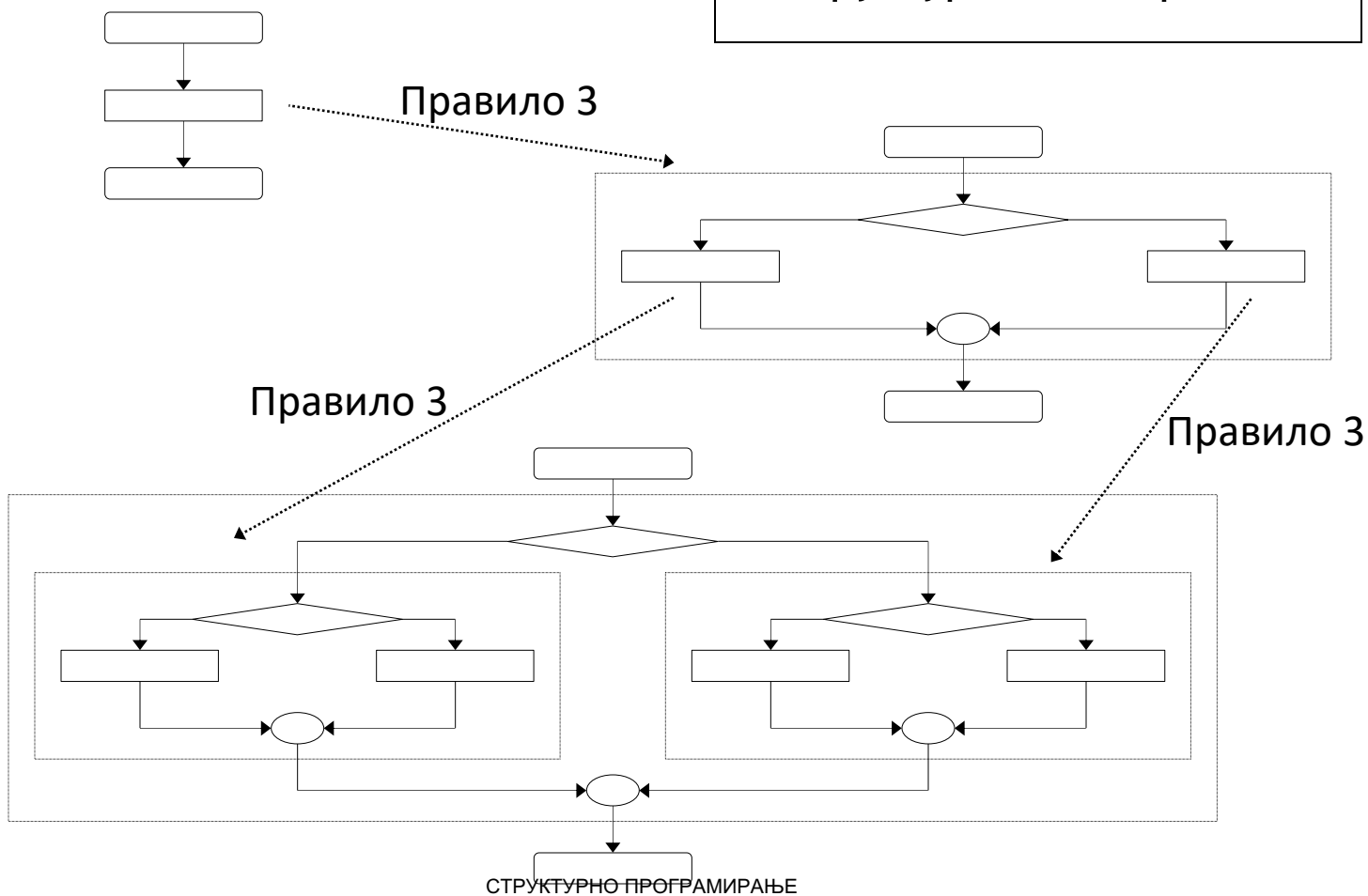
## Правило 2

Секој правоаголник може  
да се замени со два  
последователни  
правоаголници



## Правило 3

Секој правоаголник може да  
се замени со контролна  
структура за избор



# Правилно градење алгоритми

- Кога пишуваме алгоритам, треба да го имаме во предвид следното:
  - Алгоритамот треба да биде јасен и недвосмислен
  - Треба да има добро дефинирани влезни вредности
  - Алгоритамот мора да произведе една или повеќе добро дефинирани излезни вредности
  - Алгоритамот мора да запре или заврши по конечен број на чекори
  - Во еден алгоритам инструкциите се даваат чекор по чекор и тие се независни од било кој компјутерски код

# Анализа на алгоритми

- Што може да се анализира?
- Може:
  - да се одреди времето на извршување на алгоритмот како функција од неговите влезни податоци
  - да се одреди максималното побарување на меморија што е потребна за податоците
  - да се одреди точната големина на програмскиот код
  - да се одреди дали програмата точно го пресметува посакуваниот резултат
  - да се одреди комплексноста на алгоритмот
  - да се види колку добро алгоритмот се соочува со неочекуваните и погрешни влезни податоци

# Анализа на сложеност

- Во предметот Алгоритми и податочни структури, семестар 3



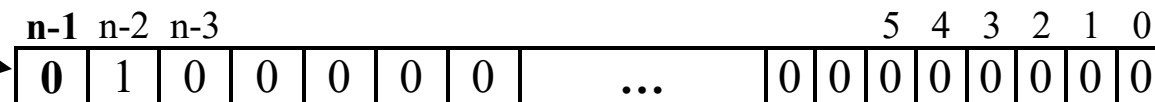


# Претставување на броевите во компјутер

# Претставување на целите броеви

- Цели броеви
  - без предзнак – неозначени (unsigned) (0 до  $max-1$ )
  - со предзнак – означени (signed) ( $-max$  до  $max-1$ )
- Кај броеви **без предзнак** битот со најголема важност е **дел од** бројот.
- Кај броеви **со предзнак** битот со најголема важност се користи за претстава на **знакот на** бројот.
  - Тој е:     **0** – за позитивните броеви и нулата  
              **1** – за негативните броеви

бит со најголема  
важност



Битот со најголема важност **НЕ Е**  
директна претстава на предзнакот!  
Не важи:  $+ \rightarrow 0$  ;  $- \rightarrow 1$

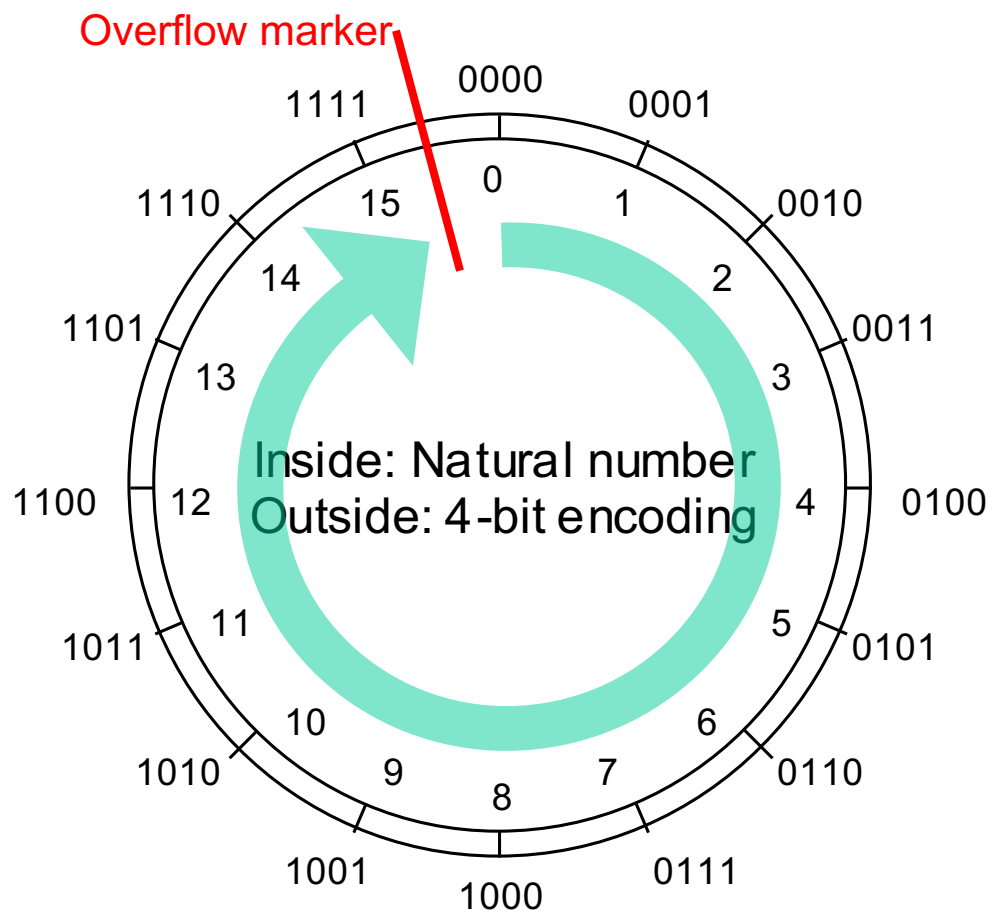


# НЕОЗНАЧЕНИ

■  $n=4$ , 0 до 15

0000 +1=	1000 +1=
0001 +1=	1001 +1=
0010 +1=	1010 +1=
0011 +1=	1011 +1=
0100 +1=	1100 +1=
0101 +1=	1101 +1=
0110 +1=	1110 +1=
0111 +1=	1111 +1=

**0000**



# Опсег на ОЗНАЧЕНИ броеви

Опсегот на означените броеви претставени со  
n-битен збор е

$$-2^{n-1} \leq x \leq 2^{n-1}-1$$

има вкупно  $2^{n-1} + 1$  (со нулата) +  $2^{n-1} - 1 = 2^n$  броеви.

	од	до
n=5	$-2^{n-1} = -2^4 = -16$	$2^{n-1}-1 = 2^4-1 = 15$
n=8	$-2^{n-1} = -2^7 = -128$	$2^{n-1}-1 = 2^7-1 = 127$
n=16	$-2^{n-1} = -2^{15} = -32768$	$2^{n-1}-1 = 2^{15}-1 = 32767$
n=32	$-2^{n-1} = -2^{31} = -2147483648$	$2^{n-1}-1 = 2^{31}-1 = 2147483647$

# Како се добиваат броевите од опсегот?

- Со  $n$  битови, броевите се наоѓаат во опсегот  $[-2^{n-1}, 2^{n-1} - 1]$
- Негативниот дел од опсегот се добива со
  - ИНВЕРТИРАЊЕ
  - Додавање 1

... **ДВОЕН КОМПЛЕМЕНТ**

Пр.  $n=3, [-4, 3]$

■	0 0 0	0
■	0 0 1	1
■	0 1 0	2
■	0 1 1	3
■	<b>1</b> 0 0	-4 (11 +1 = 100 (4))
■	<b>1</b> 0 1	-3 (10 +1 = 11 (3))
■	<b>1</b> 1 0	-2 (01+1=10 (2))
■	<b>1</b> 1 1	-1 (00+1=01 (1))

Кај овие **предзначени** броеви  
битот со најголема важност е:  
**0** за позитивен број  
**1** за негативен број.

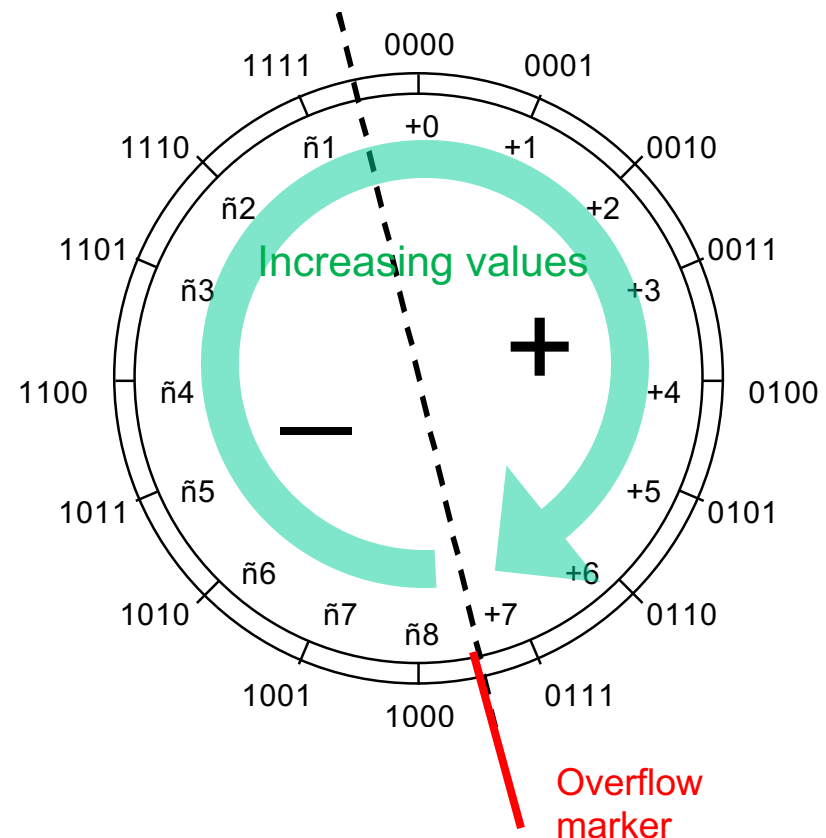
# Преполнување (overflow)

За  $n=4$ ,  $-8$  до  $7$  ( $-2^{n-1} \leq x \leq 2^{n-1}-1$ )

При аритметички операции со  
броеви ограничени во некој опсег  
доаѓа до

**Преполнување, претекување,  
прелевање (overflow),**

резултатот е НАДВОР од опсегот!!



# Уште еднаш, за $n=5$

Означени цели 16-битни броеви во опсегот -16 до 15

dekaden br oj	bi nar en br oj	dekaden br oj	bi nar en br oj
	$\overbrace{00\dots00000}^{16\text{-bita}}$		$\overbrace{11\dots10000}^{16\text{-bita}}$
0	00...00000	-16	11...10000
1	00...00001	-15	11...10001
2	00...00010	-14	11...10010
3	00...00011	-13	11...10011
4	00...00100	-12	11...10100
5	00...00101	-11	11...10101
6	00...00110	-10	11...10110
7	00...00111	-9	11...10111
8	00...01000	-8	11...11000
9	00...01001	-7	11...11001
10	00...01010	-6	11...11010
11	00...01011	-5	11...11011
12	00...01100	-4	11...11100
13	00...01101	-3	11...11101
14	00...01110	-2	11...11110
15	00...01111	-1	11...11111



# Floating Point Format

- Децималните броеви се претставуваат во формат на т.н. Подвижна запирка (floating point)  $1001 = +1.001 \times 2^3$
- Децималните броеви се претставени со 3 полиња:
  - ☐ Предзнак **s**,
  - ☐ Експонент **e**,
  - ☐ Мантиса **m**



# Податочни типови во C++

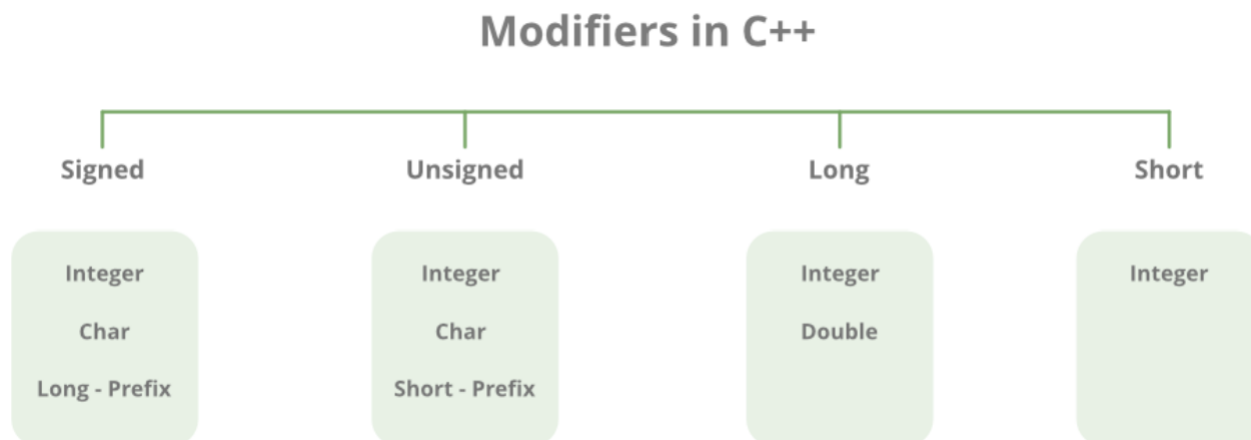
## ■ Основни податочни типови и нивна големина:

Податочен тип	Големина во бајти	опсег
int	4	-2,147,483,648 – 2,147,483,647
char	1	-128 – 127 или 0 – 255 (сите знаци од ASCII табелата 0-127)
float	4	$-3.4 \times 10^{38}$ – $3.4 \times 10^{38}$
double	8	$-1.7 \times 10^{308}$ – $1.7 \times 10^{308}$
wchar_t	2 или 4	1 проширен карактер

- `wchar_t` (wide character) проширен карактер е податочен тип кој е исто така карактер само што големината не е 8 бита, туку може да биде 16 или 32 бита.

# Податочни типови во C++

- Основните податочни типови во C++ може да се комбинираат со т.н. модификатори со цел да се промени големината на податокот кој може да биде зачуван
- Модификатори се:
  - ☐ signed
  - ☐ unsigned
  - ☐ short
  - ☐ long



# Податочни типови во C++

- Основните податочни типови со модификатори и нивна големина:

Податочен тип	Големина во бајти	Опсег
unsigned char	1	0 – 255 (сите знаци од ASCII табелата + проширената ASCII табела)
short int	2	-32,768 – 32,767
unsigned short int	2	0 – 65,535
unsigned int	4	0 – 4,294,967,295
long int	8	-9,223,372,036,854,775,808 – 9,223,372,036,854,775,807
unsigned long int	8	0 – 18,446,744,073,709,551,615
long double	12	$-1.1 \times 10^{4932}$ – $1.1 \times 10^{4932}$

\* Овие големина може да варираат зависно од компајлерот и машината на која се користат. Најправилен начин да се одреди опсегот е со користење на функцијата sizeof()

# Конверзија на типови во C++

- C++ дозволува да се прави конверзија од еден во друг податочен тип
- Постојат два типа конверзии:
  - Имплицитна конверзија
  - Експлицитна конверзија позната како податочно кастирање (type casting)

# Имплицитна конверзија

- Тип на конверзија која се извршува автоматски од страна на компајлерот

□ Позната и како автоматска конверзија

Пр. Конверзија од double во int

```
#include <iostream>
using namespace std;

int main()
{
    int num_int;
    double num_double = 9.99;

    num_int = num_double;

    cout << "num_int = " << num_int << endl;
    cout << "num_double = " << num_double << endl;

    return 0;
}
```

# Имплицитна конверзија

- Тип на конверзија која се извршува автоматски од страна на компајлерот

□ Позната и како автоматска конверзија

Пр. Конверзија од double во int

```
#include <iostream>
using namespace std;

int main()
{
    int num_int;
    double num_double = 9.99;

    num_int = num_double;

    cout << "num_int = " << num_int << endl;
    cout << "num_double = " << num_double << endl;

    return 0;
}
```

Излез:

```
num_int = 9
num_double = 9.99
```

# Имплицитна конверзија

- Пр. Конверзија од int во double

```
#include <iostream>
using namespace std;

int main() {

    int num_int = 9;
    double num_double;

    num_double = num_int;

    cout << "num_int = " << num_int << endl;
    cout << "num_double = " << num_double << endl;

    return 0;
}
```



# Имплицитна конверзија

## □ Пр. Конверзија од int во double

```
#include <iostream>
using namespace std;

int main() {

    int num_int = 9;
    double num_double;

    num_double = num_int;

    cout << "num_int = " << num_int << endl;
    cout << "num_double = " << num_double << endl;

    return 0;
}
```

Излез:

```
num_int = 9
num_double = 9
```

# Експлицитна конверзија

- Кога корисникот мануелно сака да промени еден податок во друг се нарекува експлицитна конверзија
- Три начини како може да се постигне:
  - ☐ C-style кастирање
  - ☐ Со функциска нотација
  - ☐ Со користење оператори за кастирање

# Експлицитна конверзија

## ■ Пример за C-style кастирање

## ■ Синтакса:

□ (data\_type)expression;

```
#include <iostream>
using namespace std;

int main() {

    int num_int = 116;
    char ch;

    ch = (char)num_int;

    cout << "num_int = " << num_int << endl;
    cout << "ch = " << ch << endl;

    return 0;
}
```

Излез:

```
num_int = 116
ch = t
```

# Експлицитна конверзија

- Пример за кастирање со функцииска нотација
- Синтакса:

□ `data_type(expression);`

```
#include <iostream>
using namespace std;

int main() {

    int num_int = 116;
    char ch;

    ch = char(num_int);

    cout << "num_int = " << num_int << endl;
    cout << "ch = " << ch << endl;

    return 0;
}
```

Излез:

```
num_int = 116
ch = t
```

# Експлицитна конверзија

## ■ Оператори за кастирање:

- ☐ `static_cast`

- ☐ `dynamic_cast`

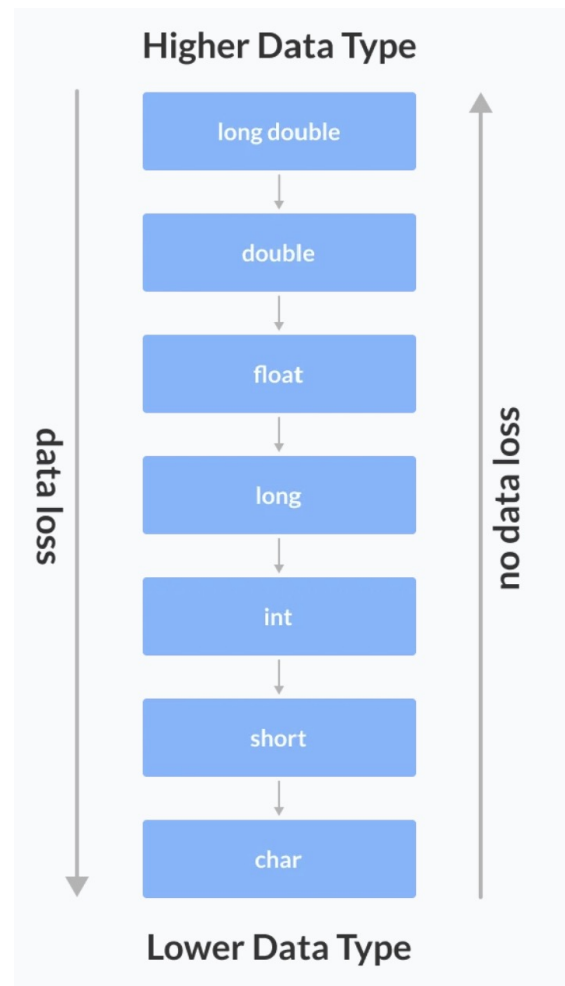
- ☐ `const_cast`

- ☐ `reinterpret_cast`

- ☐ Овие оператори ќе ги изучуваме подоцна кај  
објекто-ориентиран C++

# Проблеми со кастирање

- Недостаток:
- Можност за губење на податоци за време на конверзијата
  - Се случува кога правиме конверзија од поголем во помал податочен тип (пр. int во char)



# Проблеми со кастирање

- Недостаток:
- Можност за губење на податоци за време на конверзијата

Излез:

```
num_int = 356  
ch = d
```

```
#include <iostream>  
using namespace std;  
  
int main() {  
  
    int num_int = 356;  
    char ch;  
  
    ch = num_int;  
  
    cout << "num_int = " << num_int << endl;  
    cout << "ch = " << ch << endl;  
  
    return 0;  
}
```

ASCII кодот за знакот 'd' е 100???

Се случило преполнување - overflow

# Прашања?