4.5. Јамки- основни напомени и напредни концепти Структурирано програмирање

д-р Рамона Маркоска, вонр.професор

ramona.markoska@gmail.com

Содржини во ова поглавје ...

- Базични концепти на јамки за повторување
- Пре-тест и пост-тест концепти
- Иницијализација на јамки
- Ажурирања на состојба
- Тестови и контрола на повторувањата
- Концепти на WHILE, DO..WHILE, FOR
- Рекурзија

Јамки- Структурни наредби за повторување

- Наредби WHILE, DO … WHILE, FOR
- Бројот на повторувања е поврзан со (не)исполнувањето на некој услов.
- Условот може да биде формулиран да биде зависен од меѓурезултатите на некои пресметки или може да биде поврзан со програмски зададен број на повторувања.

- Секоја од наредбите може да се трансформира во некоја од другите

Што е јамка ? Видови на јамки ?

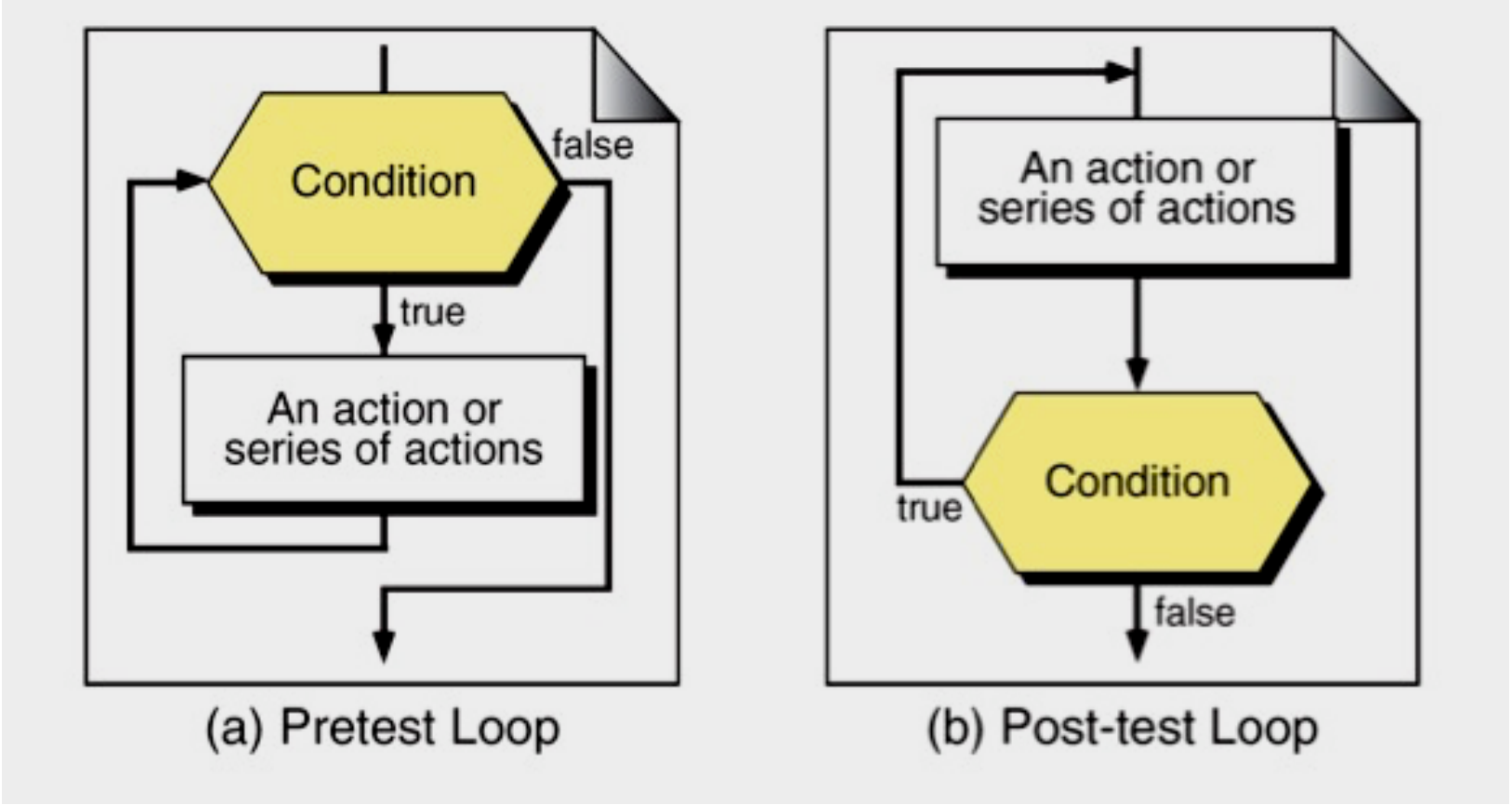
- Програмска структура која дозволува повторување на една акција се додека е исполнет еден зададен услов
- После секое повторување, наречено уште и итерација (приближување) се проверува состојбата на контролниот услов
- Се додека не се исполни условот, јамката со одредена програмска промена преминува во наредната итерација.
- Во моментот кога условот е исполнет, јамката завршува а програмата продолжува натаму

Видови на јамки

Во зависност од тоа во кој момент се врши тестирање на условот, постојат две базични категории на јамки:

а) Претест јамки

б) Пост тест јамки



Пре-тест јамка (pretest loop)

- Во секоја итерација програмата го тестира условот ПРЕД извршувањето на блокот наредби.
- Ако условот од тестот е TRUE (условот е исполнет) јамката продолжува со извршување на блокот на наредби.
- Ако условот на тестот е FALSE (условот не е исполнет), јамката завршува.
- Кај оваа јамка доколку условот не е исполнет уште на почеток може да се случи наредбите да не се извршат НИ ЕДНАШ.

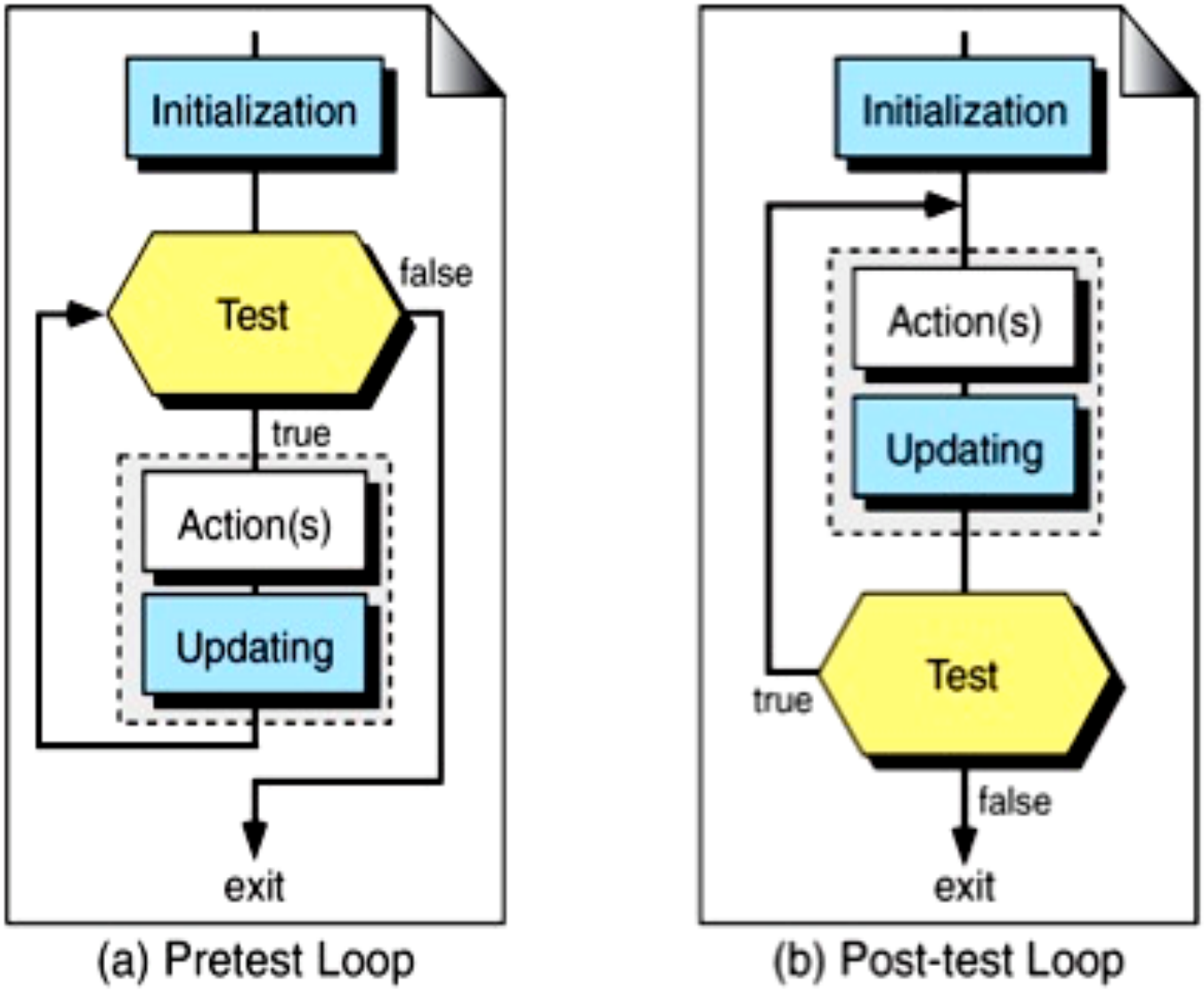
Пост-тест јамка (post-test loop)

- Во секоја итерација програмата прво го извршува блокот наредби а потоа го проверува условот.
- Се додека е условот TRUE се продолжува со итерациите.
- Кога условот постане FALSE извршувањето на јамката престанува.
- Во случај кога условот не е исполнет уште на самиот почеток, блокот наредби ќе се изврши БАРЕМ ЕДНАШ, т.е. пред првата проверка на условот.

Иницијализација и ажурирање на јамка

- Програмите бараат одреден вид на програмска подготовка како предуслов за извршување на јамките
- Иницијализација на јамката е постапка на подготовка на јамката, пред првата итерација.
- Иницијализацијата може, во зависност од начинот на изведување да биде:
  - Експлицитна- со сетирање на вредности за клучните варијабли кои се користат во јамката
  - Имплицитна- сетирањето е врз основа на некоја пре егзистирачка ситуација, предвидена да ја контролира јамката.
- Под ажурурање се подразбира промена на вредноста во секоја итерација во сличај кога условот е исполнет.
- Без постапката на ажурирање јамката би била т.н. бесконечна т.е. infinite loop

Иницијализација и ажурирање



Јамки контролирани од настан- event control loop

- Настан (event) е ***) некое случување внатре во извршувањето на јамката кое има за резултат промена на вредноста на контролниот израз од вистина во лага**.
- Програмата може да ја ажурира ваквата јамка и импицитно и експлицитно.
- Не е можно да се предвиди бројот на итерации (посебно не максималниот можен).

Јамки со сложени услови за одлучување- event control

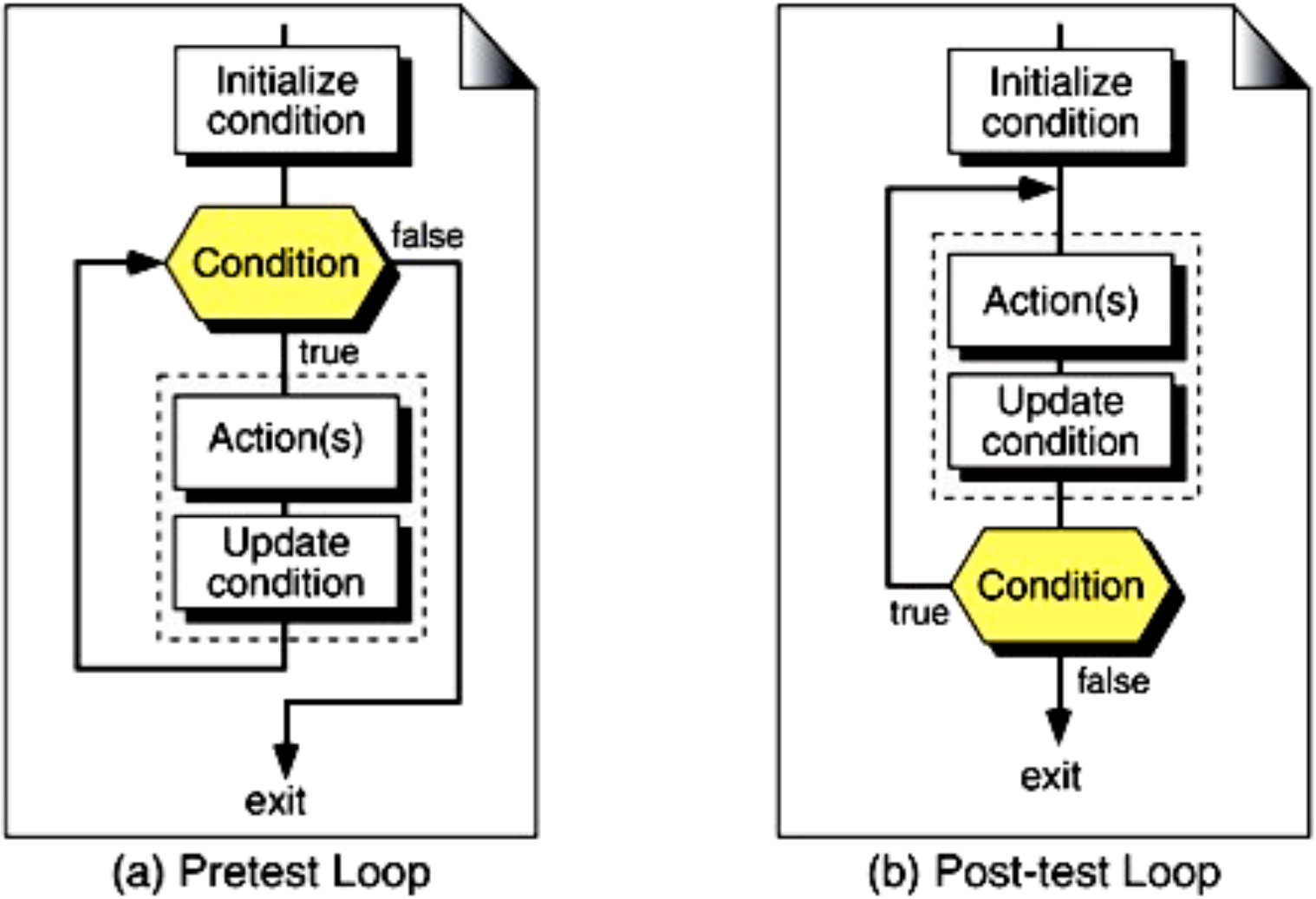
*)

```
for (int i = 0; i < 100 && !found; i++) {
   if (items[i] == "the one I'm looking for")
      found = true;
}
```

*)

```
{
    int i = 0;
    while (i < 100 && !found) {
        if (items[i] == "the one I'm looking for")
            found = true;
        i++;
    }
}
```

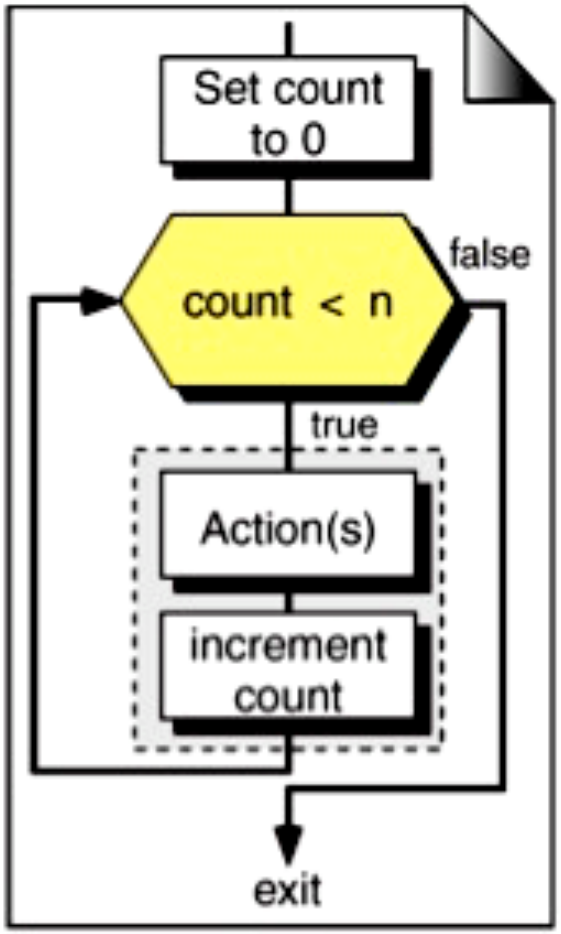
Концепт на јамки контролирани од настани



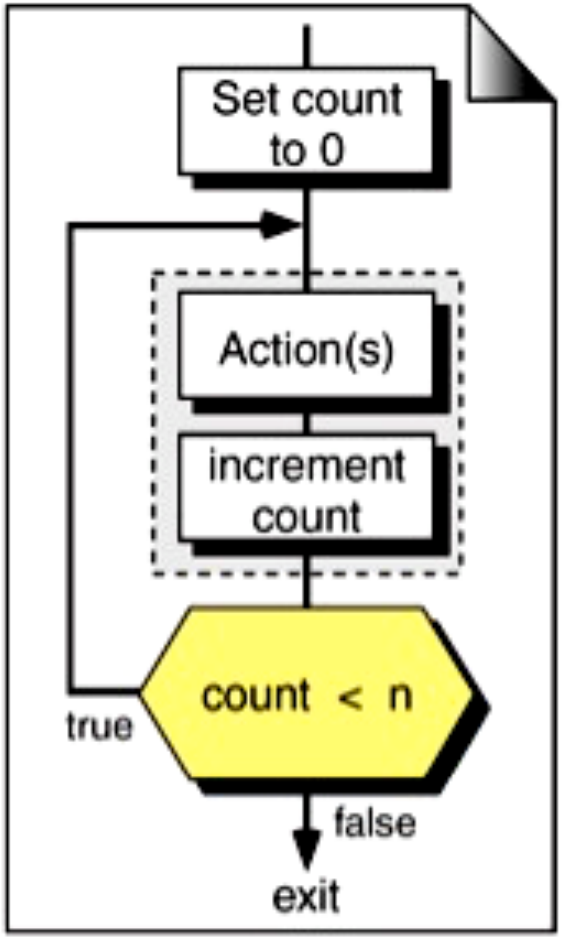
Јамки контролирани со бројач (count-ctr. loop)

- Главна одредница- можност за контрола на бројот на повторувања.
- За секоја јамка се користи бројач кој мора:
  - да се иницијализира- пред првата итерација на јамката
  - да се ажурира- после завршувањето на блокот наредби во кои се искористила тековната вредност на бројачот во блокот наредби во јамката.
  - да се тестира неговата вредност пред почетокот на секоја нова итерација.
- Ажурирањето на бројачот може да биде со инкрементирање или декрементирање
- Границите во кои се движи бројачот не мора да се константи, а може и да се услови

Концепт на јамки контролирани со бројач



(a) Pretest Loop



(b) Post-test Loop

Значење и примена на Process-Control јамка

- Бескрајна јамка, т.е. никогаш не завршува
- Примена: многу ретко за обработка на податоци, но често за бескрајни процеси (кои се по природа такви да немаат крај), работа на мрежни сервери, енвиро системи
- Нема потреба од ажурирање

Пример на контрола на процес со јамка

```
1   while (true)
2       {
3        temp = getTemperature();
4        if (temp < 68)
5            turnOnHeater();
6        else if (temp > 78)
7            turnOnAirCond();
8        else
9            {
10           turnOffHeater();
11           turnOffAirCond();
12           } // else
13      } // while true
```

Контролна јамка за регулирање на работен процес кај клима уред. (Степените се Фаренхајт)

Функции поврзани со управувањето

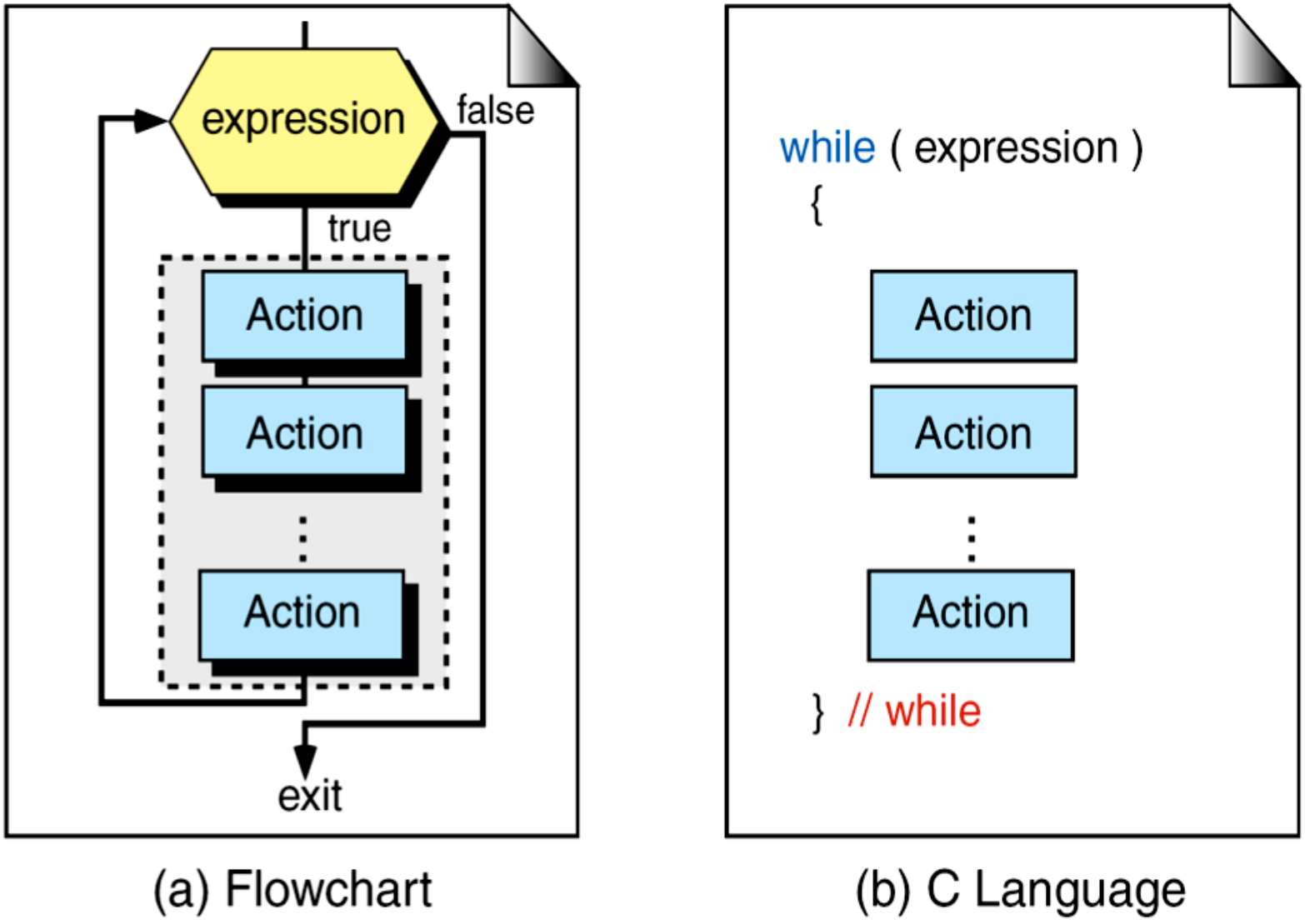
Јамки за печатење- Print Loops

- Една од многу честите примени на јамките во програмирање е користење за печатење на серии од разни ставки, информации, промелниви.
- Програмата сама може при работата да ги генерира сериите или..
- Сериите податоци може повремено како информации да се превземаат од друго место
- Можна е комбинација на овие две варијанти во смисол превеземните серии да се основа за генерирање на нови (кои ќе се печатат)

Јамки за data processing- File Loops

- Еден од најчестите начини за користење на јамки е за процесирање на подаотоци во фајл.
- Условот кој треба во тој случај да се користи и проверува во контекст на контрола на неговата вистинитост треба да има конфигурирано во себе системска константа EOF= end of file.
- Примена: функција која има за задача да го најде крајот на фајлот со цел да информира и прекине одредени процеси и активности вклучени во обработка и процесирање на фалот.

Јамка WHILE



(a) Flowchart

(b) C Language

Јамка WHILE- пример 1

```
#include <stdio.h>

main()
{
     int i = 10;

     while ( i > 0 )
     {
        printf("Hello %d\n", i );
        i = i -1;
     }
}
```

This will produce following output:

```
Hello 10
Hello 9
Hello 8
Hello 7
Hello 6
Hello 5
Hello 4
Hello 3
Hello 2
Hello 1
```

Јамка WHILE- пример 2 користење на break;

```
#include <stdio.h>

main()
{
    int i = 10;

    while ( i > 0 )
    {
       printf("Hello %d\n", i );
       i = i -1;
       if( i == 6 )
       {
          break;
       }
    }
}
```

This will produce following output:

```
Hello 10
Hello 9
Hello 8
Hello 7
```

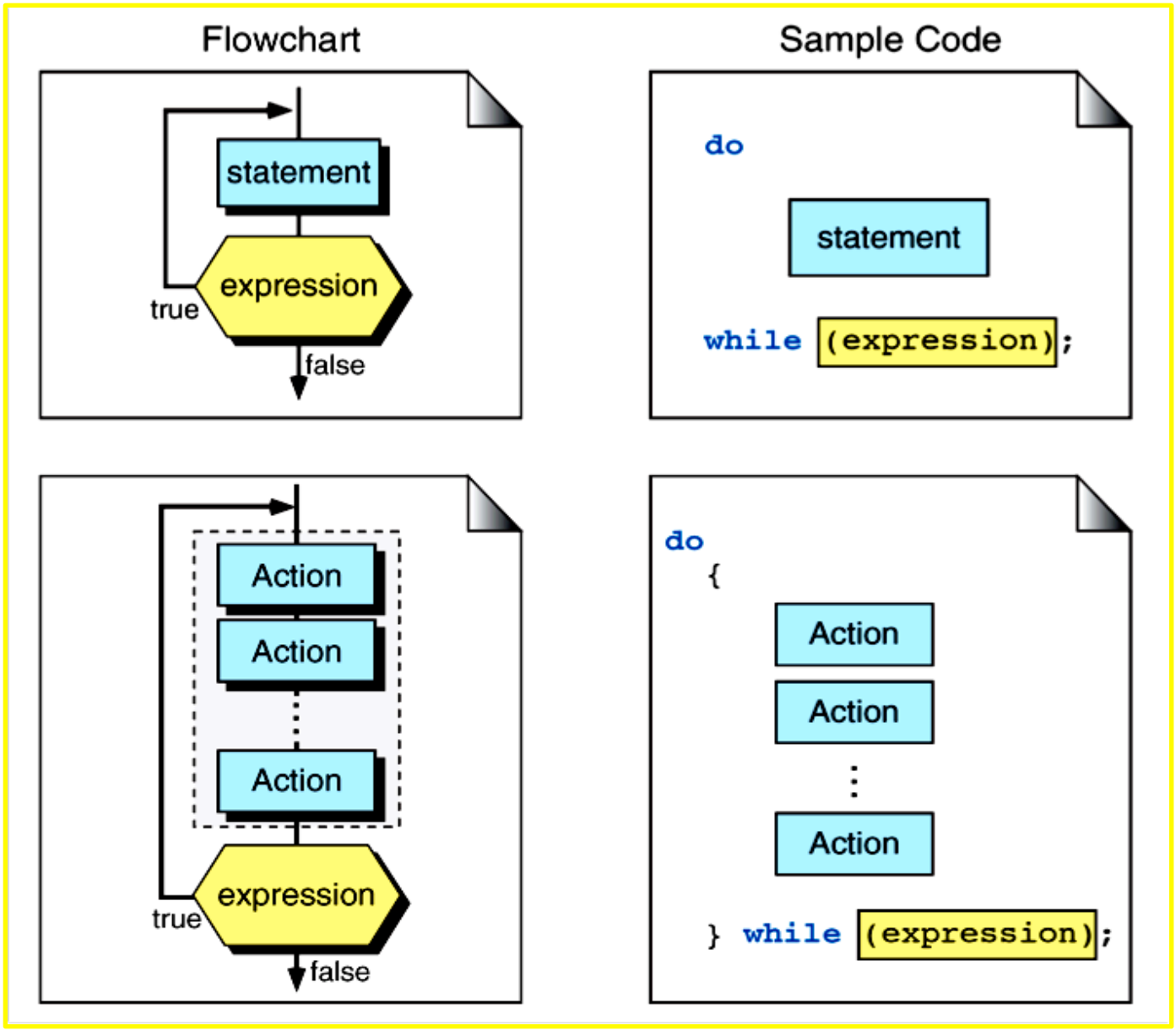
Јамка DO..WHILE

- Пост- тест јамка, слична со WHILE.
- Прво се извршуваат наредбите па потоа се проверува условот.
- Споредба!

WHILE	DO… WHILE

`int i=0;`
 	
`while (i<13)`
`{`
`  cout <<i++<<endl;`
`}`
	`int i=0;`
	
	`do`
	`{`
	`  cout <<i++<<endl;`
	`} while(i<13);`

Блок дијаграм на DO...WHILE



Јамка DO...WHILE- пример 3

```
#include <stdio.h>

main()
{
    int i = 10;

    do{
       printf("Hello %d\n", i );
       i = i -1;
    }while ( i > 0 );
}
```

This will produce following output:

```
Hello 10
Hello 9
Hello 8
Hello 7
Hello 6
Hello 5
Hello 4
Hello 3
Hello 2
Hello 1
```

Јамка DO..WHILE- пример4 со користење на break

```
#include <stdio.h>

main()
{
    int i = 10;

    do{
       printf("Hello %d\n", i );
       i = i -1;
       if( i == 6 )
       {
          break;
       }
    }while ( i > 0 );
}
```

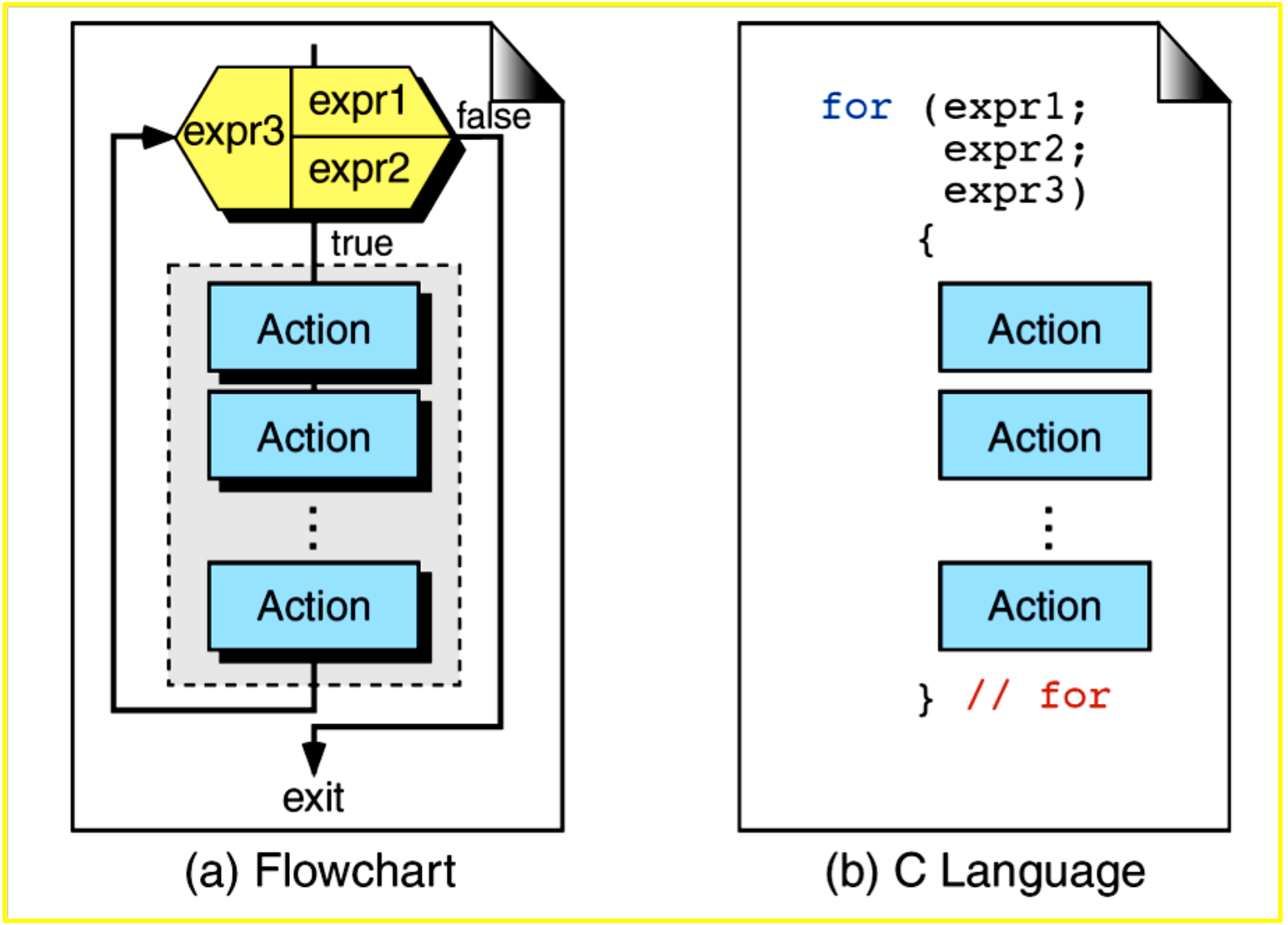
This will produce following output:

```
Hello 10
Hello 9
Hello 8
Hello 7
Hello 6
```

Јамка FOR

- Претест јамка која содржи 3 услови во насловниот почетен ред:
- Иницијализација на јамката.
- Тестен израз кој обично одредува колкав, т.е. д кога ќе се вршат повторувањата.
- Предвидена е можност за ажурирање и во самата јамка, иако не се препорачува.
- Најчесто се користи како бројач на состојби, заради можноста за контрола на бројот на итерации и заради можноста на нивно задавање и/или пресметување.
- Целосно компатибилна со преостанатите две.

Блок-дијаграм на јамка FOR



WHILE- FOR споредба

WHILE	FOR

`i = 1;`
`suma=0;`
`proizvod=1;`
 	
`while(i<=9)`
`{`
`  cin<<broj;`
`  suma+=broj;`
`  proizvod*=broj;`
`  i++;`
`}`
	`suma=0;`
	`proizvod=1;`
	
	`for(i=1; i<=9; i++)`
	`{`
	`  cin<<broj;`
	`  suma+=broj;`
	`  proizvod*=broj;`
	`}`

Споредба WHILE-FOR

- Кај сите јамки треба да се направи покрај декларација, исто и иницијализација на променливите кои ќе се ажурираат во јамката. Вредноста на иницијализација треба да се одбере така да НЕ влијае на крајниот резултат.
- Кај WHILE иницијализацијата на променливата од циклусот е НАДВОР., а кај FOR, иницијализацијата е прва во насовниот хедер.
- Кај WHILE, условот е зададен а кај FOR тоа е вториот дел од насловниот ред.
- Кај WHILE инкрементацијата на променливата која заедно со условот ја контролира јамката е како последна наредба пред следната проверка на условот, додека кај FOR, таа вредност е дадена во заглавјето, како трет финален услов.

Comma expression FOR облик

- Во одредени случаи е можно, променливата да се иницијализира внатре со самиот циклус
- Тогаш променливата која која се пресметува како резултат на ажурирањето , се наведува пред иницијализацијата на промената на циклусот.

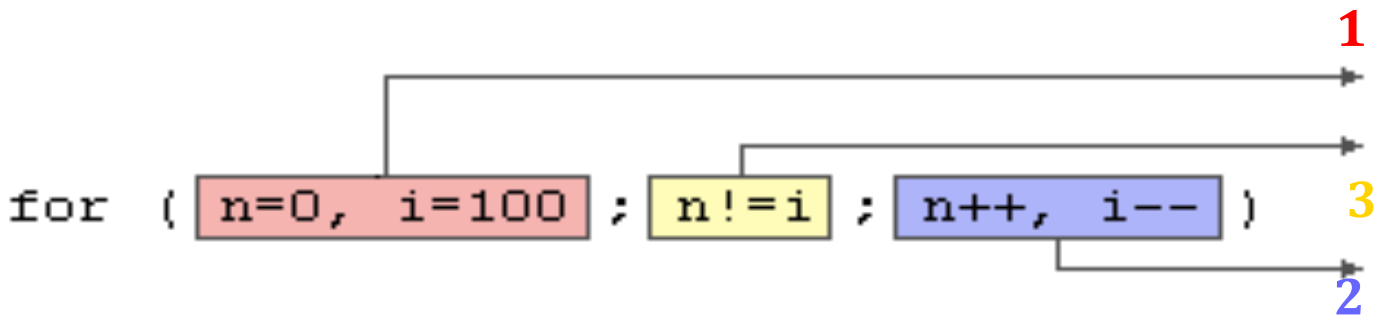
```
for (proizvod=1, i=1; i<=9; i++)

{
 cout<<broj<<endl;
 proizvod*=broj;
}
```

Multi control FOR јамка

Наместо една контролна променлива на циклусот, допуштена е синтактичка варијанта кога повеќе променливи имаат улога на циклусни промелниви.

1. Секоја од променливите има почетна иницијална вредност, и
2. Свој сопствен дефиниран тренд на промена.
3. Условот ја дава меѓузависноста на променливите и траењето на циклусот, во смисол дека циклусот се повторува се додека е ислоплнет условот

- Пример:



Значење и примена на break и continue

Break

- најчесто,во комбинација со IF, за предвремен прекин на јамката (да се избегава)

Continue

- Не ја прекинува јамката , туку ја пренсаочува контролата на условот за тестирање или ажурирање (зависно од типот на јамката)

- Во пракса да се избегнуваат овие две наредби !

Пример за програмска замена на Break

```
1  // A bad loop style
2  for ( ; ; )
3     {
4      …
5      if (condition)
6         break;
7     } // for
```

```
1  // A better loop style
2  for ( ; !condition ; )
3     {
4       …
5     } // for
```

```
1  while (x)
2     {
3      …
4      if (condition)
5         break;
6      else
7         …
8     } // while
```

```
1  while  (x && !condition)
2     {
3      …
4      if (!condition)
5         …;
6     } // while
```

Пример за програмска замена на Continue

```
1  float readAverage (void)
2  {
3  // Local Declarations
4  int   count = 0;
5
6  int   n;
7  float sum = 0;
8
9  // Statements
10 while(scanf("%d",&n)
11       != EOF)
12    {
13     if (n == 0)
14        continue;
15     sum += n;
16     count++;
17    }  // while
18
19 return (sum / count);
20 } // readAverage
```

```
float readAverage (void)
{
// Local Declarations
int   count = 0;

int   n;
float sum = 0;

// Statements
while(scanf("%d",&n)
       != EOF)
   {
    if (n != 0)
       {
        sum += n;
        count++;
       } // if
   } // while
return (sum / count);
} // readAverage
```

Задачи за вежбање и материјали

http://www.cplusplus.com/doc/tutorial/control/

http://mendo.mk/Lecture.do?id=11

4.1. и 4.2. споредени со 5.8-
4.3. и 4.4. споредба на решение со IF и SWITCH
4.5. Групирање сел.константи SWITCH
! Да се размисли, концепт за распоред !!!