Управляващи оператори в езика C++. Реализиране на разклонени алгоритми

Много често се налага да избираме между няколко възможни варианта за решаването на дадена задача. Какво решение ще вземем обикновено зависи от някакво условие. Това условие определя как ще продължи изпълнението на програмата. За да бъде изпълнена тази операция в програмните езици са въведени група оператори които под общото наименование (оператори за условно разклонение). В тази група спадат операторите `if .. else` и `switch case .. default`. `(?:)`

Операторите за избор приемат решение на базата на заключението от даден логически израз, който дава заключение истина или лъжа. Логическите изрази могат да се образуват на базата на операции, равенства и отношения.

Таблица 1 – Логически изрази в C++

Операция в C++	Условие	Коментар
==	a == b	a е равно на b
!=	a != b	a не е равно на b
>	a > b	a е по-голямо от b
<	a < b	a е по-малко от b
>=	a >= b	a е по-голямо или равно на b
<=	a <= b	a е по-малко или равно на b

Много често грешките в програмите се дължат на неправилно записване на логическите отношения. Интервалите между символите в операциите `==`; `!=`; `>=`; `<=`; водят до грешки. Така също трябва да се спазва и последователността на символите в операциите.

Пример:

`=!` `грешен запис;` `!=` `правилен запис;`

`=>` `грешен запис;` `>=` `правилен запис;`

`=<` `грешен запис;` `<=` `правилен запис;`

Небива да бъркате операцията `=` (операция за присвояване) с операцията `==` (операция за сравнение) тъй като това води до логическа грешка, т.е. програмата ще бъде компилирана но няма да работи правилно.

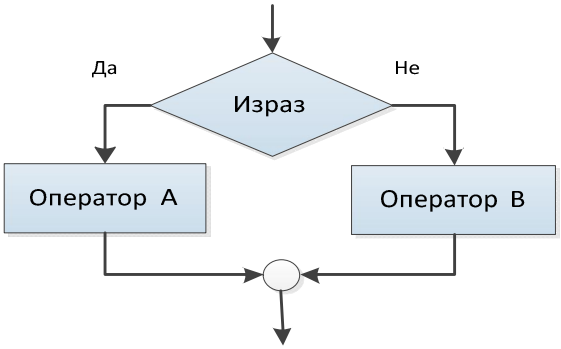
Условен оператор if else

Разклонението на изчислителните процеси в алгоритмите се извършва с помощта на условен оператор. Условен оператор се използва, когато се налага да се направи избор. Изборът може да бъде между две алтернативни възможности или да се избира да се изпълни ли даден оператор или да не се изпълни. В зависимост от вида на избора в езика C/C++ се използват два варианта на условен оператор: с пълен и непълен формат.

Условен оператор в пълен формат (if else).

Този оператор може да се представи в логическа форма посредством изречението:

Ако Израз тогава Оператор А иначе Оператор В;



Фиг.1. Условен оператор в пълен формат

Тук **Израз** е логически израз (има логическа стойност), **Оператор А** и **Оператор В** са оператори, които могат да бъдат прости или съставни (блокове). Схемата на тази конструкция е показана на фиг. 1. Записан като програмен оператор на C/C++, той има вида:

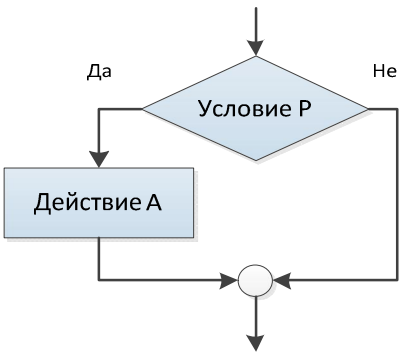
```
if( Израз ) {
    Оператор А
} else {
    Оператор В;
}
```

Действието на условния оператор в пълен формат е следното: проверява се стойността на логическия израз **Израз**; ако стойността е истина (**true**), се изпълнява **Оператор А**, а ако стойността е неистина (**false**) - се изпълнява оператор **Оператор В.**

Условен оператор в непълен формат (if).

Може да се представи в логическа форма посредством изречението:

Ако Израз тогава Оператор А;

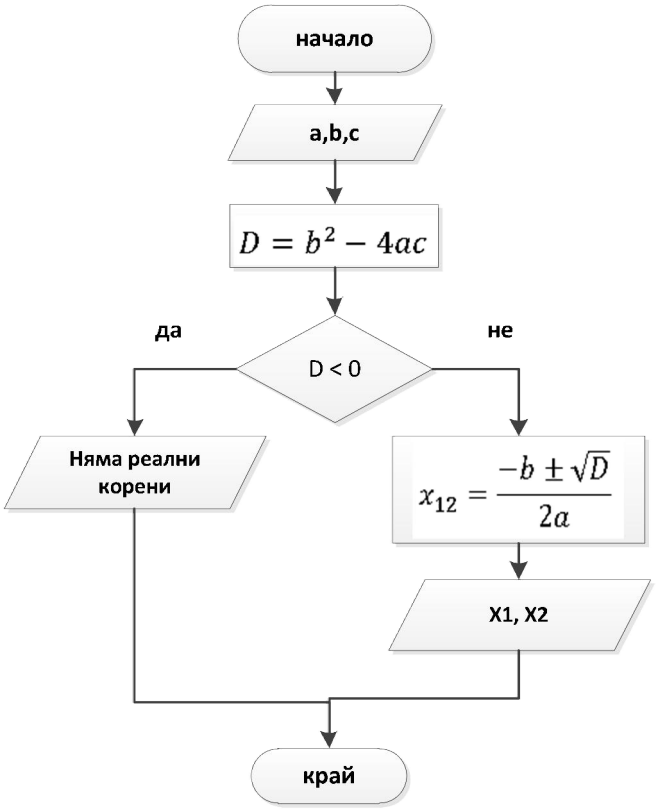


Фиг.2. Условен оператор в кратък формат

Стойността на логическия израз **Израз** определя дали да се изпълни оператора **Оператор А** или да не се изпълни. Ако стойността на **Израз** е истина (**true**), операторът **Оператор А** ще бъде изпълнен, а ако е лъжа (**false**), няма да бъде изпълни. Схемата на тази синтактична конструкция е показана на фиг. 2. Като програмен оператор на C/C++ той има видът:

```
if( Израз ) {
    Оператор А
}
```

Като пример за използване на условен оператор може да послужи един от най-често използваните примери за съставяне на компютърни програми - програма за решаване на квадратно уравнение. В примера (фиг.3) в частта за декларации са зададени като реални променливи **a**, **b**, **c** - коефициенти на квадратното уравнение; **D** - дискриминанта на уравнението и **x1** и **x2** - корени на уравнението.



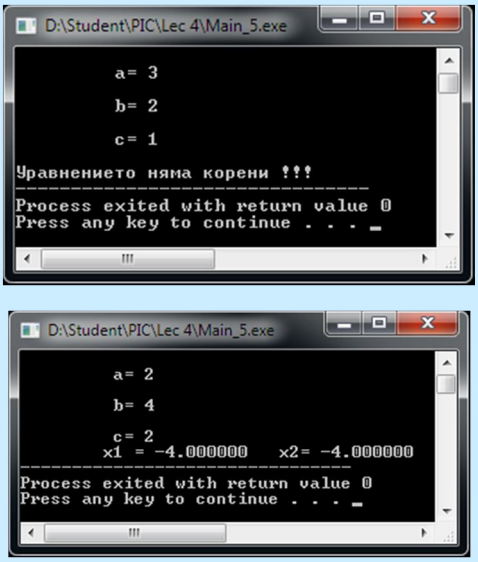
Фиг.3. Алгоритъм за решаване на квадратно уравнение

Това не е най-добрата програма за решаване на квадратно уравнение, но тя илюстрира възможностите на условния оператор. Тук е използван условен оператор в пълен формат, като роля на логически израз играе проверката за положителна стойност на дискриминантата на уравнението **D < 0.** Както се вижда, в C не е необходимо логическият израз да бъде ограден в скоби, но когато се използват по-сложни изрази и се налага използването на скоби за въвеждане на приоритети за логическите операции, може и целият логически израз да бъде ограден в скоби. В примера ролята на оператор **Оператор A** играе функцията за извеждане на информация

```
printf ("\nУравнението няма корени !!!");
```

А на **Оператор B** - съставният оператор, състоящ се от трите оператора, оградени с логическите скоби **{}**.

Програмен код	Резултат

`#include <iostream>`
`#include <stdio.h>`
`#include <math.h>`
`int main()  {`
`  double a,b,c,D,x1,x2;`
`  setlocale(LC_ALL, "Bulgarian");`
`  printf("\n\t a= "); scanf("%lf",&a);`
`  printf("\n\t b= "); scanf("%lf",&b);`
`  printf("\n\t c= "); scanf("%lf",&c);`
`  D= b*b - 4*a*c;`
`  if( D < 0)`
`   printf ("\nУравнението няма корени!!!");`
`  else {`
`    x1 = (-b+ sqrt(D))/2*a;`
`    x2 = (-b- sqrt(D))/2*a;`
`    printf("\a\tx1 = %f   x2= %f",x1,x2);`
`  }`
`}`
	

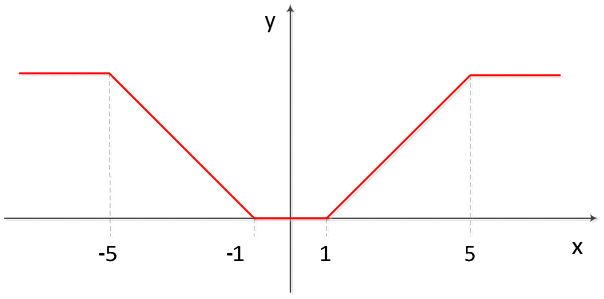
Вложение на условни оператори

Всички оператори в частта след **if** и след **else** могат да бъдат произволни оператори, включително и други условни оператори. Тогава се получава вложение на условни оператори. Това дава възможност да се направи избор между повече от две възможности. Тъй като всеки от условните оператори може да бъде в пълен или непълен формат, понякога се създават условия за неоднозначно тълкуване. Проблемите, възникващи при вложение на условни оператори, могат да се пояснят с примери:

Пример 1:

Да се състави програма за определяне стойността на следната функция:

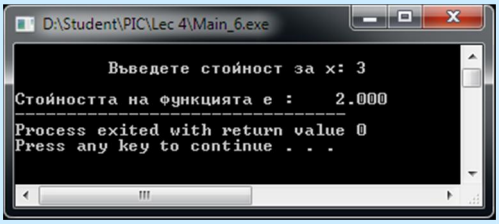
$$y(x)=\begin{cases}4\cdot \Rightarrow за\cdot x<-5;\\ x+1\cdot \Rightarrow за\cdot x\in[-5,-1];\\ 0\cdot \Rightarrow за\cdot x\in[-1,-1];\\ x-1\cdot \Rightarrow за\cdot x\in[-1,1];\\ 4\cdot\cdot \Rightarrow за\cdot x\ge 5\end{cases}$$



Фиг. 4.

Програма за решаване на задачата с използване на условен оператор изглежда по следния начин:

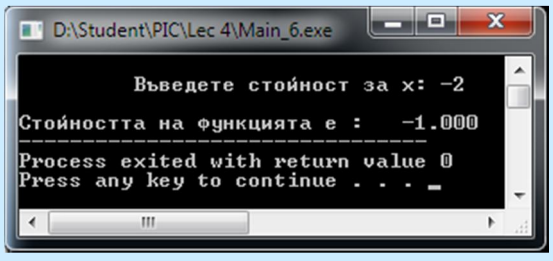
Програмен код	Резултат

<pre>#include <iostream>
#include <stdio.h>
#include <math.h>
int main() {
 double x,y;
 setlocale(LC_ALL, "Bulgarian");
 printf("\n\t Въведете стойност за x x: ");
 scanf("%lf",&x);
 if(x < -5 || x >= 5) {
 y = 4;
 } else {
 if(x >= -5 && x < -1) {
 y = x+1;
 } else {
 if(x >= -1 && x <= -1) {
 y = 0;
 } else {
 if(x > 1 && x < 5) {
 y = x-1;
 }
 }
 }
}
 printf("\nСтойността на функцията е: %8.3f",y);
}</pre>
	

Особеност в тази програма е вложението на четири условни оператора. За по-голяма яснота скобите ({ }) за даден съставен оператор са поставени една под друга, за да се подчертае за кой оператор **else** към кой **if** се отнася. Ако не е записан така, осмислянето на този оператор би било доста трудно. За компилатора на език C/C++ съществува правилото, че оператор **else** винаги се отнася към най-близкия **if**, който не е завършен със стоящ по-напред **else**.

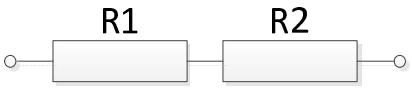
Програмата може да бъде реализирана и без използване на вложени условни оператори.

Програмен код	Резултат

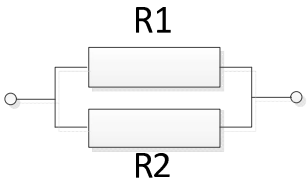
<pre>#include <iostream>
#include <stdio.h>
#include <math.h>
int main() {
 double x,y;
 setlocale(LC_ALL, "Bulgarian");
 printf("\n\t Въведете стойност за x x: ");
 scanf("%lf",&x);
 if(x < -5 || x >= 5) y = 4;
 if(x >= -5 && x < -1) y = x+1;
 if(x >= -1 && x <= -1) y = 0;
 if(x > 1 && x < 5) y = x-1;
 printf("\nСтойността на функцията е: %8.3f",y);
}</pre>
	

Очевидно е че вторият алгоритъм е по-кратък по запис и би трябвало той да бъде предпочетен. Но при оценката на алгоримите се отчита не само колко заемат в памет, а и за колко време се изпълняват. В това отношение (скорост на изпълнение) вторият алгоритъм отстъпва. Бихте ли казали защо е така?

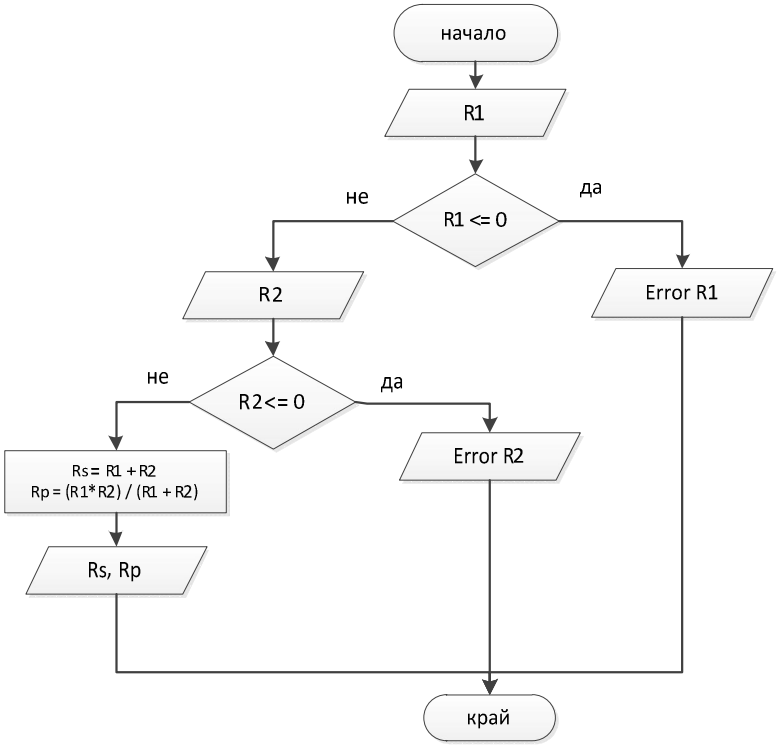
Пример: Да се напише програма, която изчислява еквивалентното съпротивление на два резистора, със стойности съответно R1 и R2, при последователно и паралелно свързване. Изчислението да се извършва само, ако въведените стойности са положителни числа.



$$R_e = R_1 + R_2$$



$$\frac{1}{R_e} = \frac{1}{R_1} + \frac{1}{R_2}$$



Фиг. 5 Алгоритъм на задачата

За решаването на задачата се спираме на алгоритъма, илюстриран с блоковата схема на фиг.5. Ако стойностите на R1 и R2 са равни на нула или по-малки от нула, изпълнението на програмата се прекратява.

Програмен код	Резултат

`#include <iostream>`
`using namespace std;`
`int main(void){`
`  float R1,R2,Rs,Rp;`
`  cout<<"R1=";  cin>>R1;`
`  if (R1<=0)  {`
`  cout<<"R1<=0, R1="<<R1<<"\n";`
`  return 1;`
`  }`
`  cout<<"R2=";  cin>>R2;`
`  if (R2<=0) {`
`  cout<<"R2<=0, R2="<<R2<<"\n";`
`  return 2;`
`  }`
`  Rs = R1+R2;`
`  Rp = R1*R2/Rs;`
`  cout << "Rs="<<Rs<<"\n";`
`  cout << "Rp="<<Rp<<"\n";`
`  return 0;`
`}`	

Оператор на многопосочно разклонение switch case default

В програмирането често се срещат ситуации изискващи многократно повторение на условен оператор. Много от тези случаи могат да се решат по-бързо и елегантно с оператора **switch** вместо използването на многократно вложени **if else** оператори. Ето един такъв пример, реализиран в двата варианта:

- С помощта на условен оператор

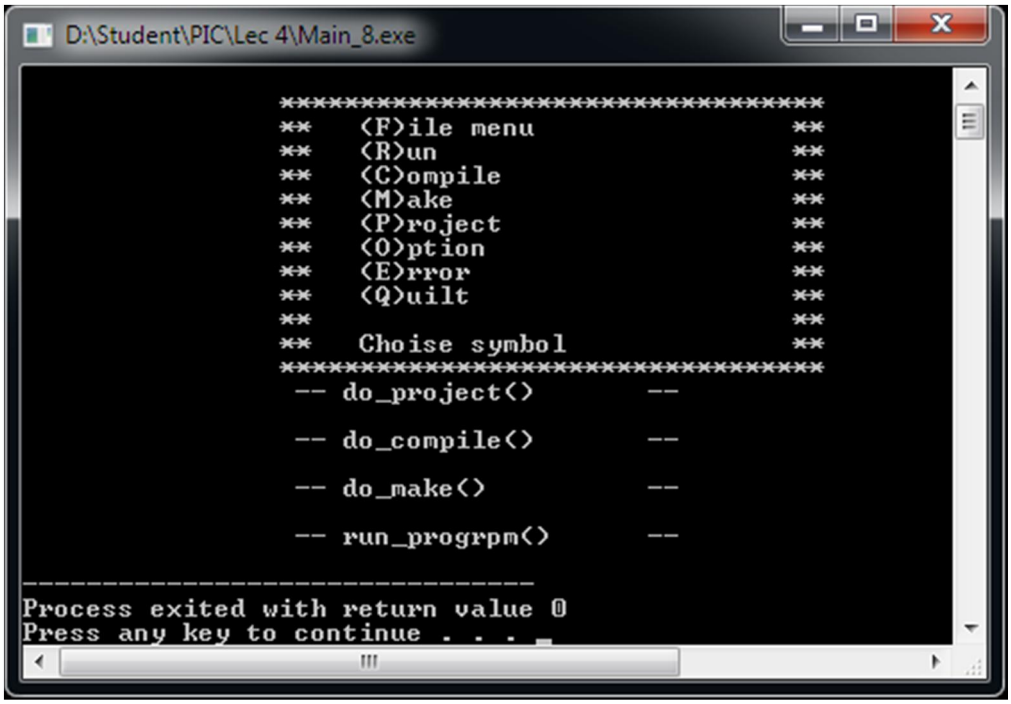
Програмен код

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
void do_main_menu(short *done);
int main()
{
     short done;
     do_main_menu(&done);
     return 0;
}
void do_main_menu( short *done) {
  char cmd;
  *done = 0;

    printf("\n\t\t**********************************");
    printf("\n\t\t**    (F)ile menu               **");
    printf("\n\t\t**    (R)un                     **");
    printf("\n\t\t**    (C)ompile                 **");
    printf("\n\t\t**    (M)ake                    **");
    printf("\n\t\t**    (P)roject                 **");
    printf("\n\t\t**    (O)ption                  **");
    printf("\n\t\t**    (E)rror                   **");
    printf("\n\t\t**    (Q)uilt                   **");
    printf("\n\t\t**                              **");
    printf("\n\t\t**   Choise symbol              **");
    printf("\n\t\t**********************************");
  do {
    cmd = toupper( (char)getch());
    if (cmd == 'F')       do_main_menu( done);
    else  if (cmd  == 'R') printf("\n\t\t -- run_progràm()      --\n");
    else  if (cmd  == 'C') printf("\n\t\t -- do_compile()      --\n");
    else  if (cmd  == 'M') printf("\n\t\t -- do_make()         --\n");
    else  if (cmd  == 'P') printf("\n\t\t -- do_project()      --\n");
    else  if (cmd  == 'O') printf("\n\t\t -- do_options()      --\n");
    else  if (cmd  == 'E') printf("\n\t\t -- do_errors()       --\n");
    else  if (cmd  == 'Q') *done = 1;
    else                   printf("\n\t\t -- hàndle_others(cmd, done)");
  } while (!*done);
}
```

- с помощта на оператор **switch:**

Програмен код

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
void do_main_menu(short *done);
int main()
{
     short done;
     do_main_menu(&done);
     return 0;
}
void do_main_menu( short *done) {
  char cmd;
  *done = 0;

    printf("\n\t\t**********************************");
    printf("\n\t\t**    (F)ile menu               **");
    printf("\n\t\t**    (R)un                     **");
    printf("\n\t\t**    (C)ompile                 **");
    printf("\n\t\t**    (M)ake                    **");
    printf("\n\t\t**    (P)roject                 **");
    printf("\n\t\t**    (O)ption                  **");
    printf("\n\t\t**    (E)rror                   **");
    printf("\n\t\t**    (Q)uilt                   **");
    printf("\n\t\t**                              **");
    printf("\n\t\t**    Choise symbol             **");
    printf("\n\t\t**********************************");
  do {
   cmd = toupper( (char)getch());
   switch(cmd) {
    case  'F' :       do_main_menu( done);                    break;
    case  'R' : printf("\n\t\t -- run_progràm()      --\n"); break;
    case  'C' : printf("\n\t\t -- do_compile()       --\n"); break;
    case  'M' : printf("\n\t\t -- do_make()          --\n"); break;
    case  'P' : printf("\n\t\t -- do_project()       --\n"); break;
    case  'O' : printf("\n\t\t -- do_options()       --\n"); break;
    case  'E' : printf("\n\t\t -- do_errors()        --\n"); break;
    case  'Q' : *done = 1;                                  break;
    default   : printf("\n\t\t -- hàndle_others(cmd,done)"); break;
   }
  } while (!*done);
}
```



Функцията **do_main_menu** чете в цикъл символ, въведен от клавиатура и го присвоява на променливата **cmd** (ако въведеният символ е малка буква, той първо се преобразува в главна от функцията **toupper**). Вторият вариант използва оператор **switch**, който предава управлението на различни оператори в зависимост от стойността на **cmd**. Цикълът завършва, когато на променливата **done** се присвои стойност 0. Операторът **switch** взема стойността на **cmd** и я сравнява с всеки от етикетите след **case**. Ако има съвпадение, изпълнението започва от етикета и продължава докато срещне оператор **break** или края на оператор **switch**. Когато няма съвпадащ етикет, ще се изпълни оператор **default**, а ако той не е включен, целият оператор **switch** ще се пропусне. Управляващата стойност в оператор **switch** (тук променливата **cmd**) трябва да бъде от тип съвместим с цял тип (да се превръща лесно в цял тип). Следователно тя може да бъде от тип **int** и всичките му разновидности, от тип **char** или **enum**. Не може да се използват реални стойности (**float** или **double**), указатели, символни низове или други структурирани данни (но може да се използва елемент от структура, съвместим с данните от цял тип). Управляващата стойност може да бъде стойността на произволен израз (константа, променлива, обръщение към функция или тяхна комбинация). Етикетите трябва да са константи. освен това след служебната дума **case** може да стои само един етикет. Ако се налага да се изброят няколко етикета, те се поставят един след друг.

Например:

```
switch (cmd) {
  case  'f':
  case  'F': printf("do_file_menu(done)\n");break;
  case  'r':
  case  'R': printf("run_program()  \n");   break;
}
```

В случая "**do_file_menu(done)**" ще се отпечата, когато се въведе буквата **f** или **F** (приемаме, че не е използвана функция **toupper**). Запомнете, че групата оператори след **case** се отделя от следващите с оператор **break** (последователното обработване на оператори продължава до срещнат оператор **break**). В примера, ако след **case 'F'**:няма **break**, при стойност **'f'** на **cmd** ще се изпълнят и операторите, предвидени за **'r'** и **'R'**. Има разбира се случаи, когато **break** се пропуска съзнателно.

Например:

```
typedef enum { sun, mon, tues,wed, thur, fri, sat } days;

void main() {
  days today;
  today = sun;
  switch (today) {
    case mon:
    case tues:
    case wed:
    case thur:
    case fri:  puts("Работа"); break;
    case sat:
    case sun:  puts ("Почивка");
  }
 }
```

Условен израз (?:)

Има случаи, в които на базата на определено условие избираме между два израза (респективно техните стойности). Това действие обикновено се извършва с помощта на **if...else.**

Например:

```
int imin(int a, int b);
void main()
{
   printf("По-малкото число е %d",imin(5,3));
}
int imin(int a, int b)
{
   if (a<b) return (a);
   else return (b);
}
```

Ситуацията се среща много често и затова C++ предвижда специална конструкция, осигуряваща съкратен запис. Форматът е:

Израз1 ? Израз2 : Израз3

Той се интерпретира така: Ако **Израз1** има стойност "истина", да се изпълни **Израз2** и целият израз да приеме неговата стойност. В

противен случай да се изпълни **Израз3** и стойността му да стане стойност на целия израз. Следователно горният пример може да се запише по следния начин:

```
int imin(int a, int b);
void main()
 {
  printf("\n\t\a По-малкото число е %d",imin(5,3));
 }

int imin(int a, int b){
  return ((a<b) ? a : b);
 }

float imin(float a, float b);

void main()
{
 printf("\n\t\a По-малкото число е %f",imin(5.8,3.7));
}

float imin(float a, float b)
 {
  return ((a<b) ? a : b);
 }
```

Така, щом програмата срещне израза **imin(e1,e2)**, тя го замества с **((e1<e2) ? e1 : e2)** и компилацията й продължава. Това действително е по-добро и общо решение, тъй като вече не е задължително **e1** и **e2** да са от тип **int**. Те могат да бъдат от произволен тип, който позволява прилагането на операцията **"<"**.

Примери

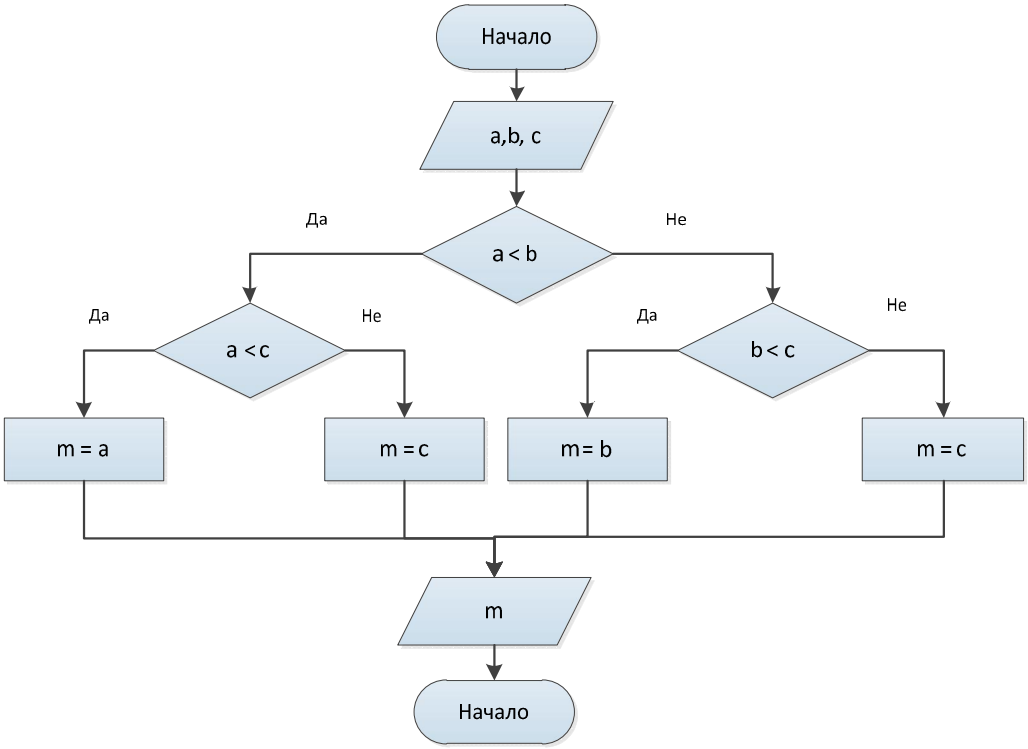
Тук ще покажем програмното решение на задачите от лекцията за алгоритмизирането.

Задача 1: Да се синтезира алгоритъм за намиране на най-малкото от три числа.

Така поставената задача може да бъде решена по два начина. При първият се изпълняват следните стъпки:

Стъпка 1: Начало.
Стъпка 2: Въведете и запомнете стойностите на a, b и c.
Стъпка 3: Проверете дали a < b
Стъпка 4: Ако a < b, проверете дали a<c
Стъпка 5: Ако a<c на m присвоете стойността на a, ако не е присвоете на m стойността на c
Стъпка 6: Ако a ≥ b, проверете дали b<c
Стъпка 7: Ако b<c на m присвоете стойността на b, ако не е присвоете на m стойността на c
Стъпка 8: Изведете стойността на m като резултат.
Стъпка 9: Край.

Алгоритмът е даден на фиг. 6

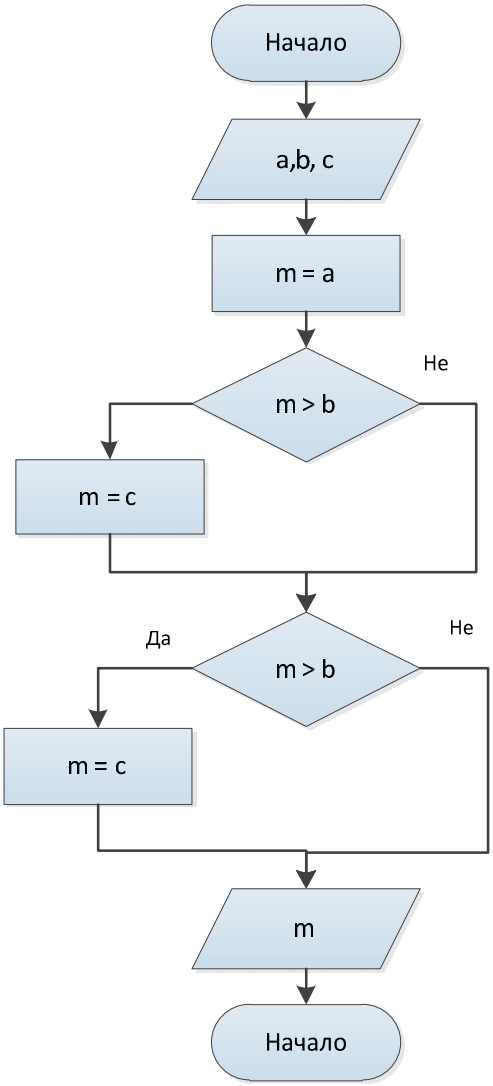


Фиг.6. Алгоритъм на програма за намиране на най-малкото от 3 числа

Втори вариант на алгоритъма е показан на фиг. 7. Опитайте се да запишете стъпките за изпълнението му.

Програмен код	Резултат

```c
#include <stdio.h>
 int main(void) {
  double a,b,c,m;
  printf("a = ");
  scanf("%lf",&a);
  printf("b = ");
  scanf("%lf",&b);
  printf("c = ");
  scanf("%lf",&c);
  if( a < b) {
    if ( a < c ) m = a;
    else         m = c;
  } else {
    if ( b < c ) m = b;
    else         m = c;
  }
  printf ( "m = %f\n",m);
  return 0;
 }
```



Фиг.7. Алгоритъм на програма за намиране на най-малкото от 3 числа – втори вариант

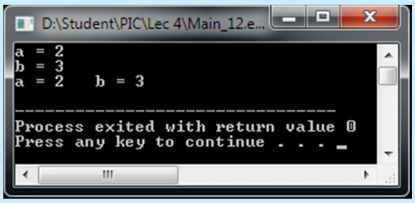
Програмен код	Резултат

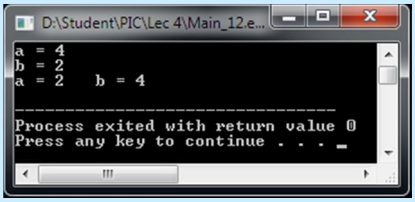
`#include <stdio.h>`
`int main(void) {`
`  double a,b,c,m;`
`  printf("a = ");`
`  scanf("%lf",&a);`
`  printf("b = ");`
`  scanf("%lf",&b);`
`  printf("c = ");`
`  scanf("%lf",&c);`
`  m = a;`
`  if ( m > b)  m = b;`
`  if ( m > c )  m = c;`
`  printf ( "m = %f\n",m);`
`  return 0;`
`}`	

Задача 2: Да се напише if-конструкция,която изчислява стойността на две целочислени променливи и сменя техните стойности ,ако стойността на първата променлива е по-голяма от втората.

Програмен код	Резултат

```c
#include <stdio.h>
 int main(void) {
  int a, b, c;
  printf("a = ");
  scanf("%d",&a);
  printf("b = ");
  scanf("%d",&b);
  if ( a > b)     {
     c = a;
     a = b;
     b = c;
  }
  printf ( "a = %d    b = %d\n",a,b);
  return 0;
 }
```



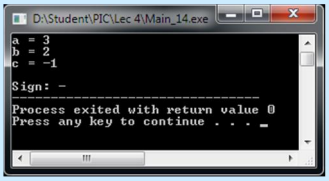


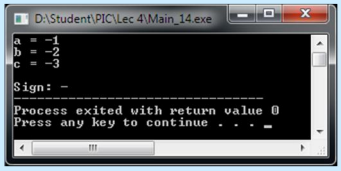
За решаване на задачата е необходимо да бъде въведена още една междинна променлива(в случая това е **c**). Нейната задача е да запази стойността на **a** (c = a) и след като в a се запише стойността на **b** (a=b), да прехвърли стойността си в **b**(b = c). Тази операция се използва много често при алгоритмите за сортиране на масиви.

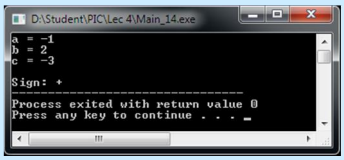
Задача 3: Напишете програма,която показва знака (+ или -) от частното на три реални числа, без да ги пресмята. Използвайте последователност от if-конструкции.

Програмен код	Резултат

```c
#include <stdio.h>
 int main(void) {
  float a, b, c;
  printf("a = ");
  scanf("%f",&a);
  printf("b = ");
  scanf("%f",&b);
  printf("c = ");
  scanf("%f",&c);
  //puts("\nSign: ");
  printf("\nSign: ");
  if ( a < 0 ) {
    if ( b < 0) {
        if ( c < 0) putchar('-');
        else putchar('+');
     } else {
        if ( c < 0) putchar('+');
        else putchar('-');
     }
    } else {
      if ( b < 0) {
        if ( c < 0) putchar('+');
        else putchar('-');
     } else {
        if ( c < 0) putchar('-');
        else putchar('+');
     }
    }
  return 0;
 }
```







Задачи за самоподготовка

1. Напишете програма,която намира най-голямото по стойност число,измежду три дадени числа. Използвайте вложени if конструкции.
2. Напишете програма,която сортира стойностите на няколко реални числа в намаляващ ред. Използвайте вложени if конструкции.
3. Напишете програма,която за дадена цифра (0-9), зададена като вход, извежда името на цифрата на български език на изхода. Използвайте switch конструкция.
4. Напишете програма,която намира най-голямото по стойност число от 5 дадени числа.

Въпроси за самоконтрол

1. Какви стойности за **a** и **b** от тип **int** ще се изведат на екрана при изпълнението на следните редове, ако за **a** и **b** се въведат следните стойности :

a) **a** = 5, **b** = 3;
b) **a** = -5, **b** = 3;

```
if (a>0){
  a=a+20;
  b=a/3;
}
else b=a-20;
cout<<"b="<<b<<"\na="<<a;
```

2. Каква стойност за **b** ще се изведе на екрана при изпълнението на следните редове, ако за **a** и **b** се въведат следните стойности :

a) **a** = 5, **b** = 3;
b) **a** = 5, **b** = -3;
c) **a** = -5, **b** = 3;
d) **a** = -5, **b** = -3?

```
if (a>0)
if (b>0) b=a+20;
else b=a-20;
cout<<"b="<<b;
```

3. Каква стойност за **s** ще се изведе на екрана при изпълнението на следните редове, ако за **n** , **m** и **x** се въведат следните стойности :

a) **n** = 5, **m** = 2.5, **x** = 1;
b) **n** = 1, **m** = 2.5, **x** = 5;
c) **n** = 5.5, **m** = 1, **x** = 3;
d) **n** = 5, **m** = 2.5, **x** = 2.

```
switch (x)
{
  case 1: s=(n+m)/5;
  case 2: s=n+m/5;
  case 3: s=(n-m)/5;
  default: s=0;
}
cout<<"s="<<s;
```

4 Каква стойност за **s** ще се изведе на екрана при изпълнението на следните редове , ако за **n** , **m** и **x** се въведат следните стойности :

a) **n** = 5, **m** = 2.5, **x** = 1;
b) **n** = 1, **m** = 2.5, **x** = 5;
c) **n** = 5.5, **m** = 1, **x** = 3;
d) **n** = 5, **m** = 2.5, **x** = 2?

```
switch (x)
{
  case 1:  s = (n+m)/5; break;
  case 2:  s = n+m  /5; break;
  case 3:  s = (n-m)/5; break;
  default: s = 0;
}
cout<<"s="<<s;
```