თემა 4. პროგრამის მართვის ინსტრუქციები

* მინიჭების შეტყობინება
* შეტყობინებები case და switch
* შეტყობინებები if,if-else

**მინიჭების შეტყობინება (Assignment statements)**

იმპერატიული პარადიგმის ენებში არსებობს მხოლოდ ერთი შეტყობინება (statement), რომლის შესრულება ცვლის მონაცემთა ერთობლიობას, ე.ი. ცვლის მდგომარეობას – ეს არის მინიჭების შეტყობინება. ყველა დანარჩენი შეტყობინება (როგორიცაა განშტოება, ამორჩევა, განმეორება და პროცედურების გამოძახება) სინამდვილეში ემსახურება მინიჭებების თანმიმდევრობების შესრულების მართვას.

მიუხედავად თავისი სინტაქსური სიმარტივისა, მინიჭება რთული პროცესია და წყვეტს სამ განსხვავებულ ამოცანას:

1. გამოითვლება შეტყობინების მარჯვენა მხარეს მდებარე გამოსახულება.

2. გამოითვლება შეტყობინების მარცხენა მხარეს მდებარე გამოსახულება – მან უნდა განსაზღვროს მეხსიერების უჯრედის მისამართი.

3. პირველ ბიჯზე მიღებული მნიშვნელობის კოპირება ხდება მეხსიერების უჯრედებში, დაწყებული მისამართიდან, რომელიც გამოთვლილია მე–2 ბიჯზე.

ასე რომ, მინიჭების შეტყობინება a ( i + 1 ) = b + c; განსაზღვრავს რთულ გამოთვლებს.

მინიჭების შესრულებას შეგვიძლია გაუკეთოთ ინტერპრეტირება დაბალ დონეზე. რადგან განსხვავებული კომპიუტერების პროცესორებს აქვთ რეგისტების განსხვავებული ჯგუფები, და სხვადასხვა ასამბლერის ბრძანებები შეიძლება სრულიად განსხვავდებოდეს, ჩვენ შემოვიღებთ ასემბლერის ბრძანებებს ჰიპოთეტური კომპიუტერისთვის - ფსევდო ასემბლერს. ძირითადი ბრძანებები იყოფა შემდეგ ჯგუფებად:

* მეხსიერებაზე წვდომა: load – მანქანური სიტყვის ჩატვირთვა რეგისტრში; store – რეგისტრის შიგთავსის ჩაწერა მეხსიერების სიტყვაში
* არითმეტიკული ოპერაციები: add, sub და ა.შ. მოქმედებები სრულდება ორი რეგისტრის შიგთავსებზე, ან რეგირსტრის და მეხსიერების სიტყვის შიგთავსებზე. შედეგი რჩება რეგისტრში
* შედარება და გადასვლა. მაგალითად,

jump\_eq R1, L1

...

L1: ...

ნიშნავს L1 ჭდით მონიშნულ ბრძანებაზე გადასვლას, თუ R1 რეგისტრის შიგთავსი უდრის 0 –ს, ან შემდეგი ბრძანების შესრულებას წინააღმდეგ შემთხვევაში.

მაგალითად, ქვემოთ მოცემული ინსტრუქციები ნიშნავს შემდეგს:

load R1,54 – R1 რეგისტრში ჩავტვირთოთ მნიშვნელობა მეხსიერების მისამართიდან 54;

load R2,#54 – R2 რეგისტრში ჩავტვირთოთ მნიშვნელობა 54;

load R3,&N – R3 რეგისტრში ჩავტვირთოთ N ცვლადის მისამართი;

load R4,N – R4 რეგისტრში ჩავტვირთოთ N ცვლადის მნიშვნელობა.

C –ის გამოსახულება n++ შესრულდება ასე:

load R1,n – R1–ში ჩაიტვირთა n –ის მნიშვნელობა;

add R1,#1 – R1–ის მნიშვნელობას დაემატა 1 –იანი;

store R1,n – R1–ის შიგთავსი ჩაიწერა მეხსიერებაში n–ის მისამართზე.

თუმცა, ინკრემენტის ოპერაციისთვის ასემბლერებს აქვთ ბრძანება inc:

inc n

ხოლო a \*(b + c)გამოსახულების იმპლემენტაცია შემდეგია:

load R1,b

load R2,c

add R1,R2 შევკრიბეთ b და c, შედეგი R1 რეგისტრშია

load R2,a

mult R1,R2 გავამრავლეთ a და b + c, შედეგი R1 რეგისტრშია

ამრიგად, მინიჭების შეტყობინების ზოგადი სახეა

< assignment statement > : : = < lvalue > <assign operator> < rvalue >

< assign operator > : : = : = | =

< lvalue > : : = < variable > | < expression >

< rvalue > : : = < expression >

სხვადასხვა დაპროგრამების ენებში მინიჭების ოპერატორს შეიძლება ჰქონდეს განსხვავებული სახე:

: = (Pascal, Ada) ; = (Fortran, C/C++, Java, C#).

შემდეგი Pascal –ის კოდი გვიჩვენებს სწორად ჩატარებულ მინიჭებებს

Var X: Real; Y: String; Z: Boolean;

Begin

Х : = 2 + 3.5 ; { X <– 5.5 }

Y : = 'Viena'; { Y <– 'Viena'}

Z : = 4 > 7; { Z <– False }

End.

ზოგ ენაში დაშვებულია მრავალჯერადი მინიჭება. შემდეგი

int v1 , v2, v3;

v1 = v2 = v3 = e;

C –ს კოდის შესრულება ხდება მარჯვნიდან მარცხვნივ.

ხოლო Ada –ს კოდი

V1,V2,V3: Integer := Е;

განიხილება როგორც შემოკლებული ჩანაწერი შემდეგი

V1: Integer := E;

V2: Integer := Е;

V3: Integer := Е;

საწყისი მინიჭებებისა, და არა როგორც მრავალჯერადი მინიჭება.

C - ს შტოს ენების სასარგებლო თვისებას წარმოადგენს მინიჭების და სხვა არითმეტიკული ოპერატორების კომბინაცია:

|  |  |  |
| --- | --- | --- |
| ოპერატორები | შეტყობინება | ეკვივალენტური შეტყობინება |
| += | x += 2; | x = x + 2; |
| –= | x -= 2; | x = x - 2; |
| \*= | x \*= 2; | x = x \* 2; |
| /= | x /= 2; | x = x / 2; |
| %= | x %= 2; | x = x % 2; |

**პროგრამის მართვის ინსტრუქციები**

მართვის შეტყობინებები (Control Statements) გამოიყენება პროგრამის ინსტრუქციების შესრულების რიგის შესაცვლელად. არსებობს კონტროლის შეტყობინებების ორი კლასი: 1) ამორჩევის შეტყობინებები (if და switch/case), რომლებიც ორი ან მეტი ალტერნატივიდან ირჩევენ ერთ შესასრულებელ ინსტრუქციათა თანმიმდევრობას; 2) განმეორების (loop) შეტყობინებები (for, while, ... ), რომლებიც მრავალჯერ იმეორებენ ინსტრუქციათა ჯგუფის შესრულებას.

**შეტყობინებები case და switch**

გამოიყენება იმისათვის, რომ შეარჩიოს გამოთვლისთვის რამდენიმე შესაძლო გზიდან ერთი.

Pascal -ში არსებობს ფორმით case of

var Symbol: Char;

case Symbol of

'A'..'Z', 'a'..'z' : Writeln('letter');

'0'..'9' : Writeln('digit');

'+', '-', '\*', '/' : Writeln('operator');

else Writeln('special symbol')

end;

Ada –ში მსგავსი ფრაგმენტი ასე ჩაიწერება:

С: Character;

case С is

when 'A'..'Z'|'a'..'z' => Put\_line("It’s a letter");

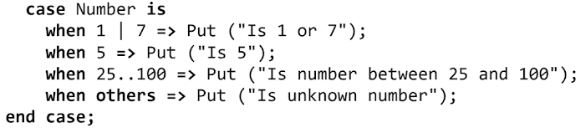
when '0'.. '9' => Put\_line("It’s a digit");

when '+'|'-'|' \*'|'/' => Put\_line("It’s an operator");

when others => Put\_line("I dont know");

end case;

კიდევ ერთი Ada –ს მაგალითი



C/C++ –ში ამორჩევის შეტყობინებას წარმოადგენს switch-ი. მისი ზოგადი სახეა:

switch( გამოსახულება )

{

caseკონსტანტა1:

შეტყობინება; . . .

break;

caseკონსტანტა2:

შეტყობინება; . . .

// სრულდება მომდევნო შეტყობინებაც

default:

შეტყობინება; . . .

break;

}

ქვემოთ მოცემულია switch-ის რამდენიმე მაგალითი, განხორციელებული C/С++ –ზე:

1.

int x = 7;

switch (x)

{

case 1: cout << "x = 1\n"; break;

case 3: cout << "x = 2\n"; break;

default: cout << "x != 1, x != 3\n"; break;

}

2.

int month\_number;

cout << "Enter month's number ";

cin >> month\_number;

switch (month\_number)

{

case 12: case 1:

case 2: cout << "Winter\n"; break;

case 3: case 4:

case 5: cout << "Spring\n"; break;

case 6: case 7:

case 8: cout << "Summer\n"; break;

default: cout << "Autumn\n";

}

3.

enum Traffic\_light\_color{ yellow, green, red };

Traffic\_light\_color translate(string );

int main()

{

cout << "What color of a traffic light is turned on? ";

string color;

cin >> color;

Traffic\_light\_color ob = translate(color);

switch (ob)

{

case yellow: cout << "Be ready !\n"; break;

case green: cout << "Go !\n"; break;

case red: cout << "Stop !\n";

}

}

Traffic\_light\_color translate(string s)

{

if (s == "yellow") return yellow;

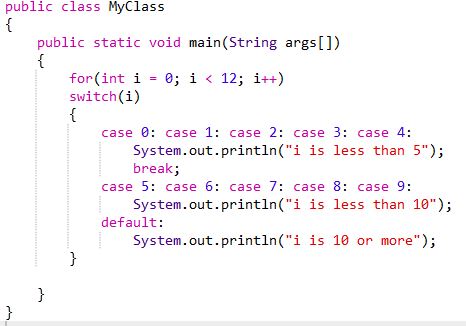
if (s == "green") return green;

if (s == "red") return red;

}

Java –ს მაგალითი:

// In a switch, break statements are optional.



C# –ის კოდი:

// In a switch, break statements are mandatory.

1.

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter language (C#, VB или C++)");

string myLanguage = Console.ReadLine();

sw1(myLanguage);

Console.ReadLine();

}

static void sw1(string s)

{

switch (s)

{

case "C#":

Console.WriteLine("You chose C#");

break;

case "VB":

Console.WriteLine("You chose Visual Basic");

break;

case "C++":

Console.WriteLine("You chose С++");

break;

default:

Console.WriteLine("I don't know that language");

break;

}

}

}

2. class Program

{

static void Main(string[] args)

{

L1:

Console.WriteLine("Enter month number");

string month = Console.ReadLine();

if (month == "exit") goto L2;

byte month\_number = Convert.ToByte(month);

switch (month\_number){

case 12:

case 1:

case 2: Console.WriteLine("Winter\n"); goto L1;

case 3:

case 4:

case 5: Console.WriteLine("Spring\n"); goto L1;

case 6:

case 7:

case 8: Console.WriteLine("Summer\n"); goto L1;

default: Console.WriteLine("Autumn\n"); goto L1;

}

L2: Console.ReadLine();

}

}

*# case in Ruby*

1.

score = 70

result = case score

when (0..40) === score then "Fail"

when 41..60 then "Pass"

when 61..80 then " Good"

when 81..100 then "Excellent"

else "Invalid Score"

end

puts result

2.

b = 9

case

when b < 3

puts "Little than 3"

when b == 3

puts "Equal to 3"

when (1..10) === b

puts "Something in closed range of [1..10]"

end

3.

salary = 1000001; season = "Summer";

happy = case

when salary > 1000000 && season == "Summer" then

puts("I’m happy")

when salary > 500000 && season == "Spring" then puts("That's not bad")

else puts("That's awful")

end

***იმპლემენტაცია***

switch/case კომპილირდება შედარების ტესტების თანმიმდევრობით შესრულებაში.

მაგალითად, მოცემული switch–ის

switch (expr) {

case value\_1: statement\_1;

break;

case value\_2: statement\_2;

break;

. . .

default\_statement: d\_statement;

}

რეალიზება შეგვიძლია აღვწეროთ შემდეგნაირად:

compute R1,expr გამოსახულების გამოთვლა

jump\_eq R1,#value\_1,L1

jump\_eq R1,#value\_2,L2

. . .

default\_statement default –ის ინსტრუქციები

jump End\_Case

L1: statement\_1 statement\_1–ის ინსტრუქციები

jump End­\_Case

L2: statement\_2 statement\_2–ის ინსტრუქციები

jump End\_Case

. . .

End\_Case

**ამორჩევის (პირობითი) შეტყობინებები**

if შეტყობინებას აქვს ორი ფორმა: შემოკლებული და სრული. შემოკლებული ფორმის ზოგადი სახეა:

if (პირობა)

შეტყობინება;

ხოლო სრული ფორმისა

if (პირობა)

შეტყობინება1;

else

შეტყობინება2;

if შეტყობინება არის switch/case შეტყობინებების კერძო შემთხვევა, სადაც პირობა ლოგიკური ტიპის გამოსახულიბაა. ვინაიდან ლოგიკურ ტიპს აქვს მხოლოდ ორი დასაშვები მნიშვნელობა, if აკეთებს ამორჩევას პროგრამის გაგრძელების ორ შესაძლო გზას შორის. მაგალითად,

if X > Y then

statement\_1;

else

statement\_2;

end if;

სწორი კოდის ფრაგმეტია Ada–ში.

გავრცელებული შეცდომაა if შეტყობინების გამოყენება ბულის ტიპის მნიშვნელობის შესაქმნელად

if X > Y then

Result = True;

else

Result = False;

end if;

დაპროგრამების კარგი სტილი გვკარნახობს ამ შემთხვევაში გამოვიყენოთ მარტივი მინიჭება

Result := X > Y;

if –ის პირობაში ყველაზე ხშირად გვხვდება შედაების გამოსახულებები, რომლებიც აბრუნებენ Тruе –ს ან False –ს. მაშინ როგორღა მართავს if –ი განშტოებას იმ ენებში, სადაც ბულის ტიპი განსაზღვრული არაა, მაგალითად, С –ში?

if (x > у)

statement\_1;

else

statement\_2;

აქ „მცდარი“ (False) გაიგივებულია მთელრიცხვა მნიშვნელობასთან ნული, ხოლო „ჭეშმარიტი“ (Тruе) – მნიშვნელობასთან არა ნული. აქედან, С/С++ –ში მართებულია if –ის პირობაში გამოვიყენოთ არითმეტიკული გამოსახულებები:

შემდეგი ფრაგმენტი

if( 2\*5 - 1.2 ) cout << "1111\n";

else cout << "2222\n";

დაბეჭდავს 1111 –ს, ხოლო

if( 2\*5 - 10 ) cout << "1111\n";

else cout << "2222\n";

გამოიტანს 2222 –ს.

საკმაოდ ხშირად გვხვდება ჩადგმული if-else-ები. ამ დროს მთავარია გვახსოვდეს, რომ else ეკუთვნის მის წინ მდგომ პირველივე if -ს.

ვთქვათ, პროგრამაში უკვე შეყვანილია ორი არანულოვანი ნამდვილი რიცხვი x და y, და ჩვენი ამოცანაა გავარკვიოთ და დავბეჭდოთ თუ სიბრტყის რომელ საკოორდინატო მეოთხედში არის მოთავსებული (x, y) წერტილი. ამ მიზნით შეგვიძლია გამოვიყენოთ შემდეგი კოდი:

if (x > 0)

if ( y > 0)

cout << "პირველი მეოთხედი\n";

else

cout << "მეოთხე მეოთხედი\n";

else

if ( y > 0)

cout << "მეორე მეოთხედი\n";

else

cout << "მესამე მეოთხედი\n";

სასურველია, არ მოვახდინოთ if-else-ების ძალიან ღრმა ჩადგმა - საკმარისია მაქსიმუმ სამი ან ოთხი დონე. ამის მიზეზი ის არის, რომ სხვაგვარად რთულდება კოდის ლოგიკის აღქმა. გარდა ამისა, ლოგიკური შეცდომების წყაროს წარმოადგენს ტექსტის წანაცვლება დონეების მიხედვით, რადგან ის მხოლოდ ორიენტირია: თუ თქვენ გამოტოვებთ else-ს, სინტაქსურად კოდი მაინც შეიძლება იყოს სწორი, თუმცა იმუშავებს არასწორად. მაგალითად, შემდეგ კოდში

if (x1 > у1)

if (x2 > у2)

statement\_1;

else

statement\_2;

გაკეთებული წანაცვლებებიდან ჩანს, რომ ავტორი ცდილობს შეასრულებინოს if-else შეტყობინებას statement\_2–ი, თუ x1 > у1 პირობა მცდარია. მაგრამ else ეკუთვნის მეორე და არა პირველ if-ს. სიტუაცია შეიცვლება, თუ ჩავწერთ

if (x1 > y1)

{

if (x2 > у2)

statement\_1;

}

else

statement\_2;

თუ არჩევანი ხორციელდება ერთი გამოსახულების საფუძველზე, შეგვიძლია გამოვიყენოთ switch გადამრთველი. თუ კი არჩევანი კეთდება გამოსახულებების თანმიმდევრობის საფუძველზე, საჭირო ხდება ჩადგმული if-else-ების თანმიმდევრობა:

if (х > у) {

...

} else if (x > z) {

...

} else if (y < z) {

...

} else {

...

}

ჩადგმული if-else-ები ბევრ ენაში (Ruby, Perl, Oracle) გავრცელდა if-else-if სტრუქტურამდე elsif მომსახურე სიტყვის შემოღებით. მაგალითად, როგორც [Ada](https://en.wikibooks.org/wiki/Ada_Programming/Libraries/Ada) -ს შემდეგ კოდში:

[type](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/type) Degrees [is](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/is) [new](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/new) Float [range](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/range) -273.15 .. Float'Last;

...

Temperature : Degrees;

...

[if](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/if) Temperature >= 40.0 [then](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/then)

Put\_Line ("It's extremely hot");

[elsif](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/elsif) Temperature >= 30.0 [then](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/then)

Put\_Line ("It's hot");

[elsif](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/elsif) Temperature >= 20.0 [then](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/then)

Put\_Line ("It's warm");

[elsif](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/elsif) Temperature >= 10.0 [then](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/then)

Put\_Line ("It's cool");

[elsif](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/elsif) Temperature >= 0.0 [then](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/then)

Put\_Line ("It's cold");

[else](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/else)

Put\_Line ("It's freezing");

[end](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/end) [if](https://en.wikibooks.org/wiki/Ada_Programming/Keywords/if);

***იმპლემენტაცია***

if შეტყობინების განხორციელება მარტივია:

compute R1,expression

jump eq R1,L1 False წარმოდგენილია 0 –ით

statement\_1

jump L2

L1: statement\_2

L2: