Спецификација на јазикот Inger

Акции во Inger

Акции

- Прости искази
- Сложени искази
- Повторливи искази
- Условни искази
- Искази со контрола на проток (тек)

- Секоја компјутерска програма се состои од инструкции-искази кои извршуваат некакви акции и дејствуваат врз податоците потребни за таа програма
- Првично, акциите може да се поделат на две поглавни категории: прости искази и сложени искази, но постојат и други.

Прости искази

- Исказ за доделување на вредност

<променлива>=<израз>

Левата страна е променлива на која и се доделува израз и се вика lvalue, додека пак десната страна е израз којшто е доделен на променливата и се нарекува rvalue. Знакот = е оператор за доделување вредности.

Примери

- 2 * 3 – 4 * 5 = (2 * 3) – (4 * 5) = -14
- 80 / 5 / 3 = (80 / 5) / 3 = 5
- 9.0 * 3 / 2 = (9.0 * 3) / 2 = 13.5

  Забелешка: При операцијата делење ако и двата операнди се од тип integer, тогаш и резултатот е исто така integer (заокружен), а ако барем еден од двата операнди е float, тогаш резултатот е float.

Сложени искази

- Група од ниеден или повеќе искази заграден со загради { и }.
- BNF формата за овој исказ:

  блок: { код } .

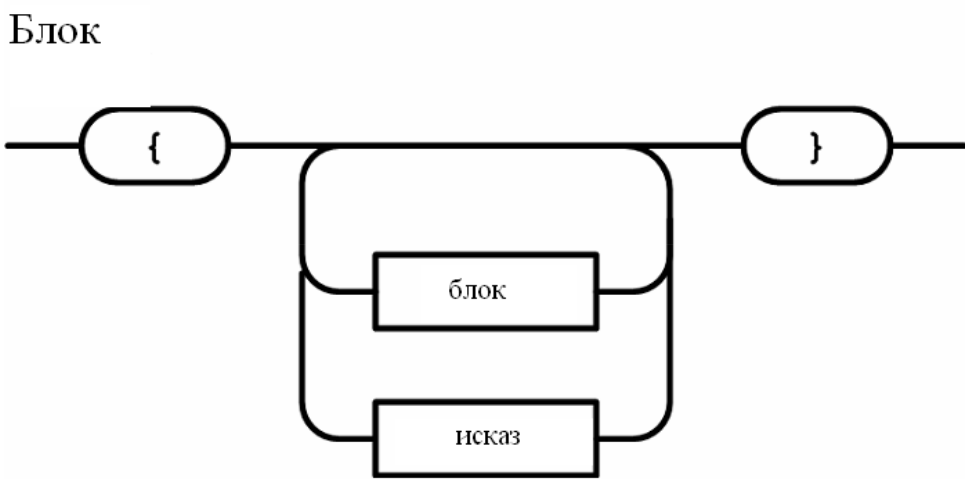
  блок: e.

  код: блок код.

  код: исказ код.

  Може да не содржи искази или повеќе искази и исто така може да содржи други блокови.

Синтаксен дијаграм за блок



Пример

```
module compound;
start f : void -> void
{
   {
        int a = 1;
   }
}
```

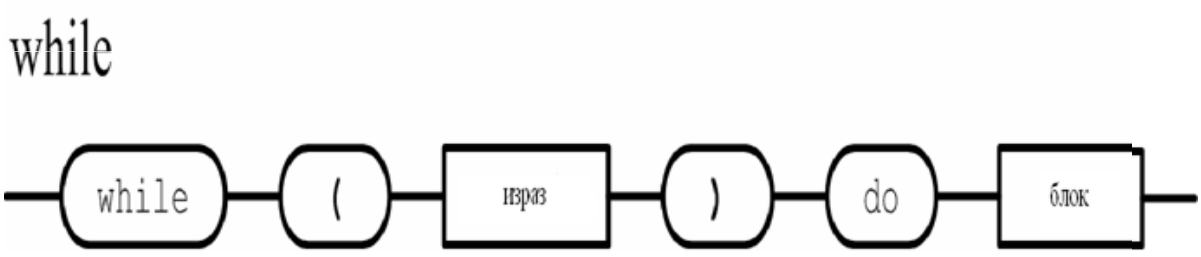
- Во овој пример функцијата f има свој блок (тоа е телото на функцијата), којшто содржи друг блок, којшто на крај содржи прост исказ – декларација.

Повторливи искази

- Повторливите искази се формираат со помош на сложените искази, кога истите ќе бидат вметнати во исказ којшто ќе се повторува, цо цел да биде извршен повеќе пати.
- За разлика од другите јазици, кои подржуваат повеќе типови на повторливи искази, Inger подржува само еден и тоа while исказот.

BNF за while исказот

- statement: **while** (expression) **do** block
- Синтаксен дијаграм за while



Пример

```
module while_demo;
# import "printint. ih"
start main: void -> void
{
    int n = 10;
    float h = 0;
    while( n > 0 ) do
    {

        h = h + 1 / n;
        n = n - 1;
    }
printint ( h );
}
```

- Изразот што се наоѓа измеѓу заградите треба да биде од тип bool, така што пред да се изврши сложениот исказ во блокот треба да се провери изразот дали е точен и ќе се извршува се додека изразот во заградите не стане false.
- Пожелно е изразот да биде попрост
- Постојат и два додатни контролни искази:
  - break (за прекин)
  - continue (за продолжување)

Условни искази: if и switch

- If исказ
- Се состои од израз од тип boolean и еден или повеќе сложени искази. Ако boolean изразот е точен, се извршува првиот исказ, ако не вториот.
  - BNF:

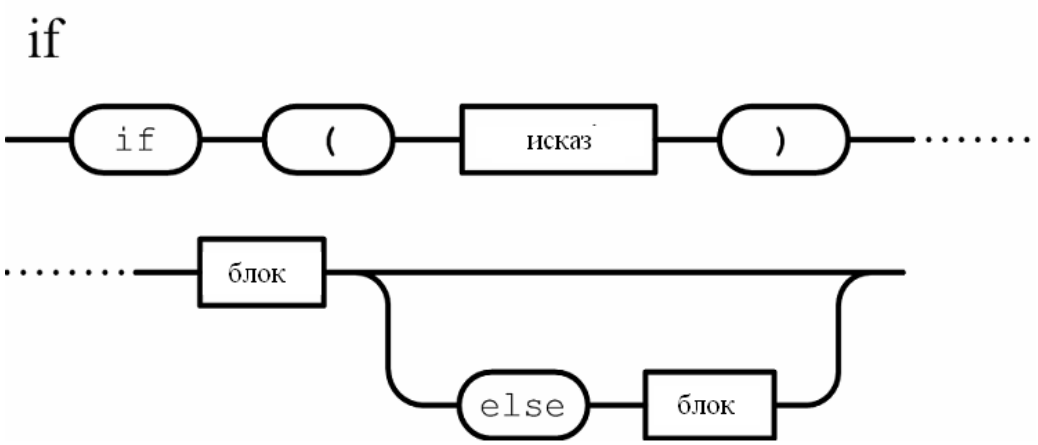
    исказ: **if** (израз) блок инакублок

    инакублок : е.

    инакублок : else блок.

Синтаксен дијаграм за if

- Else блокот покажува дека овој исказ може да содржи и втор сложен исказ или да не содржи



Условни искази: if и switch

- Case исказ
- BNF

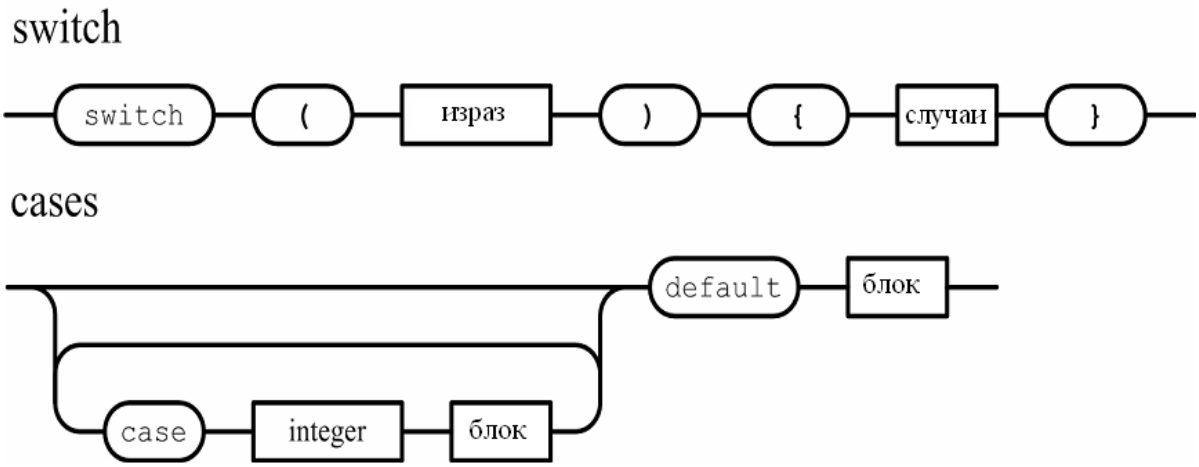
  исказ: **switch** (израз) {случаи **default** блок}.

  случаи : e.

  случаи : **case** <int literal > блок случаи.

- Се состои од израз и листа од алтернативни случаи, означени со броеви. Се извршува изразот, кој би требало да биде од тип integer, и доколку се поистоветува со некој од случаите тогаш се извршува тој, ако не тогаш се извршува default случајот (ако го има, а пожелно е).

Синтаксен дијаграм за switch



Пример

```
switch( a )
{
    case 0
    {
        printstr ( "Case 0\n" );
    }
    case 1
    {
        printstr ( "Case 1\n" );
    }
    case 2
    {
        printstr ( "Case 2\n" );
    }
    default ger
    {
        printfstr ( "Case >2\n" );
    }
}
```

Искази со контрола на проток (тек)

- Да го стопираат извршувањето на програмата
- Да го прелоцираат извршувањето на друга локација
- Да го продолжат извршувањето
- goto_considered_harmful – името на наредбата за контрола на текот на податоците
- Прави скок на предефинирано место од самиот програмер користејќи го клучниот збор **label**
- Треба да се избегнува користење на goto_considered_harmful

Пример

```
int n = 10;
label here ;
printstr ( n );
n = n - 1;
if ( n > 0 )
{
   goto_considered_harmful here;
}
```

Низи

- За разлика од простите податочни типови: integer, float, char, boolean и untyped, Inger подржува и манипулирање со низи.
- Низата содржи предетерминиран број на елементи, но сите од ист тип
- Бројот и големината на елементите е фиксен
- Низа – структура со случаен пристап
- Еднодимензионални и дводимензионални - матрици

Низи - еднодимензионални

- Пр: int a[5];

  - декларирана е низа од пет елементи од тип int
  - прв индекс е 0
  - пристап до елементи:
    - a[1] – втор елемент од низата a
    - a[4] – последниот елемент од низата

Низи

- Дводимензионални низи матрици
- Пр: int a[4][6];
- Се декларира матрица 4 x 6
- Пристап да елемент: a[2][2] е елементот од втората редица и втората колона.
- Низа од каракатери претставува string
- Пр: char a[20] = “hello, world!”;
- Првите 13 елементи од низата се иницијализирани со соодветните карактери, додека пак елементот a[13] е иницијализиран на 0, за да индицира дека наредните карактери се неиницијализирани.

Покажувачи

- Тие содржат адреси и служат за да покажуваат на други променливи
- Операторот & се користи за да покаже на адресата на било која променлива
- Операторот * се користи за да пристапи до променливата на дадената адреса.
- Пример1:

```
int a;
int *b = &a; // на пром. b и е доделена адресата на пром. a
*b = 2; // вредност 2 и е доделена на пром. На која покажува b (a)
printint ( a ); // на крај a ќе има вредност 2
```

Пример2

Покажувач се однесува на друг покажувач

```
int a;
int *b = &a;
int **c = &b;
**c = 2;
printint ( a ); /* 2 */
```

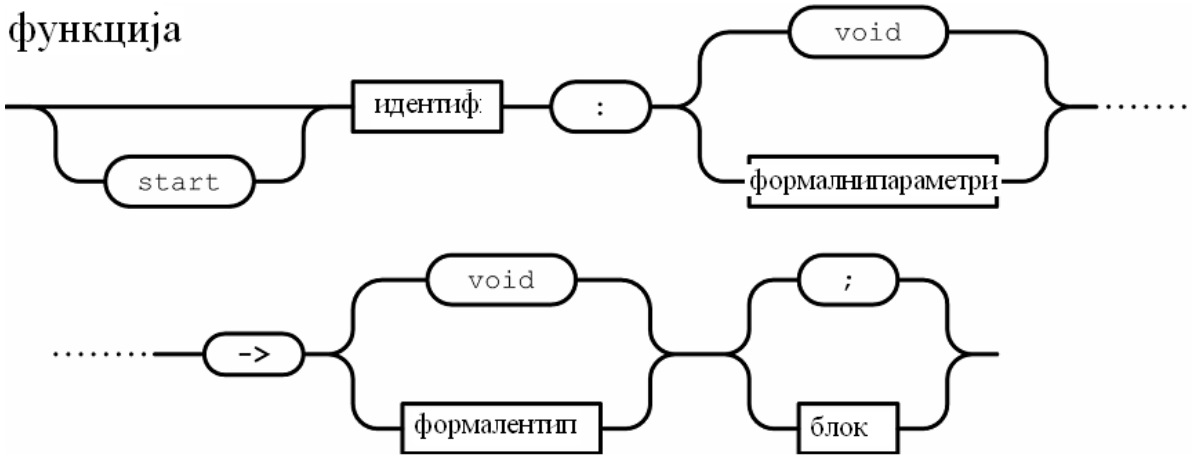
Друга употреба на покажувачите

- Статички променливи
  - Се наоѓаат на стекот
- Динамички променливи
  - Се наоѓаат во куп
  - Се креираат со користење на функциите на оперативниот систем за да за нив може да се лоцира меморија и нивната адреса се зачувува во покажувачот

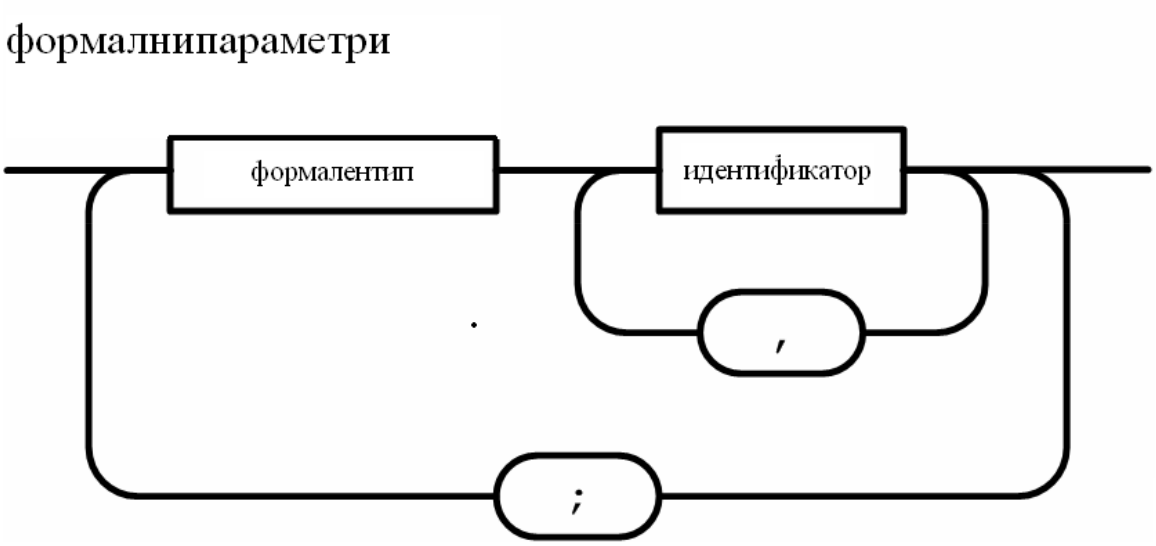
Функции

- Нивна намена е да се оддели дел од програмата и на тој дел да му се додели име или идентификатор
- Во секоја Inger програма постои барем една почетна функција која се означува со **start** функција.
- Секоја функција мора да биде декларирана пред да се користи

Синтаксен дијаграм за функција



Синтаксен дијаграм за блок за формалнипараметри



Примери за валидни имиња на функции

- f: void -> void

  (не зема аргументи и не враќа резултат)
- g: int a; bool b -> int

  (зема int и bool аргумент, а враќа int)
- h: char str[][] -> int *

  (зема дводимензионална низа од тип char како аргумент, а враќа покажувач од тип int)

Што има во телото на функцијата?

- Параметри, локални променливи, клучниот збор return за да вратат вредност кога ќе се повика функцијата
- Некои не содржат параметри воопшто-void
- Некои не содржат return вредност-void
- Нема ни return ни параметри- double void

Како се повикува функција?

- Со операторот ()
- f();

  повик на функција без параметри
- int result = g(3, false);

  повик на функција со два параметри, првиот од тип int, а вториот од тип boolean
- И ова е дозволено

  int result = g(g (3, false), false);

Повик со вредност и со референца

```
f : int a -> void
{
    a = 2;
}
```

```
f : int *a -> void
{
    *a = 2;
}
```

Повик со вреднсот

```
int i = 1;
f ( i );
printint ( i ); /* 1 */
```

```
int i = 1;
f(&i);
printint ( i ); /* 1 */
```

Модули

- Програмата може да се состои од повеќе модули, не само од еден.
- Еден е главен (**main**) модул, којшто содржи една и само една функција - start функцијата. Таа мора да биде од тип void, а може да биде повикана од други функции како и секоја друга
- Импортирање со extern и со #import

C и Inger имплементација на функцијата printint

- C имплементација на printint функцијата

```
void printint ( int x )
{
   printf ( "%d\n", x );
}
```

- Header file за printint функцијата во Inger

```
extern printint : int x -> void;
```

Библиотеки

- Што е разликата помеѓу Inger и останатите најкористени програмски јазици?
- Inger не користи готови функции, за определена операција!

  Пример: SIN, COS, READ, WRITE, …
- Дали се треба да се програмира???

Библиотеки

- Можност за избор:

  - Програмерот сам да си направи функции за определени операции (напредно);

  - Со искористување на библиотеки кои постојат во Inger, и од таму да се преземат функции

Библиотеки

- Содржина:

  - еден или повеќе модули, и секој од нив не содржи START функција (во спротивно јавува грешка при поврзувањето)

  -но, копајлерот не е тој што проверува за постоењето на повеќе START функции, тоа го врши поврзувачот (linker)

Библиотеки

- Со помош на поврзувачот (linker) се вклучуваат библиотеките во самата програма, аво кодот се користи:

  #import

Библиотеки

- Може да се вметнат и функции напишани во C

- Ваквите функции се вметнуваат со **extern** декларација на функција, при тоа се користи секаде каде што се вметнуваат вакви функции.

Библиотеки

- Пример:

```
module program;
#import “printint.ih”;
int a,b;
start main: void -> void
{
    a=b=1;
    printint(a+b);
}
```