Подалгоритми

Програмирањето е процес на подготовка на некој проблем (задача) за решавање со помош на компјутер. Тој процес се состои од: поставување на проблемот, дефинирање на постапката (алгоритамот) и пишување програма.

При поставување на проблемот потребно е истиот да се разбере, анализира и, по можност, да се разбие на потпроблеми. Потоа секој потпроблем може да се разгледува како посебна целина, за чие решавање може да се применуваат различни алгоритми и да се пишуваат посебни програми. Ваквите програми се викаат потпрограми. Со логичко поврзување на сите потпрограми се формира програма за решавање на поставениот проблем.

На пример, проблемот "Пресметување просечен успех на едно училиште" може да се разбие на следните потпроблеми: пресметување просечен успех на еден ученик, на едно одделение, на една година, на сите години.

Ако за секој од овие потпроблеми напишеме посебни програми - потпрограми и сите нив ги поврземе во една програма, така добиената програма ќе биде многу појасна, попрегледна и поразбирлива.

За секој подалгоритам може да се напише посебна програма која се нарекува потпрограма (анг. subprogram). Програмата напишана за некој алгоритам се нарекува главна програма (анг. main program). Во главната програма може да се повикуваат повеќе потпрограми.

Потпрограмите претставуваат независни програмски целини, кои имаат свои влезни податоци и даваат излезни резултати. Секоја потпрограма може да биде напишана од друг човек. Корисникот на потпрограмата треба да знае само да ја користи: да знае кои и какви влезни податоци се дозволени и кои резултати се добиваат.

Потпрограмите кои се користат за решавање на стандардни проблеми, се напишани најчесто од група програмери и усовршени се чуваат во посебни програмски библиотеки. Потпрограмите од библиотеките можат да се користат од секоја корисничка програма со повикување преку името на потпрограмата. Претходно, програмата треба да биде поврзана со програмската библиотека.

Со постоењето на програмските библиотеки се ослободуваат програмерите од пишување на исти програми за проблеми со кои често се среќаваат. На пример, во секоја програмска библиотека постојат потпрограми за: наоѓање корени на квадратна равенка, собирање на два полиноми, решавање систем линерни равенки итн.

Потпрограмите што најчесто се користат при програмирањето, наречени стандардни потпрограми, се ставени во посебни програмски модули кои се составен дел на преведувачот на соодветниот програмски јазик.

Постојат два вида потпрограми:

- функциски потпрограми или функции (анг. function), и
- процедурални потпрограми или процедури (анг. procedure)

Подалгоритми

Подзадачите се користат за добивање еден или повеќе меѓурезултати, од кои во понатамошните чекори, се добиваат резултатите на задачата. За добивање на меѓурезултатите не мора да се користат сите влезни податоци, туку само некои од нив или некои претходно добиени меѓурезултати.

За секоја подзадача може да се напише посебен алгоритам, кој ќе го наречеме подалгоритам. Подалгоритмите, исто како и алгоритмите, се именуваат. Влезните податоци во подзадачата и меѓурезултатите што се добиваат, се нарекуваат аргументи.

Влезните податоци се нарекуваат влезни аргументи, а меѓурезултатите се нарекуваат излезни аргументи. Аргументите се наведуваат во заграда по името на подалгоритмот. На пример, во подалгоритмот *Поголем*, влезните аргументи можеме да ги запишеме со broj1 и broj2, а излезниот аргумент со pogolem. Тогаш насловот на подалгоритмот Поголем може да се запише со *Поголем*(broj1, broj2, pogolem; На крајот од насловот на подалгоритмите се става знакот ; (точка и запирка).

Алгоритмите за посложени задачи можат да бидат многу долги и непрегледни. Затоа и програмите што ќе се напишат според тие алгоритми можат да бидат тешко разбирливи. Тоа е посебно важно кога ќе се јави потреба од некоја промена или надополнување во нив. За полесно снаоѓање во големите програми, денес тие се пишуваат со техника на програмирање позната како структурно програмирање.

При структурното програмирање се користат две техники на програмирање, и тоа:

- програмирање одгоре надолу (анг. top-down programming) и
- модуларно програмирање (анг. modular programming).

Програмирањето одгоре надолу се врши со разделување (расчленување) на задачата на помали и поедноствни задачи, кои ќе ги наречеме подзадачи. Ако е потребно, и тие подзадачи понатаму се разделуваат на уште поедноставни, додека не се добијат задачи што лесно се програмираат.

Секоја подзадача, од така расчленетата задача, може да се разгледува како посебна целина, независно од другите.

На пример, во алгоритамот на задачата за наоѓање на најголемиот од три дадени броја, двапати се бара одредување на поголемиот од два броја.

```
алгоритам Najgolem;
                почеток
                        читај a,b,c;
                        ако a>b
                                тогаш
                                        pogolem <- a
                                иначу
                                        pogolem <- b;
                        крај_ако{a>b}
                        p <- pogolem;
                        ако p>c
                                тогаш
                                        pogolem <- p
                                иначу
                                        pogolem <- c;
                        крај_ако{p>c}
                        n <- pogolem;
                        печати n;
                крај{Najgolem}
```

Тоа може да се издвои како посебен алгоритам, односно подалгоритам, со кој може да се извршат двете споредувања: *a* и *b* во првото, и *p* и *c* во второто споредување на кои било два броја.

Постојат два вида подалгоритми, и тоа:

- функциски и
- процедурални.

Функциски подалгоритми

Функциските подалгоритми, наречени и функции (анг. function), го добиле името бидејќи имаат само еден излезен параметар, исто како и функциите. Излезниот параметар се нарекува *излезен формален параметар*, додека влезните параметри, кои можат да бидат и повеќе, се нарекуваат *влезни формални параметри* .

Излезниот формален параметар од функциските подалгоритми е **самото име на функцијата.** Влезните формални параметри се стават во заграда по името.

За да одлучиме дали за една подзадача може да се напише функциски подалгоритам, потребно е да се одговори на следното прашање:

0. Дали подалгоритмот за подзадачата има една излезна вредност?

Ако одговорот на ова прашање е да, тогаш за подзачата може да се напише функциски подалгоритам. При пишување на функциски подалгоритам за зададена подзадача, потребно е да се одговорат и прашањата:

1. Од кои аргументи директно зависи резултатот на подзадачата?

2. Од кој тип е излезниот резултат на подзадачата?

Одговорот на првото прашање ќе ни укаже на тоа: кои, колку и каков тип влезни формални аргументи ќе има во листата на формални аргументи; додека, пак, одговорот на второто прашање ќе ни го одреди типот на излезниот резултат од функцискиот подалгоритам.

На пример, подалгоритамот за наоѓање на поголемиот од два броја може да се запише како функциски подалгоритам на следниов начин:

```
подалгоритам Pogolem (broj1, broj2);
        почеток
                ако   broj1>broj2
                        тогаш
                                Pogolem <- broj1
                        инаку
                                Pogolem <- broj2;
                kraj_ako (broj1>broj2)
крај (Pogolem)
```

Еве и еден пример на задача каде решението не може да се добие со функциски подалгоритам: Да се подредат три броја по големина.

Одговорот на основното прашање (ред. бр. 0) е НЕ, затоа што во задачата се бара влезните податоци (трите броја) да бидат и излезни резултати, пак три броја, само подредени по големина. Затоа решението не можеме да го добиеме со функциски подалгоритам.

За функцијата да врати резултат, името на функцискиот подалгоритам мора да се јави на левата страна барем во една наредба за доделување во самиот подалгоритам, бидејќи преку името се пренесува резултатот од подалгоритамот во главниот алгоритам.

Функциските подалгоритми се повикуваат со чекор за доделување. На пример со чекорот:

```
p <- Pogolem(a,b);
```

се повикува подалгоритамот за наоѓање на поголемиот од броевите *a* и *b*. Притоа, влезните формални параметри *broj1* и *broj2* се заменуваат со вистинските *a* и *b* соодветно. Преку името на

подалгоритамот *Pogolem*, со чекорот за доделување *p ß Pogolem(a,b);*, во променлива *p* се пренесува резултатот од споредувањето на *a* и *b*.

Алгоритамот за наоѓање на најголемиот од три дадени броја со користење на функцискиот подалгоритам *Pogolem*, ќе биде

```
алгоритам Najgolem;
почеток
     читај a,b,c;
     p <- Pogolem(a,b);
     n <- Pogolem(p,c);
     печати n;
крај(Najgolem)
```

Во програмите кои користат потпрограми променливите можат да се поделат на: глобални променливи и локални променливи. Променливите кои се декларираат во главната програма се нарекуваат глобални променливи, а променливите кои се декларираат во потпрограмите се нарекуваат локални променливи. Самото нивно име ни зборува и за областа во која со нив може да се оперира, т.е. кога тие се достапни (видливи).

Променливите што се најавуваат на почетокот на програмата се нарекуваат *глобални променливи*. За да се избегне зависност на потпрограмите од главната програма, Паскал дозволува декларирање константи, променливи, типови, па и други функции во декларативниот дел на функцијата и тие имаат значење само во функцијата, т.е. делуваат **локално**.

Пример:

```
Function Stepen (x : Real; N : Integer) : Real;
Var
       Pom : Real;
      i : 1..MaxInt;
Begin
      Pom:=1.0;
      For i:=1 to abs(n) do
           Pom:=Pom*x;
           If n >= 0 then Stepen:=Pom
           Else Stepen:=1.0/Pom;
End;
```

***Pom , i* се локални променливи.**

Накратко, вреднување на функциски повик се одвива на следниот начин:

1. Се креира мемориска локација за функцискиот повик;

2. Вистинските параметри на функцискиот повик формираат парови со формалните параметри на функцијата, од лево на десно. Нивниот број мора да е еднаков;

3. Се креира мемориска локација за секој формален параметар на функцијата, кон кои е придружен соодветниот вистински параметар. Типовите на двата вида параметри мора да се исти;

4. Се креира мемориска локација за секоја локална променлива на функцијата;

5. Се изведуваат наредбите во телото на функцијата;

6. На крај, сите мемориски локации креирани за функцискиот резултат, формалните параметри и локалните променливи се ослободуваат. Вредноста на функцискиот резултат се пренесува како вредност на функцискиот повик.

Процедурални подалгоритми

Функциските подалгоритми се корисни, но ограничени поради тоа што даваат само еден резултат. Некогаш е потребен подалгоритам кој ќе произведе неколку резултати. Такви подалгоритми се *процедуралните подалгоритми* - *процедури*. **Главна разлика** во однос на **функциските подалгоритми** е тоа што **процедурите не враќаат вредност.**

Процедурите имаат свое име, кое е единствено и по кое се разликуваат и се повикуваат.

Процедурите можат да бидат без параметри и со параметри. Процедурите без параметри претставуваат, всушност, еден блок од програмата, означен со име и крај.

Кај процедурите со параметри, после името, во мали загради се наведуваат параметрите. За секој параметар се наведува и типот. Овие параметри се нарекуваат *формални параметри*, бидејќи тие не се дефинирани додека не се заменат со *вистинските параметри*, кои се задаваат при повикување на процедурата.

До повикувањето, процедурата е непозната и нема никакво дејство.

Во главната програма една процедура може да се повикува повеќе пати.

Процедурите се повикуваат во главната програма преку името, односно со наредба за повикување на процедура.

```
ИмеНаПроцедурата;
```

Кај процедурите со параметри, по името, во мали загради се наведуваат параметрите. Вистинските параметри се задаваат при повикување на процедурата со наредба за повикување на процедура со синтакса:

```
ИмеНаПроцедура(листа на вистински параметри);
```

При повикувањето, формалните параметри се заменуваат соодветно со вистинските параметри. Бројот, редоследот и типот на вистинските параметри, мора да биде ист со формалните параметри.

Да го земеме истиот пример како кај функциите.

```
Procedure Pogolem(broj1,broj2 : integer; var  pog:integer);
Begin
      If  broj1>broj2
            Then
                  pog := broj1
             Else
                   pog := broj2;
End;
Program Najgolem;
Var a,b,c,p,n : integer;
Begin
       ReadLn(a,b,c);
       Pogolem(a,b,p);
        Pogolem(p,c,n);
        WriteLn(n);
End.
```

Анализа: Променливите *a,b* и *p* се вистински параметри. При повикувањето на подалгоритамот се врши замена на формалните параметри со вистинските. Формалниот параметар *broj1* ќе се замени со вистинскиот параметар *a, broj2* со *b* и *pog* со *p*. По извршувањето на подалгоритмот *Pogolem*, променливата p ќе ја содржи вредноста на поголемиот од броевите *a* и *b*, која ја добил преку формалниот параметар *pog*. Значи, *pog* е формален параметар преку кој се пренесува резултатот од подалгоритамот во главниот алгоритам. Слично се случува и ако подалгоритамот го повикаме со: *Pogolem(p,c,n);* Формалните параметри *broj1,broj2* и *pog* се заменуваат со вистинските *p,c* и *n* соодветно.

-Вредносни и променливи параметри

Вредносни параметри (ВП) се формалните параметри кои **внесуваат** вредности во процедурата. Со повикување на процедурата, на секој вредносен параметар му се доделува соодветниот вистински параметар. По ова доделување на вредностите, не постои повеќе заемнодејство меѓу формалните и вистинските параметри. Ако во процедурата се назначи нова вредност на формалните параметри, тоа нема да има ефект над соодветниот вистински параметар. Затоа се дефинираат *променливи*

параметри (ПП) кои можат да ја менуваат својата вредност и **изнесуваат** вредности (резлтати) од процедурата во програмата. Во Паскал се означуваат се зборот VAR пред формалниот параметар.

Пример.

```
Procedure Pogolem(broj1,broj2 : integer; var  pog:integer);
```

Повик на вредносен параметар уште се нарекува *повик по вредност (call by value)*, додека на променлив параметар *повик по адреса(call by adress)*.

Процедуралните алгоритми ќе ги објасниме и со следниов пример, за наоѓање на најмалиот елемент во низата [ai]n и неговиот реден број. Бидејќи се бараат два излезни резултати, не може да се напише функциски подалгоритам, туку ќе напишиме процедурален:

```
подалгоритам Најмал_Елемент(n,a,najmal,redbr);
почеток
      najmal <- a1;
      redbr <- 1;
      за k <- 2 зголемувај до n
            ако ak < najmal
                  тогаш
                        najmal <- ak;
                        redbr <- k;
            крај_ако {ak < najmal}
       крај_за{k}
крај{Најмал_Елемент}
```

Процедуралниот подалгоритам се повикува само со неговото име, а во заграда се стават вистинските аргументи. На пример, за наоѓање на најмалиот елемент во низата [ci]m и неговиот реден број, подалгоритмот *Најмал_Елемент* треба да се повика со чекорот

```
Најмал_Елемент (m,c,min,rb);
```

Променливите m, c, min и br се вистински аргументи. При повикувањето на подалгоритмот се врши замена на формалните аргументи и тоа: n со m, a со c, najmal со min и redbr со rb.

По извршувањето на подалгоритмот, променливата min ќе ја содржи вредноста на најмалиот елемент на низата, која ја добила преку формалниот аргумент najmal, додека променливата rb ќе го содржи

редниот број на најмалиот елемент во низата, кој го добила преку формалниот аргумент redbr. Значи, najmal и redbr се формални аргументи преку кои се пренесуваат резултатите од подалгоритмот во главниот алгоритам. Затоа тие се нарекуваат излезни формални аргументи. Аргументите, пак, n и a во кои се пренесуваат вистинските аргументи од главниот алгоритам во подалгоритмот, се нарекуваат влезни формални аргументи.

При повикување на подалгоритмот, бројот, редоследот и типот на вистинските аргументи мораат да бидат исти со бројот, со редоследот и со типот на формалните аргументи. Алгоритмот за наоѓање на најмалиот елемент во низата [ci]m и неговиот реден број, со повикување на процедуралниот подалгоритам *Најмал_Елемент* е следен:

```
Алгоритам НајмалИРедБрој;
почеток
        читај m;
        за i <- 1 зголемувај до m
              читај ci ;
        крај_за (i)
        Најмал_Елемент(m, c, min, rb);
       печати 'Најмал е ',rb,'-от елемент со вредност ',min;
крај (НајмалИРедБрој)
```

Накратко, ефектот на процедурален повик е:

1. Се создаваат парови од лево на десно на вистинските со формалните параметри од процедурата. Бројот на параметрите е ист;

2. Се креира мемориска локација за секој вредносен параметар кон кој е придружен соодветниот вистински параметар. Типовите се исти;

3. Секој променлив параметар се поврзува со соодветниот вистински параметар и тој е негов претставник. Вистинскиот параметар мора да е променлива од ист тип како и променливите параметри;

4. Се креира мемориска локација за секоја локална променлива чија иницијална вредност е недефинирана.

5. Се извршуваат наредбите од процедурата. Сега вредносните параметри се одесуваат како и локалните променливи. Секоја промена, пак, на променливите параметри се пренесува на вистинскиот параметар кого го претставува;

6. На крај, сите мемориски локации за формалните параметри и локалните променливи се ослободуваат.

Глобални и локални променливи

Подрачје на декларации

Декларација претставува воведување име и тип на променливите. *Дефиниција* е декларација кога на името на променливата и се дава и значење. *Подрачјето на важење* е множеството на програмски линии во кои е видлива (може да се користи) некоја променлива. Постојат:

- глобална декларација и
- локална декларација

Глобалните променливи имаат подрачје на важење во целата програма. Тие се декларирани надвор од функциите.

Локалните променливи имаат подрачје на важење само во телото на функцијата во која се декларирани или само во блокот во кој се декларирани (локалните променливи имаат *блок подрачје на важење*, помеѓу две загради). *Скриена* променлива е онаа која не е видлива (не може да се користи) во некој блок.(Пример: глобални и локални променливи со исто име)

Следи еден програмски код во кој во коментари се напишани карактеристиките на променливите и нивните вредности:

```
int m=10;           // m e globalna promenliva
void fun(int m, int& n);
main()
{
     int n=20;           // n i p se
     int p=30;           // lokalni za main()
     if (n<p)
     {                   //priv ne moze da se koristi nadvor
          int priv;
          priv=p;
          p=m;                // m e globalnata promenliva
     }                   // od ova { i ova } zagrada
     cout<<m;            // se pecati globalnata m=10
     int m=44;           // tuka i vo slednata linija vazi
                       //lokalnata m=44,
                     // globalnata m=10 e skriena
     fun(m,p);           // fun se povikuva za m=44 i p=30
}
void fun(int m, int& n) // ovie m i n se lokalni promenlivi za fun
{                    // i vazat samo vo ovaa funkcija
     int p;              // ova p ne e ona od main()
     p=m;
     n=m+p;
}
```

Правила за подрачјето на важење на променливите:

1. Променливата не може да се користи надвор од подрачјето на важење,
2. Глобалните променливи можат да се користат во целата програма,
3. Променливите декларирани во еден блок можат да се користат само во него (освен како формални аргументи)
4. Променлива декларирана во една функција не може да се користи во друга,
5. Променливата може да биде скриена во некој дел од нејзиното подрачје на важење,

6. Не може две различни променливи со исто име да имаат исто подрачје на важење. (Не може да се декларираат две променливи со исто име во различни блокови во иста функција.)
7. Може две функции со исто име да имаат исто подрачје на важење, само ако имаат различни листи на аргументи.

Живот на променливите

Живот на променливите е времето од креирањето до исчезнувањето.

Локалните променливи живеат од моментот на извршување на дефиницијата, до крајот на извршување на блокот (функцијата) во која се дефинирани. Тие се автоматски by default. Автоматските променливи се декларираат со зборот auto. На пример:

```
auto float o1;   // isto so float o1;
int o2;          // isto so auto int o2;
```

Глобалните променливи се *статички* и тие живеат цело време додека се извршува програмата. И локалните променливи можат да се декларираат со зборот static. На пример:

```
static int broj=n;
static double iznos;
```

Глобалните променливи живеат од моментот на нивното декларирање, до завршување на извршувањето на програмата. Иницијализација: само еднаш. (ако не се иницијализирани автоматски се иницијализираат на 0 - сите битови 0).

Локалните статички променливи живеат од едно до друго повикување на функцијата. При новото повикување на функцијата тие ја имаат состојбата (вредноста) добиена во претходното повикување.