ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Dərs №18

Preprosessor və onun istifadəsi

Mündəricat

1. Preprosessor

Preprosessor — proqramın ilkin mətni kompilyasiya edilməmişdən əvvəl onun üzərində bəzi işlər (hərdən cox əhəmiyyətli) görən proqramdır. İngilis dilindən hərfi olaraq, ilkin emal edən kimi tərcümə edilir.

Preprosessorlar kompilyatorlar üçün giriş mətni yaradır və aşağıdakı funksiyaları yerinə yetirə bilirlər:

- makroverilənlərin emalı;
- faylların işə salınması;
- ilkin «rasional» emal;
- dilin genişlənməsi.

Məsələn, tez-tez proqramlarda «heç nə deməyən» ədədlərdən istifadə etmək lazım gəlir. Bu, riyazi sabitlər və ya proqramda istifadə olunan massivlərin ölçüləri və s. ola bilər. Qəbul olunmuşdur ki, bu cür sabitlərin çoxluğu proqramın başa düşülməsini çətinləşdirir yaxşı olmayan proqramlaşdırma əlaməti sayılır. Proqramçılar arasında bu cür sabitlər istehza ilə sehrli ədədlər adını almışlar. Onların proqramda çoxluq təşkil etməməsi üçün, proqramlaşdırma dili imkan verir ki, sabitə ad verilsin və sonra hər yerdə sabitin əvəzinə o, işlədilsin.

C dilində belə bir imkanı preprosessor yaradır. Məsələn, belə bir təyindən sonra

```
#define P1 3.14159
#define E  2.71284
```

Preprosessor proqramda olan bütün P1 və E adlarını uyğun ədədi sabitlərə çevirir. Əgər siz aşkar etsəniz ki, natural loqarifm əsaslarının təqribi qiymətlərini düzgün yazmamısnız, onda bütün proqram üzrə axtarış etmək lazım deyil, kifayətdir ki, sabitin təyin olunduğu sətirdə düzəliş edəsiniz :

```
#define E 2.71828
```

C dilinin preprosessoru təkcə sabitləri deyil, həm də bütövlükdə proqram konstruksiyalarını yenidən təyin etməyə imkan verir. Məsələn, əvvəlcə belə bir təyin vermək:

```
#define forever for(;;)
```

sonra isə hər yerdə aşağıdakı şəkildə sonsuz dövrlər yazmaq olar:

```
forever { <тело цикла> }
```

Əgər fiqurlu mötərizələrdən istifadə etmək istəmirsinizsə, onda belə yazın:

```
#define begin {
#define end }
```

və sonra isə operator mötərizələri əvəzinə ***Pascal*** dilində olduğu kimi ***begin*** və ***end*** operatorlarından istifadə edin.

Makrotəyinlər (makroslar) adlanan bu cür təyinlər parametrlərə malik ola bilərlər (bunun nəticəsində isə daha da qüvvətlənə bilərlər), lakin bu haqda sonra danişacağıq.

Preprosessorun daha bir vacib xüsusiyyəti - ilkin mətnə digər faylların içindəkiləri daxil etməkdir. Bu imkan əsasən, bütün fayllar üçün eyni olan təyinlər ilə proqramları təhciz etmək üçün istifadə olunur. Məsələn, tez-tez C dilində olan proqramların əvvəlində belə bir proprosessor konstruksiyası rast gəlinir:

```
#include <iostream>
```

Proqramın başlanğıc mətni proprosessor tərəfindən emal olunarkən, bu instruksiyanın yerinə, makrotəyinlərdən və giriş-çıxış axınının işi üçün vacib olan verilənlərdən ibarət olan ***iostream*** faylının içindəkilər yazılır.

Preprosessorun operatoru (direktivi) — bu, ilkin mətnin # simvolu ilə başlanan və ardınca operatorun adı (***define, pragma, include, if***) və operandlar gələn bir sətirdir. Preprosessorun operatorlarına proqramın istənilən yerində rast gəlmək olar və onların işi bütün başlanğıc fayl üzrə yayılır.

#define operatorunun köməyi ilə sabitlərin təyin edilməsi

#define operatoru cox vaxt simvol sabitlərinin təyin edilməsi üçün istifadə olunur. O, başlanğıc faylın istənilən yerində gələ bilər, bu vaxt həmin yerdən başlayaraq faylın axırına kimi öz təyinatını yerinə yetirəcək.

Qeyd: *Simvol sabiti təyin olunduqdan sonra axırda (#define operatorunun sonunda) nöqtğ vergül qoyulmur!*

```
# define min 1
# define max 100
```

Proqramın mətnində 1 və 100 sabitlərinin əvəzinə uyğun olaraq ***min*** и ***max*** yazmaq olar.

```
#include <iostream>
using namespace std;
#define NAME "Vasya
Pupkin." void main ()
{
    cout << " My name is " << NAME;
}
İşin nəticəsi:
My name is Vasya Pupkin.
```

Qeyd: *Sətirdaxili mətn, simvol sabitləri və şərhlər dəyişdirilmirlər, belə ki, sətirlər və simvol sabitləri*

C dilinin bölünməyən leksemləridir. Beləliklə, aşağıdakı makrotəyindən sonra

```
#define YES 1
```

bu operatorda

```
cout << "YES";
```

heç bir makroyerdəyişmə aparılmayacaq.

Mətndə edilən dəyişiklikləri aşağıdakı əmrin köməyi ilə ləğv etmək olar:

```
#undef <ad>
```

Belə bir direktiv yerinə yetirildikdən sonra preprosessor üçün ad təyin olunmamış kimi qalır və onu yenidən təyin etmək lazım gəlir. Məsələn, aşağıdakı direktivlər xəbərdarlıq tələb etmirlər:

```
#define M 16
#undef M
#define M 'C'
#undef M

#define M "C"
```

#undef direktivindən müxtəlif vaxtlarda və ya müxtəlif proqramçılar tərəfindən yazılmış ayrı-ayrı «mətn parçalarından» yığılmış həcmli proqramların emalı zamanı istifadə etmək daha rahatdır. Bu halda müxtəlif obyektlərin eyni işarələnməsinə rast gəlmək olur. İlkin faylı dəyişdirməmək üçün daxil edilən mətni uyğun ***#define — #undef*** direktivləri ilə əhatə etmək olar və bununla da mümkün

səhvləri aradan qaldırmaq olar. Məsələn:

```
.    .   .    .   .
   A =10; // Əsas mətn.
.    .   .    .   .
#define A  X
.    .   .    .   .
   A =5; //Daxil edilmiş mətn
.    .   .    .   .
#undef A
.    .   .    .   .
   B =A; // Əsas mətn.
.   .   .   .   .
```

Daxil edilmiş mətndə ***A=5***; mənimsəmə operatorunun olmasına baxmayaraq, proqram yerinə yetirilərkən ***B*** 10 qiymətini alacaq.

Əgər ***leksem sətri*** cox uzun olarsa, onu proqram mətninin sonrakı sətrində davam etdirmək olar. Bunun üçün davam edilən sətrin sonunda «\» simvolu yerləşdirilir. Preprosessorun emal mərhələlərinin birində bu simvol özündən sonra gələn sətrin sonu simvolu ilə birlikdə proqramdan silinəcək. Məsələn:

```
#define STROKA "\n Multum, non multa -
    \ mnogoe, no nemnogo!"
   .  .  .  .  .
   cout << STROKA;
   .   .  .  .  .

Ekrana çıxarılacaq:

   Multum, non multa - mnogoe, no nemnogo!
```

Yadınıza salmaq istəyirik ki, doqquzuncu dərsdə yerləşdirilmə haqqında öyrənəndə də biz #define direktivinin köməyilə makroslar yaradırdıq. Bu dərsə qayıtmağı və keçilmiş materialı təkrarlamağı məsləhət görürük.

3. Şərti kompilyasiya

Şəri kompilyasiyanın direktivləri müəyyən şərtlərin yerinə yetirilməsindən asılı olaraq proqramın kodunu yaratmağa imkan verir. C dilində şərti kompilyasiya əmrlər yığımı ilə təmin edilir. Bu əmrlər kompilyasiyanı deyil, preprosessor emalını idarə edirlər:

```
#if <sabitli ifadə>
#ifdef <identifikator>
#ifndef <identifikator>
#else#endif#elif
```

Birinci üç əmr şərtləri yoxlayır, sonrakı iki əmr isə yoxlanılan şərtin yerinə yetirilmə diapazonunu təyin etməyə imkan verir. Axırıncı əmr şərtlər sırasının yoxlanılmasını təşkil olunması üçün istifadə edilir. Şərti kompilyasiya direktivlərinin tətbiq olunmasının ümumi strukturu belədir:

```
#if/#ifdef/#ifndef <sabitli ifadə _
                   Və ya  identifikator>
          <mətn_1>
#else // qeyri məcburi direktiv
          <mətn_2>
#endif
```

■■ ***#else <mətn_2>*** konstruksiyası məcburi deyil .

■■***Mətn_1*** kompiyasiya olunan mətnə yalnız yoxlanılan şərtin doğru halında daxil edilir.

■■Əgər şərt yanlışdırsa, onda ***#else*** direktivinin varlığı halında, ***mətn_2*** kompilyasiyaya ötürülür.

■■Əgər ***#else*** direktivi mövcud deyilsə, onda yanlış şərt halında, ***#if*** –dən ***#endif*** –dək olan bütün mətn buraxılır.

#if əmrinin formaları arasindakı fərq aşağıdakından ibarətdir.

1. Sadalanan ***#if*** direktivinin birincisində tam qiymətli sabit ifadənin qiyməti yoxlanılır. Əgər o, sıfırdan fərqlidirsə, onda yoxlanılan ifadə doğru hesab olunur. Məsələn, direktivlərin yerinə yetirilməsi nəticəsində:

```
#if 5+12
        <mətin_1>
#endif
```

mətn_1 həmişə kompilyasiya edilən proqrama daxil ediləcək.

2. ***#ifdef*** direktivində yoxlanılır ki, hal-hazırda ondan sonra yerləşdirilən identifikator ***#define*** əmrinin köməyi ilə təyin olunmuşdurmu**.** Əgər identifikator təyin olunmuşdursa, onda kompilyator mətn_1-dən istifadə edir.
3. ***#ifndef*** direktivində bunun tərsi yoxlanılır — doğru şərt kimi identifikatorun qeyri-müəyyənliyi qəbul olunur, yəni, bu elə haldır ki, idetifikator ***#define*** əmrində istifadə olunmamışdır və ya onun təyini ***#undef*** əmri ilə ləğv olunmuşdur.

Preprosessor proqramın başlanğıc mətnini emal edərkən multi-budaqlanmanın təşkil olunması üçün aşağıdakı direktiv daxil olunur.

```
#elif <sabit_ifadə>
```

Bu direktiv #else#if konstruksiyasının ixtisarıdır. Direktiv tətbiq edilərkən başlanğıc mətnin strukturu belə olar:

```
#if <sabitli ifadə_1>
          <mətn_1>
     #elif < sabitli ifadə_2>
          < mətn _2>
     #elif < sabitli ifadə_3>
          < mətn _3>
     .  .  .  .
     #else
          < mətn _N>
     #endif
```

- Preprosessor əvvəlcə ***#if*** direktivində şərti yoxlayır, əgər o, yanlışdırsa (0-a bərabərdir) — ***sabitli ifadə_2***-ni hesablayır, yox əgər 0-a bərabərdirsə— ***sabitli ifadə_3*** hesablanır və s.
- Əgər bütün ifadələr yanlışdırlarsa, onda kompilyasiya olunan mətnə ***#else*** halı üçün mətn daxil olunur.
- Əks halda, yəni heç olmasa bir doğru ifadənin varlığı (***#if*** və ya ***#elif-də***) halında, bu direktivdən bilavasitə sonra gələn mətn emal olunmağa başlayır, qalan direktivlərə isə baxılmır.

■■Beləliklə, preprosessor həmişə mətnin şərti kompilyasiyanın əmrlərinin seçdiyi yalniz bir hissəsini emal edir.

İndi isə bir neçə nümunəyə baxaq.

Nümunə 1. Şərti daxiletmənin sadə direktivi.

```
#ifdef ArrFlg
      int Arr[30];
   #endif
```

Əgər direktivin şərhi zamanı ArrFlg makrotəyini verilmişdirsə, onda göstərilmiş yazı aşağıdakı ifadənin yaranmasına gətirir.

```
  int Arr[30];
```

Əks halda heç bir ifadə yaranmayacaq.

Nümunə 2

```
#include   <iostream>
using  namespace  std;
#define ArrFlg 1
void main ()
{
   #ifdef ArrFlg
       int Arr[30];
   #else
       cout << "Array is not defined!";
   #endif
}
```

Nümunə 3. Şərti daxiletmənin alternativ direktivi

```
#if a+b==5
      cout << 5;
   #else
      cout << 13;
   #endif
```

Əgər a+b==5 ifadəsi 0-dan fərqli bir ədədi təmsil edirsə, onda cout << 5; əmri, əks halda isə cout << 13; əmri yaranacaq.

Nümunə 4. Şərti daxiletmənin düzəltmə direktivi

```
#include <iostream>
using namespace std;
//++++++++++++++++++++++
#define Alfa 5
//++++++++++++++++++++++
#if Alfa*5>20
    void main ()
        //++++++++++++++++++++++
        #if Alfa==4
                int Arr[2];
        #elif Alfa==3
                char Arr[2];
        #else
                {
        #endif
        //++++++++++++++++++++++
        #if 0
                cout<<"One";
        #else
                cout<<"Two";
        #endif
//++++++++++++++++++++++
```

```
#else
    cout<<"Test";
#endif
//++++++++++++++++++++++
}
```

Bu yazının şərhi aşağıdakına gətirib çıxarır.

```
void main ()
{
    cout<<"Kaja";
}
```

4.Preprosessorun digər direktivləri

Bizə məlum olan direktivlərdən başqa bir neçə əlavə direktiv də mövcuddur.Onlardan bəzilərinə baxaq:

1. Sətirləri nömrələmək üçün :

```
#line <константа>
```

direktivindən istifadə etmək olar. O, kompilyatora göstərir ki, mətnin növbəti aşağı sətri tam onluq sabitlə təyin olunan nömrəyə malikdir. Bu əmr nəinki sərtin nömrəsini, həm də faylın adını təyin edə bilər:

```
#line <sabit> "<faylın adı>"
```

2. Aşağıdakı direktiv

```
#error <leksem ardıcıllığı>
```

diaqnostik xəbərin leksem ardıcıllığı şəklində verilməsinə gətirir. ***#error*** direktivinin şərti preprosessor əmrləri ilə birgə tətbiqi təbiidir. Məsələn, hər hansı ***NAME*** preprosessor dəyişənini təyin edərək,

```
#define NAME 5
```

sonradan onun qiymətini yoxlamaq və NAME dəyişəninin başqa qiymət aldığı təqdirdə məlumat vermək olar:

```
#if (NAME!= 5)
#error NAME bərabər olmalıdır 5!.
```

Məlumat bu şəkildə olmalıdır:

```
fatal: <faylın adı> <sətrin nömrəsi>
#error directive: NAME bərabər olmalıdır 5!
```

3. Bu əmr

```
#pragma <leksem_ardıcıllığı>
```

kompilyatorun konkret tətbiq edilməsindən asılı olan fəaliyyətini təyin edir və kompilyatora müxtəlif göstərişlər verməsinə imkan yaradır.

4. C dilində # və ## operatorlarının istifadəsi mövcuddur. Bu operatorları #define direktivi ilə birgə istifadə olunur.

■■ # operatoru özündən əvvəl gələn arqumenti dırnaq arasında yazılmış sətirə çevirir.

```
#include <iostream>
using namespace std;
# define mkstr(s)
#s void main()
{
    cout<<mkstr(I love C);
    // kompilyator üçün cout<<"I love C";
}
```

■■ ## operatoru iki leksemi (konkatenasiya) birləşdirmək üçün istifadə olunur.

```
#include <iostream>
using namespace std;
#define concat(a,b)
a##b void main()
{
    int xy=10;
    cout<<concat(x,y);
    //kompilyator üçün cout<<xy;
}
```

5. Layihənin bir neçə fayl üzrə bölünməsi

Bildiyiniz kimi, faylı mətnə daxil etmək üçün #include əmri istifadə olunur. Gəlin onunla yaxından tanış olaq. Bu əmr preprosessorun direktivi olub, iki yazılış formaya malikdir:

```
#include <faylın_adı> // ad künc dırnaq içərisində.
#include "faylın_adı" // ad dırnaq içərisində
```

Əgər faylın_adı — künc dırnaq içərisində yazılıbsa, onda preprosessor faylı standart sistem kataloqunda axtarır. Əgər faylın_adı dırnaq içərisində yazılıbsa, onda əvvəlvə preprosessor istifadəçinin cari kataloqunu nəzərdən keçirir məhz bundan sonra standart sistem kataloquna müraciət edir.

C dili ilə işə başlayanda, biz proqramlarda giriş-çıxış vasitələrindən istifadəsinin zərruri olması ilə qarşılaşdıq. Bunun üçün direktivi proqramın mətninin əvvəlinə yerləşdirdik:

```
#include <iostream>
```

Bu dierktivi yerinə yetirərkən, preprosessor proqrama giriş-çıxış kitabxanası ilə əlaqə vasitələrini daxil edir. ***iostream*** faylının axtarışı standart sistem kataloqunda həyata keçirilir.

Başlıq fayllar böyük proqramların modullu emalı üçün kifayət qədər səmərəli vasitədir. Həmçinin, C proqramlaşdırma təcrübəsində adi haldır ki, proqramda bir neçə funksiya istifadə olunursa, onda bu funksiyaların mətnlərini ayrıca faylda saxlamaq daha rahatdır. Proqramı hazırlayarkən istifadəçi istifadə etdiyi funksiyaların mətnlərini ***#include*** əmrinin köməyi ilə proqrama daxil edir.

Nümunə üçün sətirlərin emalı məsələsinə baxaq. Burada mətnləri ayrıca faylda yerləşən sətirlərin emalı funksiyalarından istifadə edirik.

Proqram nümunəsi

Sözləri bir-birindən boşluq ilə ayrılan, nöqtə ilə qurtaran cümləni klaviatura vasitəsi ilə daxil edin. Cümlənin hər bir sözünü əks istiqamətdə (sözü çevirmək) yazmaq və alınmış cümləni yazmaq. Sadəlik üçün daxil etdiyimiz cümlənin uzunluğunu 80 simvol ilə məhdudlaşdıraq. Bu halda verilmiş məsələnin həlli proqramı belə olar:

Əsas fayl:

```
#include <iostream>
using namespace std;
// Sərbəst yazılmış, sətirlərin birləşdirici
   funksiyasını və sətiri çevirən funksiyanı özündə
   saxlayan fayl
#include "mystring.h"

void main()
{
    char slovo[81], sp[81], c = ' ', *ptr = slovo;
```

```
    sp[0] = '\0'; // Massivin yeni cümlə üçün
                  təmizlənməsi.
    cout << "Enter string with point of the
    end:\n"; do
    {
       cin >> slovo ; //Giriş axınından bir söz oxunur
       invert(slovo); // Sözü çevirmək
       c = slovo[0];
       // Axırıncı sözün əvvəlindən nöqtəni götürmək
       if (c == '.')
           ptr = &slovo[1];

        if (sp[0] !='\0')
            conc(sp," \0"); // sözdən əvvəl boşluq
                            // Cümləyə sözün əlavə
       conc(sp,ptr);        olunması
                           // Oxunma dövrünün sonu.
     }while (c != '.');    // Cümlə sonunun nöqtəsi.
     conc(sp,".\0");       // Cümlə sonunun nöqtəsi.
     cout << "\n" <<sp;
}
```

Başlıq faylı: mystring.h:

```
void invert (char *e)
{
   char s;
   for (int m=0;e[m]!='\0';m++); for
   (int i=0,j=m-1;i < j;i++,j--)
   {
           s = e[i];
           e[i] = e[j];
           e[j] = s;
   }
}
```

```
void conc (char *c1, char *c2)
{
    for (int m=0;c1[m]!='\0';m++);
    // m – birinci sətrin uzunluğu.
    for (int i=0; c2[i]!='\0';i++)
        c1[m+i]=c2[i];
    c1[m+i] = '\0';
}
```

Koda şərh.

■■Proqramda slovo simvol massivinə giriş axınından (klaviaturadan) növbəti söz daxil olur.

■■sp — qurulan cümlədir, onun axırına həmişə nöqtə əlavə olunur.

■■char c dəyişəni— hər bir çevrilən sözün birinci simvolu.

■■Cümlənin axırıncı sözü üçün bu simvol nöqtəyə bərabərdir.

■■Bu sözün yeni cümləyə əlavə olunması zamanı ptr göstəricisinin dəyişdirilməsinin köməyi ilə nöqtə atılır.

■■ Proqramda #include direktivləri, proqramda giriş/çıxış vasitələri, invert() sətir dəyişən və conc() sətir konkatenasiyası funksiyalarının mətnləri istifadə olunmuşdur.

■■Diqqət yetirin ki, conc() funksiyasının parametrlərinin birincisi olan massivin uzunluğu nəticə sətrinin yerləşə bilməsi üçün kifayət qədər böyük olmalıdır.

■■Preprosessor bütün funksiyaların mətnlərini mystring.h faylından proqrama daxil edir və vahid bir tam kimi kompilyasiyaya ötürür.

Qeyd: *Yeri gəlmişkən, layihəyə başlıq fayl əlavə etmək üçün, əsas faylın əlavəsi zamanı görülən bütün əməlləri aparmaq zərruridir, amma faylın tipini seçərkən Header File (.h) şablonu üzərində dayanmaq lazımdır.*

6. İmtahan tapşırıqları

1. Klaviaturadan daxil olan mətni süzgəcdən keçirən proqram qurmalı. Proqram mətni oxumalı və onu ekrana verməlidir, bu vaxt verilmiş simvol yığımının boşluqlarla əvəz olunmasından istifadə olunmalıdır. Proqram filtrləmə üçün simvollar yığımının aşağıdakı variantlarını təklif etməlidir:
   - Latın əlifbası simvolları
   - Kiril əlifbası simvolları
   - Punktuasiya simvolları
   - Rəqəmlər

   Filtrlər ardıcıllıqla qoyula bilərlər. Artıq mövcud olan filtrin təkrar qoyulması zamanı verilmiş filtr götürülməlidir.
2. İstifadəçinin kompüterə qarşı «Dəniz döyüşü» oyununun proqramını yazmaq. İstifadəçi əvəzinə gəmilərin düzülüşünün avtomatik (kompüter gəmilərin düzülüşünü təsadüfü olaraq yerləşdirir) və əl ilə olan bütün mümkün yollarını nəzərdən keçirmək. Əgər atışma vaxtı kompüter məntiq ilə addım atarsa (yəni, "təsadufi atəş" olmamalıdır), onda tapşırığın dəyəri kifayət qədər artır.
3. Həqiqi ədədlər, mötərizələr və +, –, *, /, ^ (dərəcəyə yüksəltmə) əməllərindən ibarət olan riyazi ifadənin hesablanması üçün əlavə yaratmaq

Hesablamalar mötərizələr və istifadə olunan əməllərin üstünlük dərəcəsi nəzərə alınmaqla aparılmalıdır. Mümkün olan xətaların düzəlişi təmin etmək və bu haqda istifadəçini məlumatlandırmaq.