Procedūrinio programavimo pagrindai

Preprocesorius

lekt. Irmantas Radavičius
irmantas.radavicius@mif.vu.lt
Informatikos institutas, MIF, VU

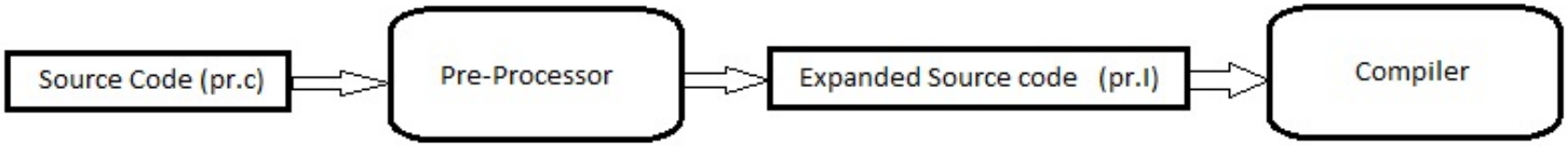
Turinys

Preprocesorius

Preprocesoriaus direktyvos

Preprocesorius

Preprocesorius iškviečiamas prieš kompiliuojant kodą.



Preprocesoriaus paskirtis – teksto apdorojimas.

- standartiniai (visada vykdomi) veiksmai
- preprocesoriaus direktyvos

Preprocesoriaus išvesties peržiūra:

gcc –E pr.c –o pr.i

Preprocesoriaus veiksmai

- simbolių koduočių derinimas
- pradinis apdorojimas

  (trigrafai, eilutės, komentarai)
- skaidymas leksemomis
- preprocesoriaus operacijų vykdymas

  (direktyvos, makrosai)

- kompiliavimas…

Komentarai

Visi komentarai pakeičiami tarpais.

main.c

```
1 int main (){
2     int a;/*komentaras*/int b;
3     int c; // komentaras
4 }
```

main.i

```
1 int main (){
2  int a; int b;
3  int c;
4 }
```

Pastaba: apdorota su opcija -P

Kodo eilutės

Ištrinamos “backslash-newline” sekos.

main.c

```
1  #defi\
2  ne VAL\
3  UE 1\
4
5  int main (){
6      int a; int b; \
7      int c;
8  }
```

main.i

```
1  #define __STDC_HOSTED__ 1
2  #define VALUE 1
3
4
5
6  int main (){
7   int a; int b; int c;
8
9  }
```

Pastaba: apdorota su opcijomis -P -dD -undef

Trigrafai

Trigrafai leidžia kitaip užrašyti tam tikrus simbolius.

Apdorojami pradinėje preprocesoriaus stadijoje.

main.c

```
1 int main()??<
2     int x??(1??);
3     return 0;
4 ??>
```

main.i

```
1 int main(){
2  int x[1];
3  return 0;
4 }
```

Trigraph	Equivalent
??=	#
??/	\
??'	^
??(	[
??)	]
??!	|
??<	{
??>	}
??-	~

Pastaba: apdorota su opcijomis -P -trigraphs

Problema:

```
// Will the next line be executed??????????????/
a++;
```

Teksto skaidymas leksemomis

Pagrindinės leksemų grupės:

identifikatoriai, skaičiai, tekstinės eilutės, skirtukai, kitos.

Įprastai (daugiaprasmiškumui išvengti) leksemos atskiriamos “whitespace” simboliais.

“Godi” (angl. greedy) strategija:

a+++++b traktuojamas kaip a ++ ++ + b (o ne a ++ + ++ b).

Digrafai

Digrafai apdorojami teksto skaidymo leksemomis metu.

main.c

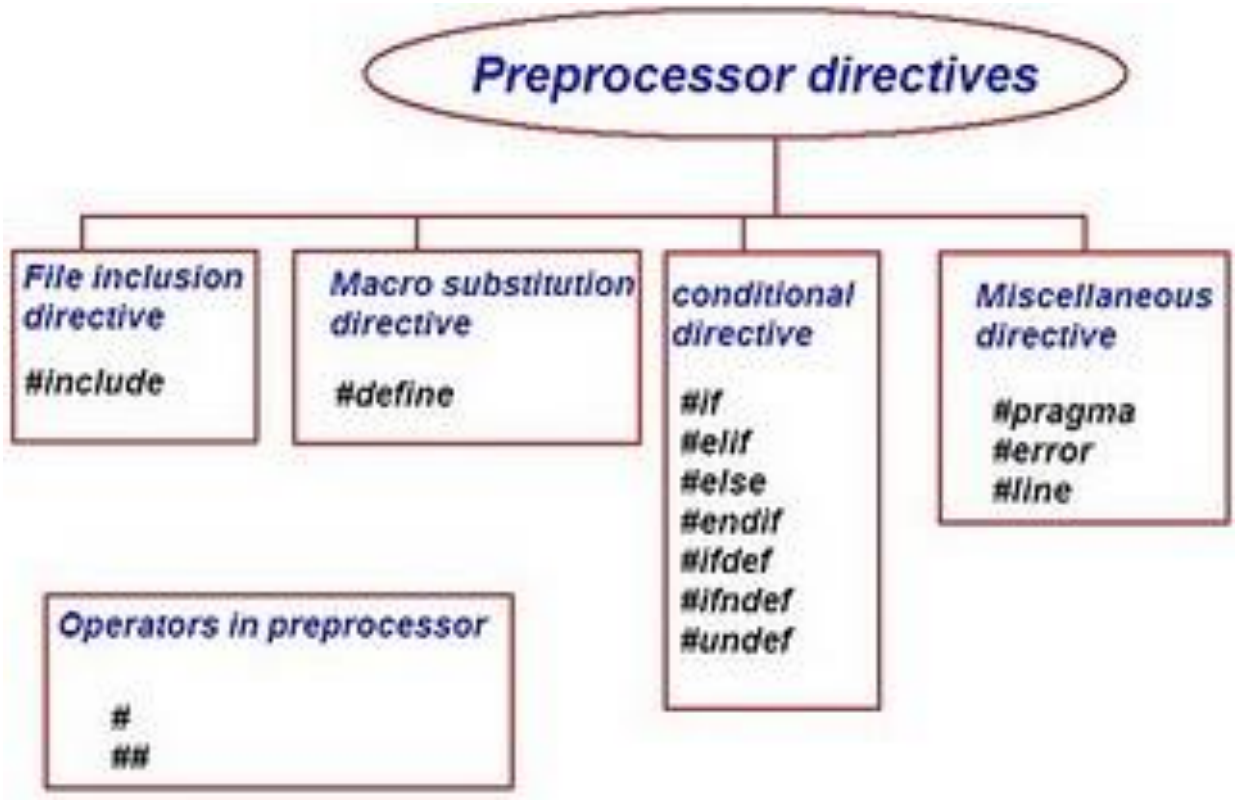
```
1 int main()<%
2  int x<:1:>;
3  return 0;
4 %>
```

Digraph	Equivalent
<:	[
:>	]
<%	{
%>	}
%:	#

- ✓ užrašas trumpesnis
- ✓ nėra potencialių problemų kaip kad su trigrafais
- – koduojama mažiau simbolių

Preprocesoriaus direktyvos

- pradedamos simboliu #
- prieš direktyvą eilutėje tegali būti “whitespace” ir/ar komentarai



Failo įtraukimo direktyva #include

Paskirtis – įtraukti vieno failo turinį į kitą failą.

Įgalina multifailines programas.

#include <failoVardas>

Failo ieškoma standartiniuose kataloguose.

#include “failoVardas”

Failo ieškoma einamajame (nebūtinai) kataloge.

Jei nepavyko rasti – ieškoma standartiniuose kataloguose.

Pastaba: apdorota su opcija -P

main.c | test

```c
#include "./test"

int main(){
    f();
    return 0;
}
```

main.c | test

```c
int f(){
    return 1;
}
```

main.i

```c
int f(){
 return 1;
}

int main(){
 f();
 return 0;
}
```

Makrosų apibrėžimo direktyva #define

Paskirtis – atlikti pakeitimus tekste.

Apibrėžimai atšaukiami su direktyva #undef.

main.c

```
1  #define MEM_SIZE 100
2  int main(){
3      int x, y;
4      printf("%d", x);
5      #define x y
6      printf("%d", x);
7      #undef x
8      printf("%d", x);
9      void *data = malloc(MEM_SIZE);
10 }
```

main.i

```
1  int main(){
2   int x, y;
3   printf("%d", x);
4
5   printf("%d", y);
6
7   printf("%d", x);
8   void *data = malloc(100);
9  }
```

Tipinis panaudojimas – konstantų apibrėžimui.

Įprastai vardai rašomi didžiosiomis raidėmis.

Pastaba: apdorota su opcija -P

Kai kurie standartiniai makrosai

Išplečiami apibrėžti pagal nutylėjimą (angl. predefined) makrosai.

main.c

```
1 int main(){
2     printf("Line number: %d\n", __LINE__);
3     printf("of file %s\n", __FILE__);
4     printf("Compilation started: %s\n", __DATE__);
5     printf("at time %s\n", __TIME__);
6 }
```

main.i

```
1 int main(){
2  printf("Line number: %d\n", 2);
3  printf("of file %s\n", "main.c");
4  printf("Compilation started: %s\n", "Apr 16 2012");
5  printf("at time %s\n", "00:39:58");
6 }
```

Pastaba: apdorota su opcija -P

Makrosai su parametrais

Apibrėžimas

```
#define FOO(x) - 1 / (x)
#define BAR (x) - 1 / (x)
```

Alternatyva funkcijoms

```
#define min(X, Y)  ((X) < (Y) ? (X) : (Y))
#define ABS(my_val) ((my_val) < 0) ? -(my_val) : (my_val)
```

- ✓ nereikia resursų funkcijų iškvietimui
- – atsiranda daug pasikartojančio kodo
- – potencialus klaidų šaltinis

Makrosų “pavojai” (1)

Operatorių prioritetai

```
#define cube( x ) x*x*x
#define double( x ) x+x
```

```
x = 3;
y = cube( x + 1 );
z = 5 * double( x );
```

```
x + 1*x + 1*x + 1
```

```
5 * x+x
```

- atskirti kiekvieną argumentą
- atskirti patį makrosą

```
#define cube( x ) ( ( x ) * ( x ) * ( x ) )
#define double( x ) ( ( x ) + ( x ) )
```

Makrosų “pavojai” (2)

Pašaliniai efektai

```
#define cube( x ) ( ( x ) * ( x ) * ( x ) )
#define double( x ) ( ( x ) + ( x ) )
```

```
x = 3;
y = double( ++x );
```

```
y = ( ( ++x ) + ( ++x ) );
```

- vengti pakartotinio argumentų įvertinimo
- nenaudoti reiškinių su pašaliniais efektais

```
#define double( x ) ( 2 * ( x ) )
```

Makrosų “pavojai” (...)

Kiti panaudojimo scenarijai...

```
#define ptrace( sts, str ) \
        if ( sts ) printf( "%s\n", str )
```

```
if ( x < 0 ) ptrace( traceon, "Negative input" );
else         ptrace( traceon, "OK input" );
```

```
if ( x < 0 )
   if ( traceon ) printf( "%s\n", "Negative input" );
   else
     if ( traceon ) printf( "%s\n", "OK input" );
```

```
if ( x < 0 ) {
   ptrace( traceon, "Negative input" );
}
else {
   ptrace( traceon, "OK input" );
}
```

Inline funkcijos (C99)

Inline funkcijos apibrėžimas

```
inline int max(int a, int b) {
  return a > b ? a : b;
}
```

- ✓ nereikia resursų funkcijų iškvietimui
- – atsiranda daug pasikartojančio kodo
- ✓ galioja funkcijoms taikomi reikalavimai (tipų kontrolė, etc)

Sąlyginis kompiliavimas

Leidžia įtraukti ar ignoruoti programos dalis kompiliavimo metu.

```
#if expression
controlled text
#endif /* expression */
```

```
#if BUFSIZE == 1020
  printf ("Large buffers!\n");
#endif /* BUFSIZE is large */
```

Reiškinyje galimi:

- skaitinės ir simbolinės konstantos
- aritmetiniai operatoriai
- makrosai (išskleidžiami) ir identifikatoriai (traktuojami kaip 0)

```
#if 1
/* This block will be included */
#endif
#if 0
/* This block will not be included */
#endif
```

Sąlyginis kompiliavimas

Sudėtingesnės direktyvos

```
#if X == 1
...
#else /* X != 1 */
#if X == 2
...
#else /* X != 2 */
...
#endif /* X != 2 */
#endif /* X != 1 */
```

```
#if X == 1
...
#elif X == 2
...
#else /* X != 2 and X != 1*/
...
#endif /* X != 2 and X != 1*/
```

Makrosų apibrėžtumo patikrinimas

`#ifdef` *name*

is equivalent to `#if defined (name)'.

`#ifndef` *name*

is equivalent to `#if ! defined (name)'.

```
#define DEBUG
...
#ifdef DEBUG
 something here
#endif
```

Kitos direktyvos

#error	preprocesorius skelbia klaidą
#line	keičia eilučių numeraciją
#pragma	priklauso nuo realizacijos

main.c

```
1 #line 200
2 #error "Something went wrong!"
3 int main(){
4     return 0;
5 }
```

Line	Message
200:2	#error "Something went wrong!"
	[Build Error] [../Test/main.o] Error 1

Preprocesoriaus operatoriai # ir

Makrosai nekeičiami eilučių viduje!

Operatorius # paverčia kodo fragmentą eilutine konstanta.

```
#define mkstr(s) # s
```

Operatorius ## apjungia dvi leksemas.

```
#define concat(a, b) a ## b
```

```
#include <stdio.h>
#define paster( n ) printf_s( "token" #n " = %d", token##n )
int token9 = 9;

int main()
{
   paster(9);
}
```

Output

```
token9 = 9
```

Preprocesoriaus išvestis

- rezultatas – apdorotas tekstas
- nebėra preprocesoriaus direktyvų
- įterpiamos eilučių žymės (angl. linemarkers)

  # eilutė failas požymiai

  požymiai:

  1 – failo pradžia

  2 – grįžimas į failą

  …
- opcijomis galima keisti teksto turinį

Pastaba: apdorota be opcijos -P

main.c | test

```
1 #include "./test"
2
3 int main(){
4     f();
5     return 0;
6 }
```

main.i

```
1  # 1 "main.c"
2  # 1 "<built-in>"
3  # 1 "<command line>"
4  # 1 "main.c"
5  # 1 "./test" 1
6  int f(){
7   return 1;
8  }
9  # 2 "main.c" 2
10
11 int main(){
12  f();
13  return 0;
14 }
```

Kai kurios GCC opcijos

-P	negeneruoti eilučių žymių
-trigraphs	konvertuoti trigrafus
-D name=definition	apibrėžti makrosą (#define analogas)
-U name	atšaukti apibrėžimą
-I dir	pridėti direktoriją įtraukiamų failų paieškai
-dM	sugeneruoti tik apibrėžtų makrosų sąrašą

...

#define END_OF_THE_LECTURE