

ЛЕКЦІЯ 14. Директиви препроцесору. Макроси

1. Препроцесор мови С
2. Директиви
3. Макроси

Препроцесор – це текстовий процесор, керуючий текстом файлу вихідного коду під час першого етапу трансляції. Препроцесор не аналізує вихідний текст, але він розбиває його на маркери виявлення викликів макросів. Хоча компілятор зазвичай викликає препроцесор при першому проході, препроцесор також можна викликати окремо для обробки тексту без його компіляції.

Препроцесор краще розглядати як окрему програму, яка виконується перед компіляцією. При запуску програми препроцесор переглядає код зверху вниз, файл за файлом, у пошуку директив.

Директиви – це спеціальні команди, які починаються із символу # і НЕ закінчуються крапкою з комою. Є кілька типів директив.

Операції

- визначення операнду як символьного рядку

- з'єднання лексем

defined – представлення `#ifdef` у формі виразу

Директива `#include`

Її вже було застосовано у роботі.

Коли ви підключаєте файл за допомогою директиви `#include`, препроцесор копіює вміст файлу в поточний файл відразу після рядка з `#include`. Це дуже корисно при використанні певних даних (наприклад, попередніх об'яв функцій) відразу в декількох місцях.

Директива `#include` має дві форми:

`#include <filename>`,

- повідомляє препроцесору шукати файл у системних шляхах (у місцях зберігання системних бібліотек мови C++). Найчастіше ця форма використовується при підключенні заголовних файлів із Стандартної бібліотеки C++.

`#include "filename"`

- повідомляє препроцесор, що потрібно шукати файл у поточній директорії проекту. Якщо його там не виявиться, то препроцесор почне перевіряти системні шляхи та будь-які інші, що вказано у налаштуваннях вашого IDE.

Директива `#define`

Директиву `#define` можна використовувати для створення макросів.

Макрос – це правило, яке визначає конвертацію ідентифікатора у зазначені дані.

Загальна форма

`#define ім'я_макросу послідовність_символів`

Послідовність символів інколи ще називають рядком заміщення. Коли препроцесор знаходить в тексті програми ім'я_макросу, він замінює його на послідовність_символів.

(Прийнято угоду позначати макроси великими літерами)

ПРОГРАМУВАННЯ

Є два основних типи макросів: макроси-функції та макроси-об'єкти. Макроси-функції поводяться як функції і використовуються з тією ж метою. Їх використання інколи вважається небезпечним, і майже все, що вони можуть зробити, можна здійснити за допомогою простої (лінійної) функції.

Макроси-об'єкти можна визначити одним із наступних двох способів:

`#define` ідентифікатор

Або: `#define` ідентифікатор текст_заміни

Верхнє визначення не має жодного текст_заміни, тоді як нижнє — має. Оскільки це директиви препроцесора, то жодна з форм не закінчується крапкою з комою.

ПРОГРАМУВАННЯ

У мові C досить поширеною практикою є оголошення глобальних констант з допомогою директиви `#define`:

```
#define PI 3.14159
```

Потім, на етапі препроцесінгу, всі використання `PI` будуть замінені вказаним значенням.

```
double area = 2*PI*r*r;
```

Після препроцесінгу, який по суті є підстановкою, цей вираз перетвориться на:

```
double area = 2*3.14159*r*r;
```

Про макроси важливо розуміти, що область видимості у них така сама, як у функцій у мові Cі.

Однак, на відміну від функцій, оголошення макросу можна скасувати:

```
#undef PI
```

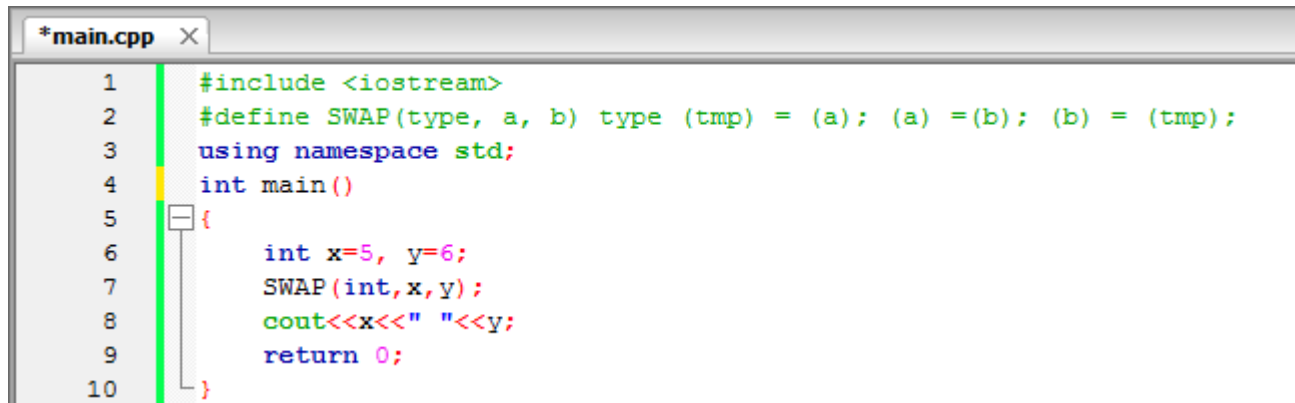
Після цього рядка звертатися до `PI` буде вже неможливо.

Макроси з параметрами

Параметри в макросах працюють приблизно так, як і аргументи функції. Простий приклад - макрос, який визначає більший із переданих йому параметрів:

```
#define MAX(a, b) ((a) >= (b) ? (a) : (b))
```

Макрос може складатися не лише з одного виразу. Наприклад макрос, який змінює значення двох змінних:



```
*main.cpp X
1  #include <iostream>
2  #define SWAP(type, a, b) type (tmp) = (a); (a) =(b); (b) = (tmp);
3  using namespace std;
4  int main()
5  {
6      int x=5, y=6;
7      SWAP(int,x,y);
8      cout<<x<<" "<<y;
9      return 0;
10 }
```

Макроси також можна записувати в кілька рядків, але тоді кожен рядок, крім останнього, має закінчуватися символом '\\'.
\\

Техніка безпеки під час роботи з макросами

Є кілька основних правил, яких потрібно дотримуватися під час роботи з макросами.

1. Параметрами макросів не повинні бути вирази та виклики функцій.
2. Всі аргументи макросу та, як правило, сам макрос мають бути укладені у дужки.
3. Багаторядкові макроси повинні мати свою область видимості.

Наприклад, у нас є макрос, який викликає дві функції:

```
#define MACRO() doSomething();\  
doSomethinElse();
```

А тепер спробуємо використати цей макрос у такому контексті:

```
if (some_condition) MACRO()
```

Після макропідстановки ми побачимо таку картину:

```
if (some_condition) doSomething();doSomethinElse();
```

Неважко помітити, що під дію if потрапить лише перша функція, а друга викликатиметься завжди. Саме для того, щоб уникнути такого, у макросів має бути оголошено свою область видимості. Для зручності з цією метою прийнято використовувати цикл `do { } while (0);`.

```
#define MACRO() do { \  
    doSomething(); \  
    doSomethingElse(); \  
} while(0)
```

ПРОГРАМУВАННЯ

Оскільки в умові циклу стоїть нуль, він відпрацює рівно один раз. Це робиться, по-перше, для того, щоб у тіла макросу з'явилась своя область видимості, обмежена тілом циклу, а по-друге, щоб зробити виклик макросу звичнішим, тому що тепер після MACRO() потрібно буде ставити крапку з комою.

Директиви умовної компіляції

До них відносяться

`#if` з директивами `#elif`, `#else` і `#endif`

Дозволяють проводити вибірккову компіляцію програми

Директива `#if` управляє компіляцією частин вихідного файла. Якщо вираз, який написано після `#if` має ненульове значення, група рядків відразу після директиви `#if` зберігається в записі перетворення.

Кожна директива `#if` повинна відповідати директиві, що закриває `#endif` .

Між директивами `#if` та `#endif` може використовуватися будь-яке число директив `#elif`, але допускається не більше однієї директиви `#else`.

Директива `#else`, якщо вона є, має бути останньою директивою перед `#endif`.

Основна форма використання директиви наступна:

```
#if вираз  
послідовність операторів  
#elif вираз1  
послідовність операторів  
#elif вираз2  
послідовність операторів  
...  
#elif виразN  
послідовність операторів  
#endif
```

Інший метод умовної компіляції є у використанні директив

`#ifdef` та `#ifndef`

Основна форма використання

`#ifdef` ім'я макросу

Послідовність операторів

`#endif`

Та, відповідно

`#ifndef` ім'я макросу

Послідовність операторів

`#endif`

Якщо макрос визначено, то при використанні `#ifdef` компілюються відповідна послідовність операторів до `#endif`. Якщо макрос не визначено або був відмінений директивою `#undef`, то відповідна послідовність операторів компілятором ігнорується.

Директива `#ifndef` діє протилежним чином.

Для того, щоб мати можливість перевіряти, чи визначені комбінації макросов у препроцесорі, визначено операцію `defined`.

Її можливо використовувати тільки у директивах `#if` та `#elif`.

Вираз `defined` (ім'я макросу) приймає значення 1 (істина), якщо макрос визначено.

Директиви

`#if defined (MAX)`

та

`#ifdef (MAX)`

надають однакову відповідь. Перевагою є можливість використання директив наступного вигляду

`#if defined (MAX) && #if defined (MIN)`

Макрос, що має значення `NULL`, вважається визначеним.

ПРОГРАМУВАННЯ

```
1  #include <iostream>
2  #define MAX 10
3  using namespace std;
4  int main()
5  {
6  #if MAX>99
7  cout<<"MAX is bigger 99"<<"\n";
8  #else
9  cout<<"MAX is equal to "<<MAX<<"\n";
10 #endif // MAX
11     return 0;
12 }
```

```
1  #include <iostream>
2  #define MAX 100
3  using namespace std;
4  int main()
5  {
6  #if MAX>99
7  cout<<"MAX is bigger 99"<<"\n";
8  #else
9  cout<<"MAX is equal to "<<MAX<<"\n";
10 #endif // MAX
11     return 0;
12 }
```

Директива `#error` (C/C++)

Директива `#error` видає вказане користувачем повідомлення про помилку під час компіляції, а потім завершує компіляцію.

Директиви `#pragma` визначають функції компілятора, що залежать від комп'ютера або операційної системи.

Ключове слово, яке стосується тільки компілятора Microsoft, дозволяє створювати `pragma` директиви коду всередині визначень макросів.

Стандарт ANSI мови C визначає 5 попередньо визначених макросів:

__LINE__
__FILE__
__DATE__
__TIME__
__STDK__

(Макрос __STDK__ має значення 1, якщо компіляція програми проводилась зі ввімкненою опцією Keyword only. Інакше він не визначений)

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout<<__FILE__<<"\n";
6      cout<<__LINE__<<"\n";
7      cout<<__DATE__<<"\n";
8      cout<<__TIME__<<"\n";
9      return 0;
10 }
```

```
D:\Cpp\c143\main.cpp
6
Dec  6 2021
20:49:49
```

Деякі стандартні заголовні файли

assert.h – діагностика програм

errno.h - перевірка помилок

math.h – математична бібліотека

stdio.h - бібліотека стандартного введення-виведення мови C

time.h - функції для роботи з датами та часом

ОПИС ВСІХ функцій, макросів та системних змінних розташовано у документації!

ДЯКУЮ ЗА УВАГУ!