پیش پردازنده ها و ماکروها

دستورات پیش پردازنده که اغلب برای تعریف ثابت و ماکرو یا ضمیمه کردن فایل بکار می روند قبل از ترجمه برنامه تفسیر و تغییراتی را در متن برنامه ایجاد می کنند.

پیش پردازنده
راهنما ها
ماکروها

پیش پردازنده

پیش پردازنده (preprocessor) بخشی از کامپایلر است که قسمت هائی از برنامه را، قبل از اینکه کل آن توسط کامپایلر ترجمه شود، مورد ارزیابی قرار می دهد. برنامه نویس می تواند دستوراتی را در برنامه خود درج کند که مستقیما پیش پردازنده را فراخوانی کند به این دستورات پیش پردازنده می گویند. پیش پردازنده ها ممکن است باعث تغییر متن برنامه قبل از تحویل آن به کامپایلر شوند.

قبلا از پیش پردازنده ها در ضمیمه کردن فایل یا تعریف ثابت استفاده کرده اید.

سه کاربرد اصلی برای پیش پردازنده ها وجود دارد:

- راهنماها
- ثابت ها
- ماکروها

راهنماها

راهنماها (directives) دستوراتی هستند که توسط برنامه نویس به پیش پردازنده داده می شوند تا عمل خاصی را انجام دهد. برای مثال ثابتی را در متن جایگزین کند، محتوای فایل دیگری را در فایل مبدا درج کند یا بخشی از کد را ترجمه نکند.

راهنماها باعث می شوند متن برنامه به سادگی تغییر کند و درمحیط های مختلف قابل کامپایل باشد.

خطوط پیش پردازنده قبل از بسط ماکروها تشخیص و اجرا می شود. بنابراین اگر ماکرو به چیزی مشابه یک فرمان پیش پردازنده بسط داده شود دستور توسط پیش پردازنده تشخیص داده نمی شود.

پیش پردازنده راهنماهای زیر را تشخیص می دهد:

#define	#error	#include	#if
#else	#elif	#endif	#ifdef
#ifndef	#import	#line	#pragma
#undef	#using		

نکته. راهنماها همیشه با علامت # (sharp sign) شروع می شوند.
نکته. بین علامت # و اولین کاراکتر راهنما می تواند فاصله باشد.
نکته. بعضی راهنماها شامل آرگومان ها و مقادیر هستند. هر متن دیگری بجز آرگومان و مقدار که به دنبال راهنما می آید ابتدایش باید علامت توضیح باشد (//) یا در بین علائم (/* */) قرار بگیرد.
نکته. خط شامل راهنما می تواند با علامت انتهای خط (\) ختم شود.
نکته. راهنماها می توانند در هرجائی از فایل منبع قرار بگیرند اما تنها روی بقیه کد تاثیر می گذارند.

ثابت ها

همانطور که در بخش ثابت ها توضیح داده شده است از راهنمای #define می توان برای تعریف یک ثابت سمبلیک استفاده کرد. شکل کلی تعریف ثابت به صورت زیر است:

```
#define [identifier name] [value]
```

پیش پردازنده قبل از کامپایل در متن برنامه شناسه Identifier name را با مقدار value جایگزین خواهد کرد.

اگر ثابت را به صورت یک عبارت محاسباتی تعریف کنید بهتر است مقدار را درون پرانتز قرار دهید. به این ترتیب الویت عملیات حفظ خواهد شد.

مثال. تعریف ثابت زیر در نظر بگیرید:

```
#define PI_PLUS_ONE (3.14 + 1)
```

که به صورت زیر استفاده می شود:

```
x = PI_PLUS_ONE * 5;
```

اگر پرانتزها را قرار ندهیم عبارت به صورت زیر محاسبه خواهد شد، یعنی ابتدا عمل 1*5 انجام می شود سپس عمل جمع.

```
x = 3.14 + 1 * 5;
```

ضمیمه کردن فایل

راهنمای #include به پیش پردازنده می گوید که متن یک فایل را بگیرد و در برنامه جاری درج کند. معمولا راهنمای #include در ابتدای برنامه قرار می گیرد. به همین دلیل نام header file به فایل هائی که ضمیمه می شوند گفته می شود.

نام فایل هدر در مقابل راهنمای #include قرار می گیرد. درج نام فایل بین علائم < > یا " " نحوه جستجوی فایل را مشخص می کند. اگر نام فایل بین علائم < > محصور باشد کامپایلر فایل را در مسیرهای تعیین شده در بخش تنظیمات کامپایلر جستجو می کند ولی اگر نام فایل مابین علائم " " قرار گیرد کامپایلر آنرا ابتدا در مسیر جاری برنامه جستجو می کند.

فایل های هدر استاندارد نظیر iostream.h بین علائم < > محصور می شوند.

کامپایل شرطی

مجموعه ای از راهنماها وجود دارند که تعیین می کنند آیا خطوط برنامه قبل از تحویل به کامپایلر حذف شوند یا خیر. این راهنماها شامل #if، #elif، #else، #ifdef و #ifndef هستند.

یک بلوک شرطی که با یکی از راهنماهای شرطی شروع می شود حتما باید به #endif ختم شود. بهتر است مقابل عبارت #endif توضیحی باشد که مشخص شود کدام بلوک شرطی بسته شده است.

یکی از کاربردهای راهنماهای شرطی وقتی است که فایل هدری در چند فایل هدر دیگر که باید در برنامه اصلی ضمیمه شوند مورد نیاز است. مشکلی که پیش می آید این است که متغیرها، ثابت ها، کلاس ها و توابع فایل هدر چندبار در برنامه ظاهر خواهند شد که سرباری برای کامپایلر می شود. با پیش پردازنده ها به راحتی می توان تضمین کرد که هر فایل هدر تنها یکبار در برنامه اصلی اضافه می شود. راهنمای #ifndef (if not defined) بلوکی از متن را تنها اگر عبارت خاصی قبلا تعریف نشده باشد اجرا می کند.

کدی که در #idndef ضمیمه می شود تنها یکبار زمانی که فایل لود می شود بار می شود.

```
#ifndef _FILE_NAME_H_
#define _FILE_NAME_H_

/* code */

#endif // #ifndef _FILE_NAME_H_
```

شناسه ای که مقابل #ifndef ذکر می شود لازم نیست حتما مقداری داشته باشد زیرا با اضافه شدن خط #define تعریف می شود.

مثال. تعریف ثابت خاص NULL توسط راهنماهای شرطی.

```
#ifndef NULL
#define NULL (void *)0
#endif // #ifndef NULL
```

مثال. راهنماهای شرطی برای بلوکی که شامل توضیحات چندخطی است و می خواهید همگی توضیحی شوند روش خوبی است.

```
#if 0
/* comment ...
*/

// code

/* comment */
#endif
```

ماکروها

همانطور که گفته شد با راهنمای #define یک شناسه را می توان به یک ثابت نسبت داد تا یک ثابت سمبلیک شکل بگیرد. از راهنمای #define برای مربوط کردن شناسه های معنی دار به عباراتی که زیاد استفاده می شوند هم بکار می رود. وقتی یک شناسه به یک عبارت یا جمله ربط داده می شود ماکرو (macro) نامیده می شود.

شکل کلی تعریف ماکرو به صورت زیر است:

```
#define MACRO_NAME(arg1, arg2, ...) [code to expand to]
```

arg1، arg2 و ... آرگومان های ماکرو هستند که تعداد آنها اختیاری است.

مثال. ماکرو افزایش عدد.

```
#define INCREMENT(x)  ((x)++)
```

مثال. ماکرو مربع عدد.

```
#define SquareOf(x)  ((x)*(x))
double yout,xin=3;
yout = SquareOf(xin);
```

ماکرو می تواند چندخطی باشد برای اتصال خطوط در انتهای هر خط باید علامت \ قرار بگیرد. خط آخر نیازی به علامت \ ندارد.

مثال. ماکرو جابه جا کردن دو مقدار.

```
#define SWAP(a, b)  {          \
                     a ^= b;   \
                     b ^= a;   \
                     a ^= b;   \

                    }
```

وقتی پیش پردازنده با نام ماکرو در برنامه مواجه می شود آن را یک فراخوانی به ماکرو تلقی می کند و یک کپی از بدنه ماکرو را جایگزین نام ماکرو می کند. اگر ماکرو دارای پارامترهائی باشد، آرگومان هائی که بدنبال نام ماکرو قرار دارند جایگزین آن ها در بدنه ماکرو می شوند.

در عمل دو نوع ماکرو وجود دارد: شیء گونه که پارامتری ندارد و تابع گونه که می تواند آرگومان هائی را بپذیرد و بسیار شبیه فراخوانی تابع می ماند.

یک ماکرو را می توان دوباره با همان مقدار قبلی تعریف کرد ولی تعریف مجدد ماکرو با مقدار جدید ممکن نیست. مگر اینکه تعریف قبلی را حذف کنید. راهنمای #undef برای حذف تعریف یک ماکرو است.

چون ماکرو سربار فراخوانی تابع را ندارد سریع تر از تابع عمل می کند. اما بخاطر داشته باشید فراخوانی ماکرو تنها یک جایگزینی عبارت است و کامپایلر آن را بلافاصله تفسیر نمی کند بنابراین کاری در ارتباط با الویت عملگرها، سازگاری نوع، قواعد انتساب یا ارجاع و موارد دیگر ندارد. در نتیجه مخاطراتی را بدنبل خواهد داشت که به برخی از آنها در ادامه اشاره می شود. بنابراین بهتر است با احتیاط تعریف شود یا اصلا استفاده نشود.

الویت عبارت

یک راه برای اطمینان از اینکه الویت عبارت در ماکرو حفظ می شود این است که آرگومان های ماکرو را درون پرانتز قرار دهید.

مثال. اگر ماکرو به صورت زیر تعریف و استفاده شده باشد انتظار دارید مقدار ۳۰ به متغیر z اختصاص داده شود درحالیکه مقدار ۱۳ را می گیرد.

```
#define MULT(x, y) x * y
int z = MULT(3 + 2, 4 + 2);
```

زیرا وقتی بدنه ماکرو جایگزین می شود به صورت زیر در می آید.

```
int z = 3 + 2 * 4 + 2;
```

برای حل این مشکل بهتر است ماکرو به صورت زیر نوشته شود.

```
#define MULT(x, y) ((x) * (y))
```

که به صورت زیر جایگزین خواهد شد.

```
int z = (3 + 2) * (4 + 2)
```

ماکروهای چند دستوری

ماکروهائی که مشابه تابع مقداری را محاسبه و برمی گردانند expression macro نامیده می شوند. و ماکرو می تواند به جای اینکه مقداری را محاسبه کند دارای چند دستور باشد که آنها را action macro می نامند.

بهتر است بدنه ماکروهای چنددستوری را درون علائم {} محصور کنید تا از بروز مشکل جلوگیری کنید.

مثال. وقتی ماکرو فراخوانی شود تنها اولین دستور آن a^=b; درون شرط قرار می گیرد و دو دستور دیگر همواره اجرا خواهند شد.

```
#define SWAP(a, b)  a ^= b; b ^= a; a ^= b;

int x = 10;
int y = 5;

SWAP(x, y);   // works OK
// What happens now?
if(x < 0)
    SWAP(x, y);
```

بهتر است ماکرو فوق به صورت زیر تعریف شود:

```
#define SWAP(a, b)  do { a ^= b; b ^= a; a ^= b; } while ( 0 )
```

نکته. قرار دادن کل بدنه ماکرو درون پرانتز زمانی که ماکرو مقداری را برنمی گرداند الزامی نیست.
نکته. در C++ از ماکروها تا حد ممکن استفاده نمی شود چون در اکثر موارد نیازی به آنها دیده نمی شود. بیشتر از عبارت const برای تعریف ثابت و توابع درون خطی به ماکروها ترجیح داده می شوند زیرا مشکلات ماکروها را ندارند.

مثال. تابع درون خطی محاسبه مربع عدد.

```
void SquareOf(int x) {return x*x;}
```

نکاتی درباره نوشتن ماکرو

۱ - استفاده از یک قرارداد نامگذاری برای اسامی ماکرو تشخیص آنها را در برنامه ساده تر می کند. برای مثال نام همه ماکروها با حرف m شروع شود.
۲ - درنظر داشته باشید چه نوع ماکروئی می نویسید عبارتی یا چنددستوری.
۳ - کل بدنه ماکرو را در پرانتز قرار دهید.
۴ - در بدنه ماکرو آرگومان ها را درون پرانتز قرار دهید.
۵ - در ماکروهایچنددستوری هر دستور را به علامت سمیکولن ختم کنید و کل بدنه ماکرو را درون آکولاد قرار دهید.
۶ - کلیه ماکروها را در یک فایل هدر قرار دهید.

ماکروهای تعریف شده

تعدادی ماکرو در C++ تعریف شده است که می توانید از آنها استفاده کنید. کامپایلر قادر به شناسائی ماکروهای پیش تعریف شده است. این ماکروها آرگومانی نمی گیرند و مجدد قابل تعریف نیستند.

بعضی از آنها که توسط ANSI تعریف شده است در جدول زیر آمده است:

ماکرو	شرح
__DATE__	تاریخ ترجمه برنامه فعلی به صورت یک رشته Mmm dd yyyy. مشابه تاریخی که تابع asctime در TIME.H تولید می کند.
__FILE__	نام فایل جاری.
__LINE__	شماره خط در فایل جاری. مشابه راهنمای #line
__TIME__	زمان اخرین ترجمه برنامه فعلی به فرمت hh:mm:ss
__TIMESTAMP__	تاریخ و ساعت آخرین تغییرات فایل فعلی به صورت Ddd Mmm Date hh:mm:ss yyyy