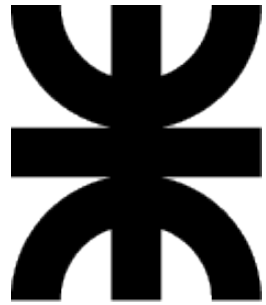
Informática II

El compilador de C del proyecto GNU (`gcc, g++`) – El preprocesador

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

El preprocesador

- ¿Qué orden le corresponde a la etapa de preprocesamiento?

El preprocesador

- ¿Qué orden le corresponde a la etapa de preprocesamiento?
- ¿Qué función realiza el preprocesador?

El preprocesador

- ¿Qué orden le corresponde a la etapa de preprocesamiento?
- ¿Qué función realiza el preprocesador?

El preprocesamiento ocurre antes de la compilación de un programa.

El preprocesador

- ¿Qué orden le corresponde a la etapa de preprocesamiento?
- ¿Qué función realiza el preprocesador?

El preprocesamiento ocurre antes de la compilación de un programa.

Algunas de la acciones son:

1. Inclusión de otros archivos dentro del archivo a compilar.
2. Definición de constantes simbólicas y macros.
3. Compilación condicional del código del programa.

El preprocesador – Inclusión

La directiva del preprocesador `#include` provoca la inclusión de una copia del archivo especificado en el lugar de la directiva.

El preprocesador – Inclusión

La directiva del preprocesador `#include` provoca la inclusión de una copia del archivo especificado en el lugar de la directiva.

Tiene dos formas:

```
1 #include <nombre de archivo>
2 #include "nombre de archivo"
```

El preprocesador – Inclusión

La directiva del preprocesador `#include` provoca la inclusión de una copia del archivo especificado en el lugar de la directiva.

Tiene dos formas:

```
1 #include <nombre de archivo>
2 #include "nombre de archivo"
```

Difieren en la ubicación en la que el preprocesador busca el archivo a incluir:

1. Nombre del archivo entre llaves angulares (< y >) se utiliza por los *encabezados de la biblioteca estándar*.
2. Nombre del archivo entre comillas: busca el archivo en el mismo directorio en donde se encuentra el archivo que va a compilarse.

El preprocesador – Inclusión

La directiva del preprocesador `#include` provoca la inclusión de una copia del archivo especificado en el lugar de la directiva.

Tiene dos formas:

```
1 #include <nombre de archivo>
2 #include "nombre de archivo"
```

Difieren en la ubicación en la que el preprocesador busca el archivo a incluir:

1. Nombre del archivo entre llaves angulares (< y >) se utiliza por los *encabezados de la biblioteca estándar*.
2. Nombre del archivo entre comillas: busca el archivo en el mismo directorio en donde se encuentra el archivo que va a compilarse.

Las declaraciones que se incluyen en archivos de cabecera son de estructuras y uniones, enumeraciones y prototipos de funciones.

El preprocesador – Contantes simbólicas

La directiva `#define` crea constantes simbólicas (constantes representadas por símbolos) y macros (operaciones definidas como símbolos). El formato es:

```
#define identificador texto de reemplazo
```

El preprocesador – Contantes simbólicas

La directiva `#define` crea constantes simbólicas (constantes representadas por símbolos) y macros (operaciones definidas como símbolos). El formato es:

```
#define identificador texto de reemplazo
```

Hace que las ocurrencias subsecuentes del identificador sean reempazadas con el texto de reemplazo. Por ejemplo:

```
#define PI 3.14159
```

reemplaza todas las ocurrencias de la constantes simbólica `PI` con la constante numérica `3.14159`.

El preprocesador – Macros

- Una macro es un identificador definido dentro de una directiva de preprocesador `#define`.
- Como en las constantes simbólicas, el identificador de la macro se reemplaza en el programa con el texto de reemplazo.

El preprocesador – Macros

- Una macro es un identificador definido dentro de una directiva de preprocesador `#define`.
- Como en las constantes simbólicas, el identificador de la macro se reemplaza en el programa con el texto de reemplazo.

Las macros se puede definir con o sin argumentos:

Macro sin argumentos: se procesa como una constante simbólica.

Macro con argumentos: los argumentos se sustituyen dentro del texto de reemplazo y después se desarrolla la macro.

El preprocesador – Macros

- Una macro es un identificador definido dentro de una directiva de preprocesador `#define`.
- Como en las constantes simbólicas, el identificador de la macro se reemplaza en el programa con el texto de reemplazo.

Las macros se puede definir con o sin argumentos:

Macro sin argumentos: se procesa como una constante simbólica.

Macro con argumentos: los argumentos se sustituyen dentro del texto de reemplazo y después se desarrolla la macro.

Por ejemplo:

```
#define AREA_CIRCULO( x ) ( (PI) * (x) * (x) )
```

El preprocesador – Macros

- Una macro es un identificador definido dentro de una directiva de preprocesador `#define`.
- Como en las constantes simbólicas, el identificador de la macro se reemplaza en el programa con el texto de reemplazo.

Las macros se puede definir con o sin argumentos:

Macro sin argumentos: se procesa como una constante simbólica.

Macro con argumentos: los argumentos se sustituyen dentro del texto de reemplazo y después se desarrolla la macro.

Por ejemplo:

```
#define AREA_CIRCULO( x ) ( (PI) * (x) * (x) )
```

Cuando apareza `AREA_CIRCULO(y)` en el archivo, el valor de `x` se sustituirá por `y` dentro del texto de reemplazo.

El preprocesador – Macros

Por ejemplo, la línea:

```
area = AREA_CIRCULO(4);
```

se desarrolla como:

```
area = ( ( 3.14159 ) * (4) * (4) );
```

y el valor de la expresión se evalúa y se asigna a la variable `area`.

El preprocesador – Macros

Por ejemplo, la línea:

```
area = AREA_CIRCULO(4);
```

se desarrolla como:

```
area = ( ( 3.14159 ) * (4) * (4) );
```

y el valor de la expresión se evalúa y se asigna a la variable `area`.

Los paréntesis alrededor de cada `x` dentro del texto de reemplazo fuerza el orden apropiado de evaluación, cuando el argumento de la macro es una expresión:

```
area = AREA_CIRCULO(c + 2);
```

se desarrolla como:

```
area = AREA_CIRCULO( (3.14159) * (c + 2) * (c + 2) );
```

El preprocesador – Macros

Por ejemplo, la línea:

```
area = AREA_CIRCULO(4);
```

se desarrolla como:

```
area = ( ( 3.14159 ) * (4) * (4) );
```

y el valor de la expresión se evalúa y se asigna a la variable `area`.

Los paréntesis alrededor de cada `x` dentro del texto de reemplazo fuerza el orden apropiado de evaluación, cuando el argumento de la macro es una expresión:

```
area = AREA_CIRCULO(c + 2);
```

se desarrolla como:

```
area = AREA_CIRCULO( (3.14159) * (c + 2) * (c + 2) );
```

Las macros pueden tener más de un argumento. Por ejemplo:

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

El preprocesador – Macros con valor

dtestval.c

```c
#include <stdio.h>

int main(void)
{
  printf("El valor de NUM es %d\n", (NUM));
  return 0;
}
```

El preprocesador – Macros con valor

dtestval.c

```c
#include <stdio.h>

int main(void)
{
  printf("El valor de NUM es %d\n", (NUM));
  return 0;
}
```

```
$> gcc -Wall -DNUM=100 dtestval.c
$> ./a.out
El valor de NUM es 100
```

(-DNAME=VALUE)

El preprocesador – Macros con valor

dtestval.c

```c
#include <stdio.h>

int main(void)
{
  printf("El valor de NUM es %d\n", (NUM));
  return 0;
}
```

```
$> gcc -Wall -DNUM=100 dtestval.c
$> ./a.out
El valor de NUM es 100
```

(-DNAME=VALUE)

```
$> gcc -Wall -DNUM="2+2" dtestval.c
$> ./a.out
El valor de NUM es 4
```

El preprocesador – Macros con valor

dtestval1.c

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5   printf("Diez veces NUM es %d\n", 10 * (NUM));
6   return 0;
7 }
```

El preprocesador – Macros con valor

dtestval1.c

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5   printf("Diez veces NUM es %d\n", 10 * (NUM));
6   return 0;
7 }
```

```
$> gcc -Wall -DNUM="2+2" dtestval1.c
$> ./a.out
El valor de NUM es 40
```

El preprocesador – Macros con valor

dtestval1.c

```c
#include <stdio.h>

int main(void)
{
  printf("Diez veces NUM es %d\n", 10 * (NUM));
  return 0;
}
```

```
$> gcc -Wall -DNUM="2+2" dtestval1.c
$> ./a.out
El valor de NUM es 40
```

¿Qué valor se imprime si no se ponen los paréntesis en la macro?

El preprocesador – Macros con valor

dtestval1.c

```c
#include <stdio.h>

int main(void)
{
  printf("Diez veces NUM es %d\n", 10 * (NUM));
  return 0;
}
```

```
$> gcc -Wall -DNUM="2+2" dtestval1.c
$> ./a.out
El valor de NUM es 40
```

¿Qué valor se imprime si no se ponen los paréntesis en la macro?

Valor por defecto de una macro

```
$> gcc -Wall -DNUM dtestval.c
$> ./a.out
El valor de NUM es 1
```

El preprocesador – Macros con valor

dtestval1.c

```c
#include <stdio.h>

int main(void)
{
  printf("Diez veces NUM es %d\n", 10 * (NUM));
  return 0;
}
```

```
$> gcc -Wall -DNUM="2+2" dtestval1.c
$> ./a.out
El valor de NUM es 40
```

¿Qué valor se imprime si no se ponen los paréntesis en la macro?

Valor por defecto de una macro

```
$> gcc -Wall -DNUM dtestval.c
$> ./a.out
El valor de NUM es 1
```

Una macro vacía `-DNUM=""` queda definida (`#ifdef`) pero no se expande a nada.

El preprocesador – Compilación condicional

dtest.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5  #ifdef TEST
6    printf("Modo test\n");
7  #endif
8    printf("Ejecutando...\n");
9    return 0;
10 }
```

El preprocesador – Compilación condicional

dtest.c

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5  #ifdef TEST
6    printf("Modo test\n");
7  #endif
8    printf("Ejecutando...\n");
9    return 0;
10 }
```

```
$> gcc -Wall dtest.c
$> ./a.out
Ejecutando...
```

El preprocesador – Compilación condicional

dtest.c

```c
#include <stdio.h>

int main(void)
{
#ifdef TEST
  printf("Modo test\n");
#endif
  printf("Ejecutando...\n");
  return 0;
}
```

```
$> gcc -Wall dtest.c
$> ./a.out
Ejecutando...
```

Macro definida desde la línea de comandos

```
$> gcc -Wall -DTEST dtest.c
$> ./a.out
Modo test
Ejecutando...
```

El preprocesador – Compilación condicional

dtest.c

```c
#include <stdio.h>

int main(void)
{
#ifdef TEST
  printf("Modo test\n");
#endif
  printf("Ejecutando...\n");
  return 0;
}
```

```
$> gcc -Wall dtest.c
$> ./a.out
Ejecutando...
```

Macro definida desde la línea de comandos

```
$> gcc -Wall -DTEST dtest.c
$> ./a.out
Modo test
Ejecutando...
```

(probar utilizando `gcc -E`)