CPROUC/2021-2022

Jesse op den Brouw

CPROUC

Preprocessor

DE HAAGSE HOGESCHOOL

Preprocessor

- De preprocessor is een faciliteit (een programma) dat vóór de echte C-compiler gestart wordt.
- De preprocessor is geen C-compiler.

- De preprocessor doet:
  - Inlezen header bestanden (`stdio.h`, `math.h`, …)
  - Verwerken van macro's (`#define`)
  - Verwerken van pragma's (`#pragma` …)
  - Conditionele compilatie (`#if`, `#elif`, `#else`, `#ifdef`)

#include

- Met `#include` kan een header-bestand worden ingelezen (in principe elk bestand).
- Gebruik < en > voor systeem-header-bestanden:

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdint.h>
```

- Gebruik " en " voor eigen gemaakte header-bestanden:

```
#include "studenten.h"
```

#include

- In een header-bestand staan:
- Macro-definities
- Structure declaraties
- Enum declaraties
- Typedef
- Functie-declaraties (functie-prototypes)

- Voordeel: je hoeft het niet zelf te typen en het is gestandaardiseerd.

Macro's

- Met `#define` definieer je een macro:
- Simpele vorm:

```
#define AANTAL 10
```

- Nu kan je overal in het C-programma `AANTAL` gebruiken als *vervanging* van `10`.

```
int rij[AANTAL];
```

- Let op: `AANTAL` is geen variabele! De C-compiler ziet: `int rij[10];`

DE HAAGSE HOGESCHOOL

Macro's

- Met `#define` definieer je een macro:
- Vorm met parameters:

```
#define kwadraat(A) A * A
```

- Nu kan je overal in het C-programma `kwadraat` gebruiken als *vervanging* van `A * A`.

```
int k = kwadraat(2);
```

- Let op: de C-compiler ziet dit als: `int k = 2 * 2;`

Macro's

- Vorm met parameters:

```
#define kwadraat(A) A * A
```

- Dit gaat fout met:

```
int k = kwadraat(z + 1);
```

- De C-compiler ziet nu:

```
int k = z + 1 * z + 1;  // * gaat voor op +
```

Macro's

- Gebruik extra haakjes!

```
#define kwadraat(A) ((A) * (A))
```

- Nu gaat het goed:

```
int k = kwadraat(z + 1);
```

- De C-compiler ziet nu:

```
int k = ((z + 1) * (z + 1));
```

Macro's

- Het kan nog steeds fout gaan!

```
#define kwadraat(A) ((A) * (A))
```

- Nu met post-increment

```
int k = kwadraat(z++);
```

- De C-compiler ziet nu:

```
int k = ((z++) * (z++));
```

- De eerste z++ wordt direct na gebruik verhoogd.

Macro's

- Voordeel: onafhankelijk van het datatype

```
#define kwadraat(A) ((A) * (A))
```

- Dus:

```
int k = kwadraat(3);
float f = kwadraat(3.14f);
double d = kwadraat(3.1415926536);
```

Macro's

- Maximum bepalen:

```
#define max(A,B,D) if ((A) > (B)) { D = (A); } else { D = (B); }
```

- Dus:

```
int c;
max(2, 3, c); /* punt-komma is niet noodzakelijk! */
```

- De macro wordt vervangen door het `if-else`-statement.

Macro's

- Macro over meerdere regels:

```
#define swap(A, B) { \
    int temp = A; \
    A = B; \
    B = temp; }
```

- Het gebruik van de accolades zorgt ervoor dat temp een zeer locale variabele is.
- Gebruik nu:

```
swap(x, y);   /* punt-komma is niet noodzakelijk! */
```

Pragma's

- Pragma's zijn specifiek voor een compiler en voor een processor-architectuur:

```
#pragma warning(disable : 4996)

#pragma GCC diagnostic warning "-Wformat"
```

Conditionele compilatie

- Met behulp van `#if`, `#elif`, `#else` en `#endif` kan je bepaalde stukken C-programma compileren (of juist niet):

```
#if _MSC_VER == 1927
char compiler[] = "Visual Studio 2019 version 16.7";
#elif _MSC_VER == 1928
char compiler[] = "Visual Studio 2019 version 16.8 or 16.9";
#elif _MSC_VER = 1929
char compiler[] = "Visual Studio 2019 version 16.10 or 16.11";
#else
char compiler[] = "Visual Studio";
#endif
```

Conditionele compilatie

- Met behulp van `#if`, `#elif`, `#else` en `#endif` kan je bepaalde stukken C-programma compileren (of juist niet):

```
#if android > 1102 && android < 1501
char version[] = "Android 11";
#else
char version[] = "Android";
#endif
```

Conditionele compilatie

- Met behulp van `#ifdef` en `#ifndef` kan je bepaalde stukken C-programma compileren (of juist niet):

```
#ifdef android
#include <android.h>
#endif
#ifdef windows
#include <windows.h>
#endif
```

Conditionele compilatie

- Bij gebruik van debuggen (of niet)

```
int c; /* werk met variabele c */
…
#define DEBUG
…
#ifdef DEBUG
printf("DEBUG: c = %d\n", c);
#endif
```

- Alleen als DEBUG gedefinieerd is, wordt de `printf`-regel gecompileerd. De waarde van DEBUG is niet interessant

Code distributie

- Als je een aantal functies hebt geschreven en je wilt die distribueren, maak je een C-bestand en een header-bestand. In je C-bestand zet je:
  - Globale variabelen
  - Functie-definities (dus met een body)
- Andere gebruikers kunnen zo jouw functies gebruiken.
- In Visual Studio, Code::Blocks en Xcode: voeg het bestand toe aan het project.

Code distributie

- Als je een aantal functies hebt geschreven en je wilt die distribueren, maak je een header-bestand aan met daarin:
  - Macro-definities
  - Structure declaraties
  - Enum declaraties
  - Typedefs
  - Functie-declaraties (functie-prototypes, dus zonder code)

- Andere gebruikers kunnen zo jouw functies gebruiken.

Code distributie

- Het C-bestand `studenten.c`:

```c
double gemiddelde(student_t studs[], int aantal)
{
   /* definitie is hier */
}
double maximum(student_t studs[], int aantal)
{
   /* definitie is hier */
}
```

Code distributie

- Het H-bestand (header-bestand) `studenten.h`:

```
typedef struct student_struct {
    int idcode;
    char achternaam[30];
    char klas[10];
    double resultaat;
} student_t;
double gemiddelde(student_t studs[], int aantal);
double maximum(student_t studs[], int aantal);
```

Code distributie

- In het C-bestand met `main`:

```c
#include <stdio.h>
#include "studenten.h"
int main(void) {
    student_t studenten[50];
    int aantal = sizeof studenten / sizeof studenten[0];
    ...
    double gem = gemiddelde(studenten, aantal);
    ...
}
```

Conditionele inclusie

- Met behulp van `#ifndef` kan een header-bestand slechts één keer verwerkt worden

```
#ifndef _STUDENTEN_H
#define _STUDENTEN_H
   /* alle declaraties, typedefs, enums etc hier */
#endif
```

- De eerste keer dat dit bestand ingelezen wordt, is _STUDENTEN_H nog niet gedefinieerd en wordt de *body* verwerkt.
- Bij een volgende inlezing is _STUDENTEN_H wel gedefinieerd en wordt de body niet verwerkt.

Predefined macro's

- Elke toolchain definieert een aantal bekende macro's:

  `__FILE__` – de gecompileerde bestandsnaam

  `__LINE__` – het huidige regelnummer

  `__DATE__` – de datum tijdens compilatie

  `__TIME__` – de tijd tijdens compilatie

  `__STDC__` – compiler draait in ISO Standard C.

- Kunnen gebruikt worden voor debugging:

```
fprintf (stderr, "Internal error: negative string length "
                 "%d at %s, line %d.",
                 length, __FILE__, __LINE__);
```

Predefined macro's

- Een toolchain definieert zelf een aantal macro's. Dit is per toolchain verschillend dus opzoeken op internet.

- Visual Studio:
  `_MSC_VER` – versienummer van de compiler.

- Er zijn nog meer macro's gedefineerd. Zie Visual Studio Predefined Macros.

Predefined macro's

- Een toolchain definieert zelf een aantal macro's. Dit is per toolchain verschillend dus opzoeken op internet.
- GNU C-compiler (Code::Blocks):

  `__GNUC__`

  `__GNUC_MINOR__`

  `__GNUC_PATCHLEVEL__`
- Als de GCC versie $x.y.z$ is, dan is `__GNUC__` x, `__GNUC_MINOR__` y en `__GNUC_PATCHLEVEL__` z.
- Er zijn nog meer macro's gedefineerd. Zie GNU C Predefined Macros.

let’s change