Preprocessorn

- Ett c-program innehåller ofta *preprocessordirektiv*.
- Instruktioner som analyseras *före* själva kompileringen av programmet
- Alla preprocessordirektiv inleds med “brädgårdsmärket” **#**
- Vi har alltså redan träffat på flera preprocessordirektiv:

  `#include` för att inkludera *headerfiler*
  `#ifndef, #define, #endif` för *include guards*
- Andra vanliga preprocessordirektiv:

  `#ifdef, #undef, #if, #elif, #else`

#define

- **`#define`** används för att namnge konstanter
- För att definiera en konstant **TRUE** med värdet 1 och en konstant **FALSE** med värdet 0:

```
#define TRUE 1
#define FALSE 0
```

- Observera syntaxen: Inget **=** mellan konstantens namn och dess värde, inget semikolon efter instruktionen!
- Konstanter skrivs ofta med versaler för att enkelt skilja dem från ickekonstanta värden
- Kan se **`#define`** som en “klipp-och-klistra” - instruktion till preprocessorn: Ersätt alla förekomster av konstantens namn med det definierade värdet innan kompileringen startar.
- Tidigare har vi använt nyckelordet **`const`** för att definiera konstanter – vilken metod är bättre?

#define

- En **`#define`**-konstant kan omdefinieras;
  en **`const`**-konstant kan inte omdefineras

```
#define VERY_IMPORTANT_CONSTANT 42
const int ANOTHER_VERY_IMPORTANT_CONSTANT = 79;

int main(void) {

    // This works (although gcc emits a warning)
    #define VERY_IMPORTANT_CONSTANT 43

    // This gives a compile-time error
    ANOTHER_VERY_IMPORTANT_CONSTANT = 80;
}
```

Användningsområden för #define

- **`#define`** kan, i motsats till **`const`**, användas för att ange dimensioner för globala räckor (jjfr. Chomp)
- **`#define`** kan användas för att definiera funktionsliknande makron, t.ex
  ```
  #define SQUARE(x) x*x
  #define SUM(x,y) x+y
  ```
  ...men normalt är det ändå bättre att göra “riktiga” funktioner med definierade datatyper
- **`#define`** kan användas för att implementera s.k. “conditional compilation”
  - “Include guards” är en variant av denna teknik

“Conditional compilation”

- Innebär att endast en del av programmet kompileras
- Typiska användningsområden:
  - Konstruktion av plattformsoberoende program
  - Debuggning
- Använd **`#define`** för att skapa en 'flagga' som indikerar t.ex. i vilken omgivning ett program skall köras, eller om debuggningsinformation skall skrivas ut
- Kompilera olika versioner av t.ex. samma funktion, beroende på flaggans värde.
- Kan kontrollera om en flagga **existerar** med **`#ifdef`**, och kontrollera **värdet** på en flagga med **`#if`**

“Conditional compilation” för debuggning

```
#define DEBUG
int main(void) {
    // ...part of the 'guess the secret number' game
    int secretNumber = rand()%100 + 1;
#ifdef DEBUG
    // Only output secret number if debug mode is on
    printf("Secret number is %d\n", secretNumber);
#endif
    printf("Enter a number between 1 and 100: ");
    scanf("%d", &num);
    ...
}
```

“Conditional compilation” för plattformsoberoende

```
#define PLATFORM 1
...
#if PLATFORM == 1 // Windows
const char* PATH = "C:\\Windows\\System32";
#elif PLATFORM == 2 // Linux
const char* PATH = "/bin";
#elif PLATFORM == 3 // Mac
...
#else
// Unknown platform, assume Windows
const char* PATH = "C:\\Windows\\System32";
#endif

printf("Path is %s\n", PATH);
```