Präprozessor

- Präprozessor nimmt *textuelle* Modifikationen an Übersetzungseinheit vor
- Ausführung automatisch und direkt vor Kompilation zu Binärdatei

`#define`

"`#define`" ⟨ident⟩ ["(" ⟨ident⟩ {"," ⟨ident⟩} ")"] ⟨text⟩

- Definiert token, wird überall folgend ersetzt durch gegebenen ⟨text⟩
- Kann parametrisiert sein, Parameter werden eingesetzt in ⟨text⟩ bei Ersetzung
- Wirklich nur textuelle Ersetzung; insb. keine Rücksicht auf Operatorvorrang

define_demo.cpp

```cpp
#include <iostream>
using namespace std;

#define SEVENTEEN (14 + 3)
#define PLUS(a, b) (a + b)

int main() {
  cout << SEVENTEEN << " "
       << PLUS(3, 14)
       << endl;
}
```

```
17 17
```

#ifdef

"#ifdef" ⟨ident⟩
"#ifndef" ⟨ident⟩
"#else"
"#endif"

- Fügt Text zwischen #ifdef und #endif ein, gdW. ⟨ident⟩ mit #define definiert wurde
- #ifndef negiert Bedingung

ifdef_demo.cpp

```cpp
#include <iostream>
using namespace std;

#define SEVENTEEN 17

int main() {
#ifdef SEVENTEEN
  cout << SEVENTEEN << endl;
#else
  cout << 18 << endl;
#endif
}
```

```
17
```

#include

"#include" ("" ⟨file⟩ "" | "<" ⟨ident⟩ ">")

- Findet Datei im lokalen System anhand von ⟨file⟩ bzw. ⟨ident⟩
- Ersetzt #include-Anweisung durch Inhalt der Datei
- Variante mit " sucht i.d.R. im gleichen Verzeichnis, Variante mit <, > i.d.R. in systemweiten Standardverzeichnissen

hello.cpp

```cpp
#include <iostream>

using namespace std;

int main() {
  cout << "Hello, World!" << endl;
}
```

```
Hello, World!
```

Objekt-Dateien

- Kompilieren einer Übersetzungseinheit liefert Objekt-Datei
- Bisher immer automatisch weiterverarbeitet zu ausführbarer Binärdatei
- `g++ -c hello.cpp` ⟶ `hello.o`
- Objektdatei enthält kompilierten Maschinencode und Metadaten bzgl. Funktionen etc., die *Symbole*

Linker

- Linker fügt beliebig viele Objektdateien zusammen zu ausführbarer Binärdatei
- `g++ hello.o` ⟶ `a.out`
- Sind benötigte Symbole in keiner der Objektdateien vorhanden ⟶ Linkerfehler

```
…/bin/ld: …/lib/crt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

Header-Dateien

- Motivation: selbe Funktion in mehreren Übersetzungseinheiten verwenden
- Kompilierte Instruktionen in Objekt-Datei notwendig, aber nicht hinreichend
- Insb. Typ, d.h. Deklaration wird auch benötigt
- Daher: Auslagern der Deklarationen in eigene Datei(en), Dateiendung `.h`
- Einbinden der Deklarationen mit `#include`, wo benötigt

Beispiel: Programmbibilothek

greet.h

```
#include <string>

void greet(std::string greeting);
```

greet.cpp

```
#include <iostream>
#include <string>
#include "greet.h"

using namespace std;

void greet(string greeting) {
  cout << greeting << endl;
}
```

hello1.cpp

```
#include <string>

using namespace std;

void greet(string greeting);

int main() {
  greet("Hello, World!");
}
```

```
g++ -c greet.cpp
g++ -c hello1.cpp
g++ hello1.o greet.o
./a.out
```

```
Hello, World!
```

hello2.cpp

```
#include "greet.h"

int main() {
  greet("Hello!");
}
```

```
Hello!
```

Double inclusion

grandparent.h

```
class Klasse {
  public: int attribut;
};
```

parent.h

```
#include "grandparent.h"
```

child.cpp

```
#include "grandparent.h"
#include "parent.h"

int main() {}
```

```
In file included from parent.h:1,
                 from child.cpp:2:
grandparent.h:1:7: error: redefinition of ‘class Klasse’
    1 | class Klasse {
      |       ^~~~~
In file included from child.cpp:1:
grandparent.h:1:7: note: previous definition of ‘class Klasse’
    1 | class Klasse {
      |       ^~~~~
```

Include guards

grandparent_guard.h

```
#ifndef GRANDPARENT_GUARD_H
#define GRANDPARENT_GUARD_H

class Klasse {
  public: int attribut;
};
#endif
```

parent_guard.h

```
#include "grandparent_guard.h"
```

child_guard.cpp

```
#include "grandparent_guard.h"
#include "parent_guard.h"

int main() {}
```

#pragma once

grandparent_pragma.h

```h
#pragma once

class Klasse {
  public: int attribut;
};
```

parent_pragma.h

```h
#include "grandparent_pragma.h"
```

child_pragma.cpp

```cpp
#include "grandparent_pragma.h"
#include "parent_pragma.h"

int main() {}
```