gcc preprocessor Anweisungen

Zeilen im C++ Quellcode, die mit # beginnen, werden als gcc prepocessor Anweisungen interpretiert und vor dem Kompilieren aufgelöst.

- gelten nur bis zum nächsten \n (newline)
- kein Semicolon am Zeilenende verwenden
- \ vor dem \n setzt die Anweisung in der nächsten Zeile fort

- Macros werden mit der directive `#define` und `#undef` definiert und vom preprocessor im laufenden Quelltext gesucht und durch den Macrotext ersetzt.

```
#define TEXT_ARRAY_SIZE 128
int char[TEXT_ARRAY_SIZE];  →  int char[128];
#undef TEXT_ARRAY_SIZE
#define TEXT_ARRAY_SIZE 1024
int char[TEXT_ARRAY_SIZE];  →  int char[1024];
```

Parameter Definition

Gültigkeit bis zum #undef

```
#define myloop for(int jL = 0 ; jL<5 ; jL++)
myloop {int k = jL * jL }
          →  for (int jL = 0;jL<5;jL++){int k = jL*jL}
```

ersetzen von myloop durch den Schleifenkopf

- Keine Syntaxprüfung durch den preprocessor !

gcc preprocessor Anweisungen

- Im Macrotext können auch Variable verwendet werden. Beispiele:

Macro Definition

```
#define myloop(x) for (int jL = 0 ; jL < x ; jL++)
```

im C++ Quellcode

```
myloop(20) {int k = jL * jL }
```

```
      →  for (int jL = 0;jL < 20; jL++){int k = jL*jL}
```

Macro Definition

```
#define myMin(x,y) ((x)<(y))?(x):(y))
```

im C++ Quellcode

```
double p=7,q=9;
double min = myMin(q,p);
```

```
      →  double min = q < p ? q : p
```

Der # Operator mit einem Variablen Namen wird durch einen string mit Inhalt von str( ) ersetzt.

```
#define str(x) #x
….
cout << str(myText);
```

```
→  cout << “myText” ;
```

Der ## Operator verbindet 2 Argumente

```
#define verbinde(x,y) x##y
…..
verbinde(c,out)<<“Nützlich?“;
```

```
→  cout << “Nützlich?“;
```

gcc preprocessor Anweisungen

- Bedingtes Einsetzen von Macrotext in den laufenden Quelltext durch die preprocessor Anweisungen `#if , #ifdef , #ifndef , #elif , #endif`

```
#ifndef DEBUG
#define DEBUG 1
#endif
```

Parameter DEBUG wird definiert falls noch nicht vorhanden. DEBUG wird hier true oder false gesetzt.

```
#if DEBUG
#define DPRINT(x) (std::cout << (x) << endl)
#else
#define DPRINT(x)
#endif
```

Definition einer Funktion die das Argument ausgibt falls DEBUG gesetzt ist

Sonst ist die Funktion leer

```
…..
DPRINT(keys[jKeys]); → cout << keys[jKeys] << endl;
```

Im Quelltext erscheint nur mit DEBUG 1 output !

Für x können längere Ausdrücke gesetzt werden

Fortsetzung in der nächsten Zeile

```
#define D(x) do {std::cerr << x ;} \
        while(0); std::cerr << endl

D(key << " " << myVal);  → cout<<key<<" "<<myVal<<endl;
```

gcc preprocessor Anweisungen

- Es gibt intern gesetzte Parameter auf die man zugreifen kann

```
__DATE__     __TIME__     __cplusplus     __FUNCTION__
__LINE__     __FILE__     __VA_ARGS__
```

ISO Standard des compilers

- gcc option -D gibt Zugang zu den preprocessor macros

```
gcc -D DEBUG myfile.c -o myPgm
```

Schaltet das DEBUG flag ein

```
……...
#ifdef DEBUG
   std::cout << "Debug mode switched on" << "\n" ;
#else
   std::cout << "Production mode switched on"<<"\n" ;
#endif
```

macroTest.cc

C++ Programme mit vielen Macros sind schwer lesbar!