A fordító (compiler) és a fordítás

A fordítás művelete valójában három részműveletből tevődik össze:

1. előfeldolgozás (preprocessing)
2. fordítás (compiling)
3. összefűzés (linking)

Ebben a fejezetben megnézzük, hogyan is működik az első és utolsó művelet, mikor lépnek működésbe és hogyan tudjuk befolyásolni őket. A második, azaz magáról a fordítás műveletéről nem ejtünk szót itt, azon egyszerű okból, hogy a C kód, amit írunk, valójában ennek feldolgozási egységnek szól.

Előfeltétel: Legyen a helloworld programunk megírva, *printf* ismerete.

Rövidítés: A továbbiakban a fordítsuk le a programot és futtassuk a generált futtatható állomámyt a következő rövidítéssel értjük: ***C&R*** (compile and run).

Előfeldolgozás és makrók

Méltatlanul elhanyagolt témakör, pedig mint láthatjuk, a C nyelven írt szöveges állományból gépi kód készítésének egy fontos, nem kikerülhető lépése. Ennek oka, hogy alapvetően nincs hozzáadott értéke az algoritmusokhoz, a már elkészült gépi kód futása közben sincs haszna. Valamint a C++ nyelven belül a konstans kifejezések (*constexpr*) átveszik a helyüket.

Az előfeldolgozást a makrók segítségével befolyásoljuk, amik közül már az *include* makróval találkoztunk. Az előfeldolgozó két művelettel operál valójában: másolás és törlés.
A C kódtól megkülönböztetés legelterjedtebb módja, hogy csupa nagybetűvel írjuk a makró azonosítóját, valamint nem követjük a C kód tördelését, indentálását. Mi is ezt követjük itt.

Nézzük át a teljesség igénye nélkül a leggyakrabban használtakat.

#include <header> vagy #include "header"

> Ahogy a helloworld esetén láttuk, ez a makró másolja be a fejlécállomány tartalmát a kódunkba. Akár a standard könyvtár, vagy akár saját készítésű header file-kat tudunk másoltatni a preprocessorral. A standard könyvtár esetben a <> használatával az operációs rendszerünktől függően a fordító installálásakor elhelyezett helyen keresi a preprocessor a header-t, míg utóbbi esetben az aktuális munkakönyvtárunkban keresi, ahol megadhatunk abszolút vagy relatív elérési útvonalat is. Bővebben a modularizációnál, amikor a kódunkat szétdaraboljuk.

#define AZONOSÍTÓ érték / kifejezés

> Ezzel egy konstans értéket vagy kifejezést tudunk definiálni, melyet a preprocessor az AZONOSÍTÓ helyére fog bemásolni a kódunkon belül.
>
> **Feladatok**:
> - Nyissuk meg a helloworld.c-t, írjuk be a következő sort, módosítsuk a *printf*-t, végül *C&R*:
>
> *#define HW "Hello World\n"*

printf(HW);

Megjegyzés: A fordítási alegység számára itt nincs különbség a sima helloworld kódunk és a makrós verzió között, hiszen mire a második fordítási lépéshez érünk, a preprocessor már bemásolta a *printf* zárójelei közé a karakterláncot.

- Másoljuk a *#define HW* sort különböző helyekre: a *main* függvény elé, a *printf* elé és *main* függvény törzsébe, a *printf* függvény elé, majd figyeljük meg a különbségeket. Változást csak utóbbi esetben kapunk, hiszen egy olyan makrót akarunk használni, ami még nincs.

- Írjuk a *printf* hasában lévő *HW* azonosítót *"HW"* formába. A preprocessor így nem ismeri fel, hogy ez egy makró, számára ez sima karakterlánc.

- Módosítsuk a *HW* makrónkat a következőre és a *printf*-t is, majd *C&R*:

  #define HW "Hello World%c"
  printf(HW, '\n');

  Megjegyzés: A preprocessor számára a *%c* karakter nem okoz gondot, az szintén egy az egyben bemásolja a célhelyre.

- Definiáljunk egy konstans számot, irassuk ki *printf*-fel:

  #define PI 3.14
  printf("pi = %f\n", PI);

  Megjegyzés: A π a *math.h* standard könyvtári fejlécben ugyanígy van definiálva.

#undef AZONOSÍTÓ

Az AZONOSÍTÓ nevű makrót törli, az a kód **további** részleteiben nem elérhető.

Feladat: A PI-t definiáló makró mögé tegyük be az alábbi sort:

#undef PI

Fordítsuk a kódot és vegyük észre a fordítási hibát.

#define AZONOSÍTÓ(argumentumok) utasítás

Nem csak konstansokat és karakterláncokat tudunk megadni, de utasításokat is.

Feladatok:

- Definiáljuk két szám összeadását a helloworld.c-ben mint makró, majd egy *printf*-ben használjuk:

  #define SUM(a, b) a + b
  printf("a + b = %d\n", SUM(1, 2));

Megjegyzés: A második fordítási művelet számára ez így néz ki:

printf("a + b = %d\n", 1 + 2);

- Módosítsuk a *printf*-t a következőképp:

  *printf("a + b = %d\n", SUM(1, 2) * SUM(1, 2));*

  Megjegyzés: A kapott érték nem a várt 9, hanem 5, ami nem meglepő, ha a másolást megtesszük, amit a preprocessor is:

  *printf("a + b = %d\n", 1 + 2 * 1 + 2);*

  Az ilyen false számolások elkerülésére javasolt a zárójelek használata, mely segítségével a matematikai műveletek precedenciája tartható.

- Módosítsuk a *SUM* makrót, a *printf* dupla *SUM* alakján pedig ne, *C&R* és ellenőrízzük, hogy valóban 9-t kapunk eredményül:

  #define SUM(a, b) (a + b)

- Az ilyen utasításmakrók egymásba is ágyazhatóak, azaz makrót meg lehet hívni másik makróból, ha már definiáltuk természetesen. Definiáljuk a *POWSUM* makrót a *SUM* makró után, majd irassunk ki valamit a segítségével:

  *#define POWSUM(a, b) (SUM(a, b) * SUM(a, b))*

  *printf("(a + b) * (a + b) = %d\n", POWSUM(1, 2));*

  Megjegyzés*:*Az előfeldolgozás után a *printf* kinézete:
  *printf("(a + b) * (a + b) = %d\n", ((1 + 2) * (1 + 2)));*
  Próbáljuk ki azt az esetet, amikor a *POWSUM* makró a *SUM* makró előtt van.

#ifdef AZONOSÍTÓ *vagy* #ifndef AZONOSÍTÓ
#else
#endif

Megvizsgálja, hogy az AZONOSÍTÓ makró létezik-e vagy sem, ha igen (vagy nem), akkor a makró törzsében lévő rész a kódba kerül, ellenkező esetben nem lesz figyelembe véve. Az *else* ág az ellenkező esetekben kerül a kódba és a második fordítási művelethez továbbítva. Az *#elif* pedig további elágazási pontokat tudnak jelenteni, ha nem teljesül az *#ifdef* ág.

Feladatok:

- Nyissuk meg a helloworld.c kódunkat. Tegyük a Hello World kiírását végző *printf* utasítást egy ifdef makróba:

  #ifdef PRINT
  printf("Hello World!\n");
  #endif

Megjegyzés: A Hello World kiírása nem történik meg, egyszerűen mert nincs *PRINT* makró definiálva.

- Definiáljuk a *main* függvény elé a *PRINT* makrót, *C&R*. Látjuk, hogy a kiírás megtörténik.

  Megjegyzés: A mai modernebb fejlesztői környezetek felismerik azon kódrészleteket, amik ilyen, nemdefiniált makrók között találhatóak és ki is szürkítik azokat. Ezek a kódrészletek nem kerülnek bele a második fordítási alműveletbe.

- Egészítsük ki a helloworld kódot az előző *#ifdef* makró és *printf* utasítás után a következő sorral, *C&R*:

  #else
  printf("Goodbye World!\n");

  Megjegyzés: Ahogy várjuk, a Goodbye World kiírása nem történik meg. Amennyiben kommenteljük vagy töröljük a *PRINT* makrót, úgy a Goodbye World írodik ki, a Hello World nem.

- Módosítsuk az *#ifdef* makrót *#ifndef* makróra, majd nézzük meg, hogy mi történik ekkor futáskor, valamint ha kommenteljük/töröljük a *PRINT* define-t.

```
#if KIFEJEZÉS
#elif MÁSIKKIFEJEZÉS1
#elif MÁSIKKIFEJEZÉS2
...
#else
#endif
```

Hasonlóan az *#ifdef* makróhoz, de itt kifejezéseket is adhatunk meg, ahol felhasználhatjuk a már bevezetett makrókat is.

További referencia: https://gcc.gnu.org/onlinedocs/gcc-3.0.1/cpp_3.html

Összefűzés és modularizálás

TO DO.