Misc, diverse

Funcții cu număr variabil de parametri

- În anumite situații, este util să declarăm funcții care primesc un număr variabil de parametri
- De ex:

```
int printf(char *fmt, ...)
```

- Funția printf primește ca parametri un șir de caractere și apoi un număr variabil de parametri adiționali (0, 1, oricâți...)

<stdarg.h>

- Cum putem să accesăm lista/vectorul de parametri variabili dacă aceasta nici măcar nu are un nume în definiția funcției?

- Folosim header-ul standard <stdarg.h>
- Acesta oferă 4 primitive (macro/funcții) pentru a accesa lista variabilă de parametri
  - va_arg, va_copy (din C99), va_end, va_start
  - Implementarea acestora diferă în funcție de implementarea librăriei de C folosită

va_list

- Tip care conține informațiile necesare pentru a lucra cu o listă variabilă de parametri
- Lista se inițilizează cu apelul va_start()
- Dacă o listă a fost inițializată cu va_start(), trebuie să fie invocat si va_end() pentru aceeași listă

Primitive pentru liste variabile de parametri

- **void va_start(va_list** *ap*, *last*);
  type **va_arg(va_list** *ap*, *type*);
  void va_end(va_list *ap*);
  void va_copy(va_list *dest*, **va_list** *src*);

va_start()

- void va_start (va_list ap, paramN);

- Inițilizează lista variabilă de parametri (va_list) *ap* către primul parametru adițional după *paramN*
- *ap* trebuie să fie o listă de parametri neinițializată sau care a fost deja “închisă” cu *va_end()*

- Orice funcție care apelează *va_start()*, trebuie să apeleze și *va_end()*

va_arg()

- type va_arg (va_list ap, type)

- Întoarce argumentul curent din lista variabilă de parametri *ap*
- Tipul argumentului curent este dat de către al doilea parametru, la fel și tipul valorii returnate
- Fiecare apel modifică starea internă a variabilei *ap* astfel încât, al următorul apel, să fie întors argumentul următor din listă
- *Important!* Nu se face nici o verificare pentru a vedea dacă argumetul întors este ultimul sau nu! Acest lucru trebuie făcut de către programator!

va_end()

- void va_end (va_list ap);

- Efectuează operațiile necesare pentru ca o funcție care a folosit o listă variabilă de parametri (va_list) să se termine corect

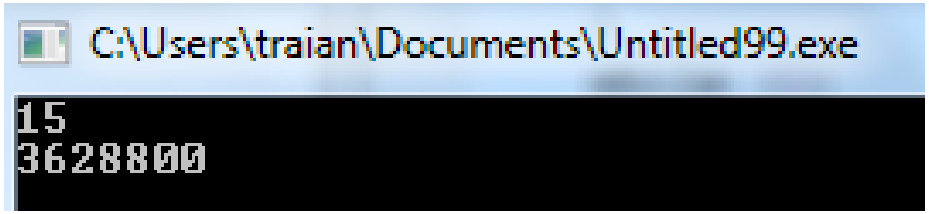
Exemplu

```
int op_int(char op, ...)
{
    int result;
    int i;
    va_list ap;

    va_start(ap, op);
    switch (op)
    {
        case '+':
            result = 0;
            break;
        case '*':
            result = 1;
            break;
    }
    while (1)
      {
        i = va_arg(ap, int);
        if (i < 0)
            break;
        switch (op)
        {
            case '+':
                result += i;
                break;
            case '*':
                result *= i;
                break;
        }
      }
    va_end(ap);
    return result;
}
```

Exemplu

```
int main()
{
    int result;
    result = op_int('+', 1, 2, 3, 4, 5, -1);
    printf("%d\n", result);
    result = op_int('*', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, -1);
    printf("%d\n", result);
    getch();
    return 0;
}
```



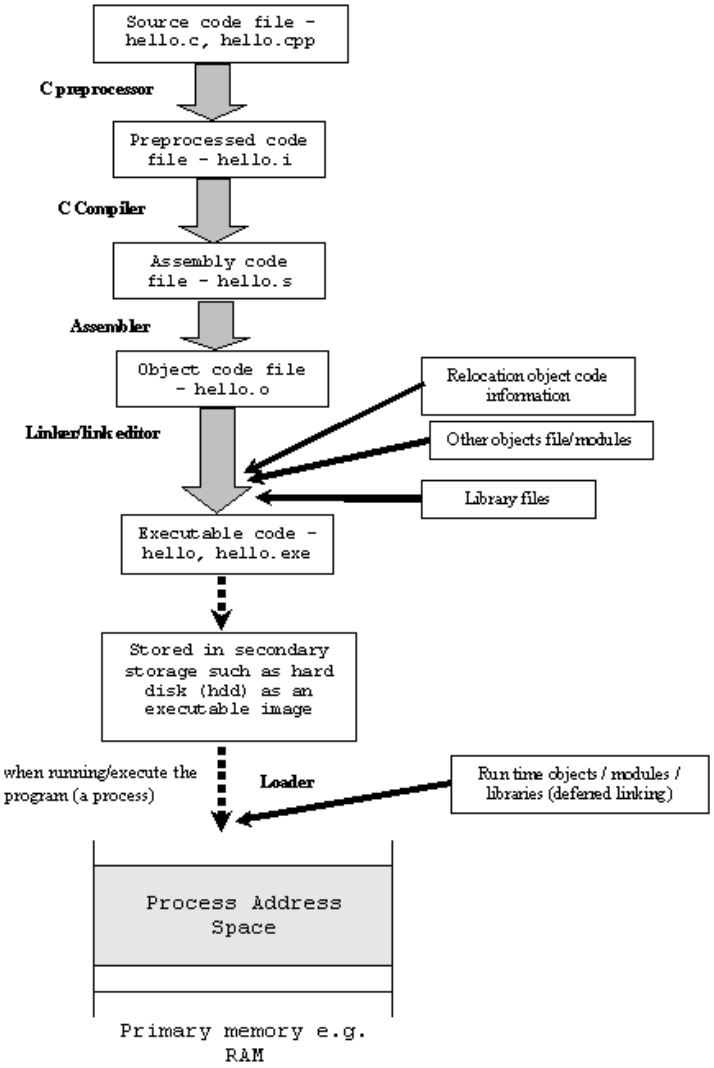
Observații

- Funcțiile cu număr variabil de argumente trebuie să primească cel puțin un argument cunoscut/numit
- Drept urmare, in C nu este corectă următoarea definiție/semnătură de funcție

```
char *wrong(...);
```

- Ce se întâmplă dacă tipul trimis de către programator lui va_arg() nu este corect / nu se potrivește cu tipul real al parametrului curent din listă?
- Cum se poate trimite lista variabilă de parametri unei alte funcții apelată de către o funcție cu număr variabil de parametri?

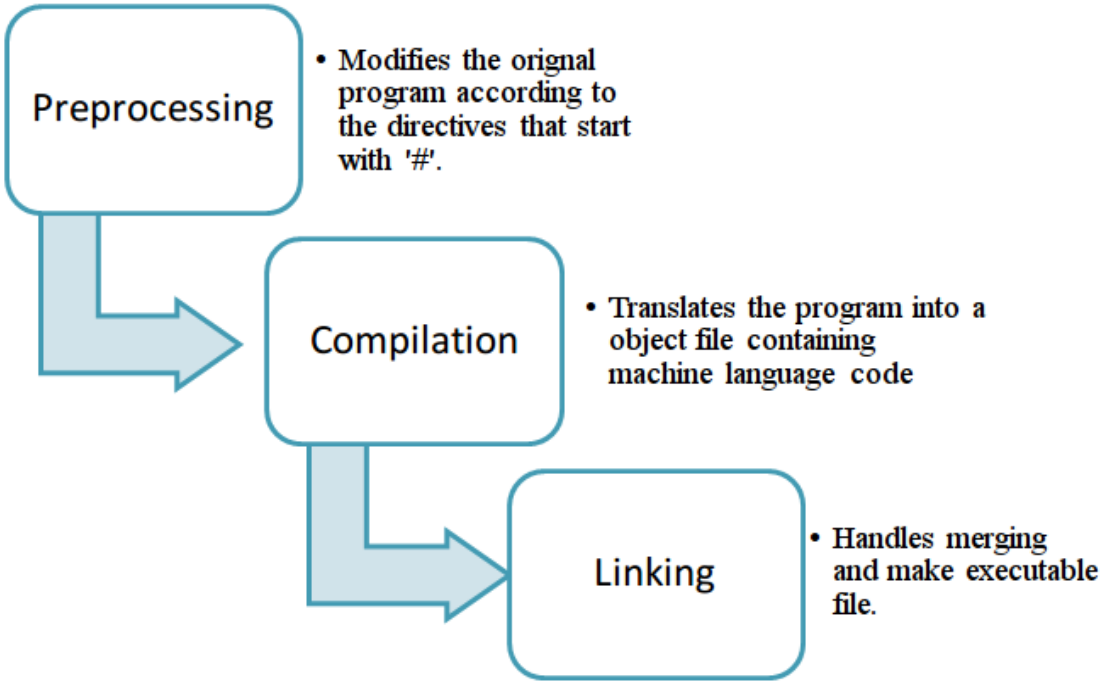
Compilarea programelor C



- http://www.tenouk.com/ModuleW_files/ccompilerlinker001.png

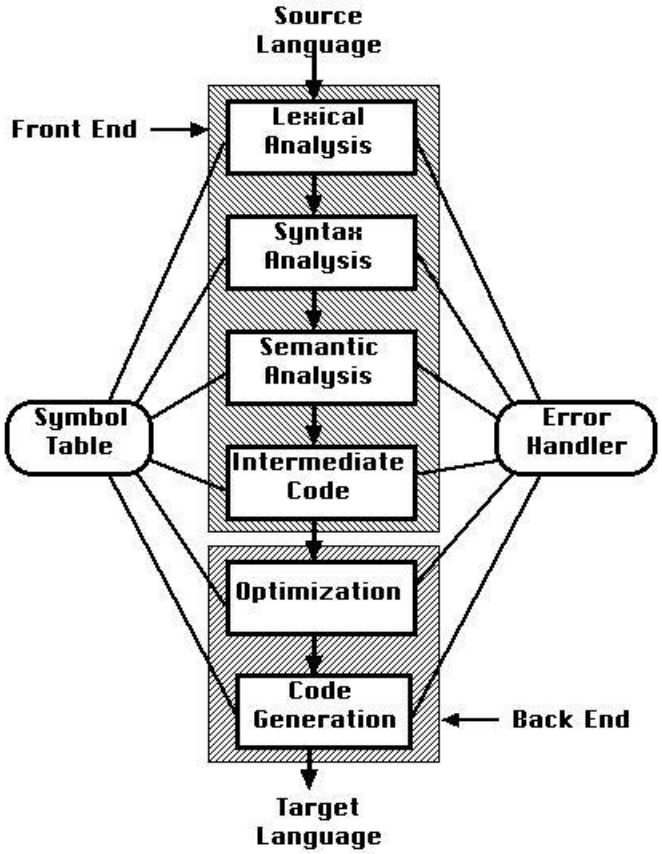
Pași compilare programe C

- http://www.cplusplus.com/articles/2v07M4Gy/Selection_101.png



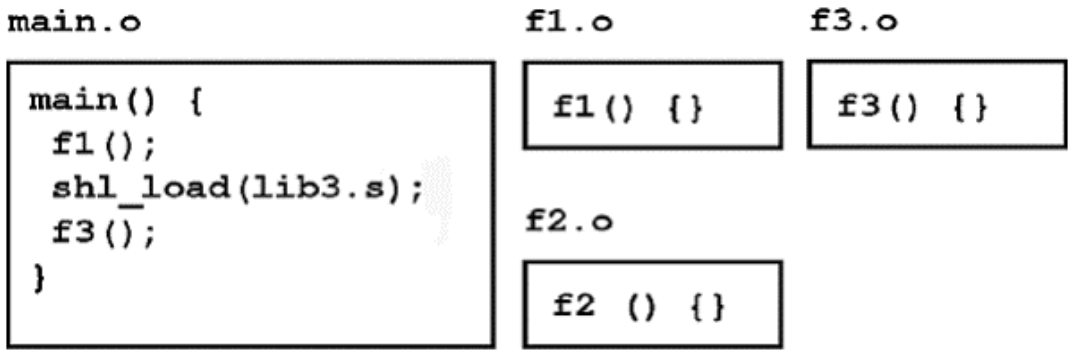
Compilarea este un proces complex!

- Există o materie în anul 4 care se ocupă doar de proiectarea compilatoarelor!



Linking

- http://stackoverflow.com/questions/6264249/how-does-the-compilation-linking-process-work
- Link-editarea (linking) este procesul prin care mai multe fișiere obiect sunt legate/combinate pentru a crea fișierul executabil
- Leagă referințele către simboluri (variabile sau funcții) nedefinite într-un fișier obiect către adresele corecte. Aceste adrese sunt definite în alte fișiere obiect sau biblioteci
- Un exemplu este legarea printf() cu definiția ei din biblioteca stdlib (care se află în libc sau glibc sau Microsoft C Runtime Library, etc.)
- Pot apare erori precum definiții lipsă sau definiții duplicate



Preprocesare în C

- Preprocesarea este primul pas din compilarea unui program

- C oferă anumite facilități prin intermediul preprocesatorului
  - Includerea de fișiere #include
  - Inlocuirea unui token cu o secvență dată de caractere (definire constante sau macro-uri simple) #define
  - Compilare condiționată
  - Definire macro-uri cu argumente

Includere de fișiere

- #include "*filename*"
- #include <*filename*>

- De obicei, se includ fișiere cu declarații (fișiere *header*)
- De obicei, incluziunile se fac la începutul unei surse

- Însă acestea nu sunt reguli, poate fi inclus orice fișier într-o sursă C, în orice loc. Important este ca fișierul rezultat să compileze!

Definire macro-uri simple

- #define *name replacement_text*

- Folosit pentru definirea de macro-uri (*macro substitution*)
- Cele mai simple sunt “constantele”
  - #define PI 3.14
- Pot fi definite însă și alte macro-uri simple
  - #define forever for (;;) /* infinite loop */

Definire macro-uri cu argumente

- Pot fi definite și macro-uri mai complexe, cu “argumente”
- Acestea nu sunt funcții!
  - #define max(A, B) ((A) > (B) ? (A) : (B))

- Instrucțiunea
  - x = max(p+q, r+s);
- Va fi inlocuită la preprocesare cu instrucțiunea
  - x = ((p+q) > (r+s) ? (p+q) : (r+s));

Definire macro-uri cu argumente

- Pot duce la greșeli
- De exemplu, instructiunea
  - max(i++, j++) /* WRONG */
- Se va inlocui cu
  - ((i++) > (j++)) ? (i++) : (j++)

- Unele definiții de macro-uri pot duce mai ușor la erori:
  - #define square(x) x * x /* WRONG */

Compilare condiționată

```
#if !defined(HDR)
#define HDR
/* contents of hdr.h go here */
#endif
```

- #ifndef HDR
- #define HDR
- /* contents of hdr.h go here */
- #endif

```
#if SYSTEM == SYSV
        #define HDR "sysv.h"
#elif SYSTEM == BSD
        #define HDR "bsd.h"
#elif SYSTEM == MSDOS
        #define HDR "msdos.h"
#else
        #define HDR "default.h"
#endif
#include HDR
```