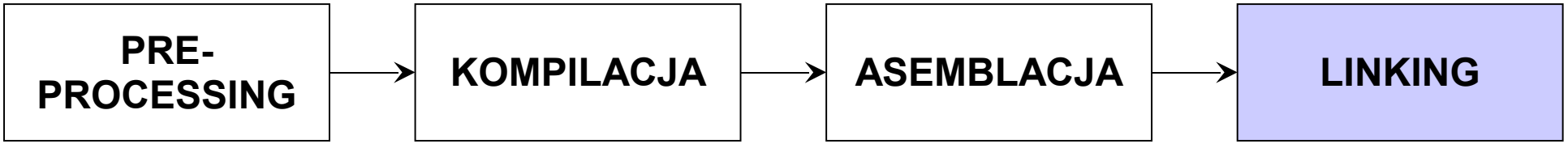
Konsolidacja (linking)

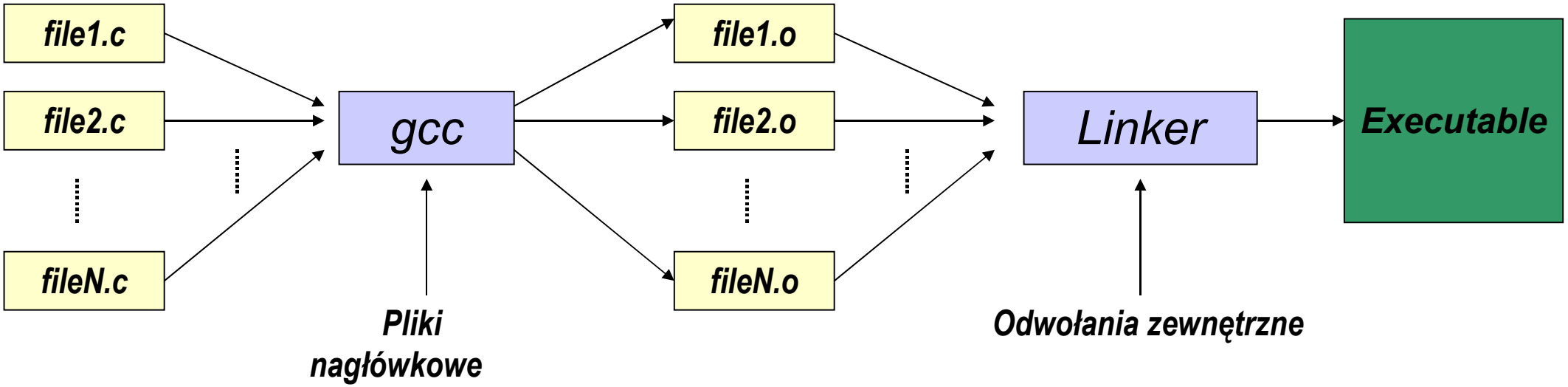
- Ostatni etap budowania programu



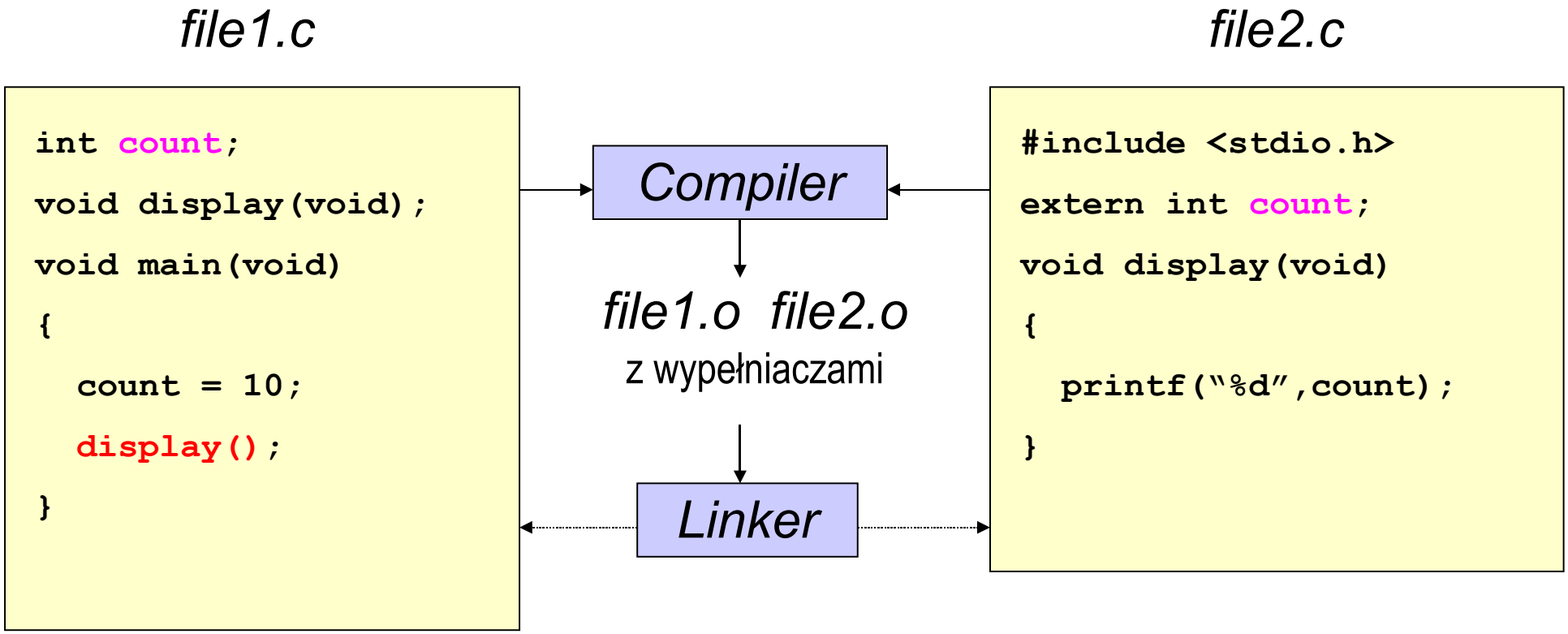
- Łączenie osobnych fragmentów kodu maszynowego w plik wykonywalny
- Wykonywana przez linker
  - ld w systemie Unix
- Często w sposób przezroczysty dla programisty (gcc samo się tym zajmuje)

Konsolidacje obejmuje...

- Łączenie szeregu plików pośrednich (plików `.o` odpowiadających plikom `.c` files) w jeden plik
- Rozwiązywanie zewnętrznych odwołań do zmiennych i funkcji
- Generacja pliku wykonywalnego (jeżeli nie ma błędów)



Rozwiązywanie odwołań zewnętrznych



- **`file1.o`** ma wypełniacz dla **`display()`**
- **`file2.o`** ma wypełniacz dla **`count`**
- pliki pośrednie są relokowalne
  - adresy są relatywnymi przesunięciami od początku pliku

Biblioteki

- Definicja:
  - plik zawierający funkcje które mogą być wykorzystywane przez program w języku C
- Cel:
  - łatwy dostęp do wielokrotnie używanych funkcji
  - promocja modularności kodu i ponownego użycia
  - redukcja rozmiaru źródeł i pliku wykonywalnego

Biblioteki

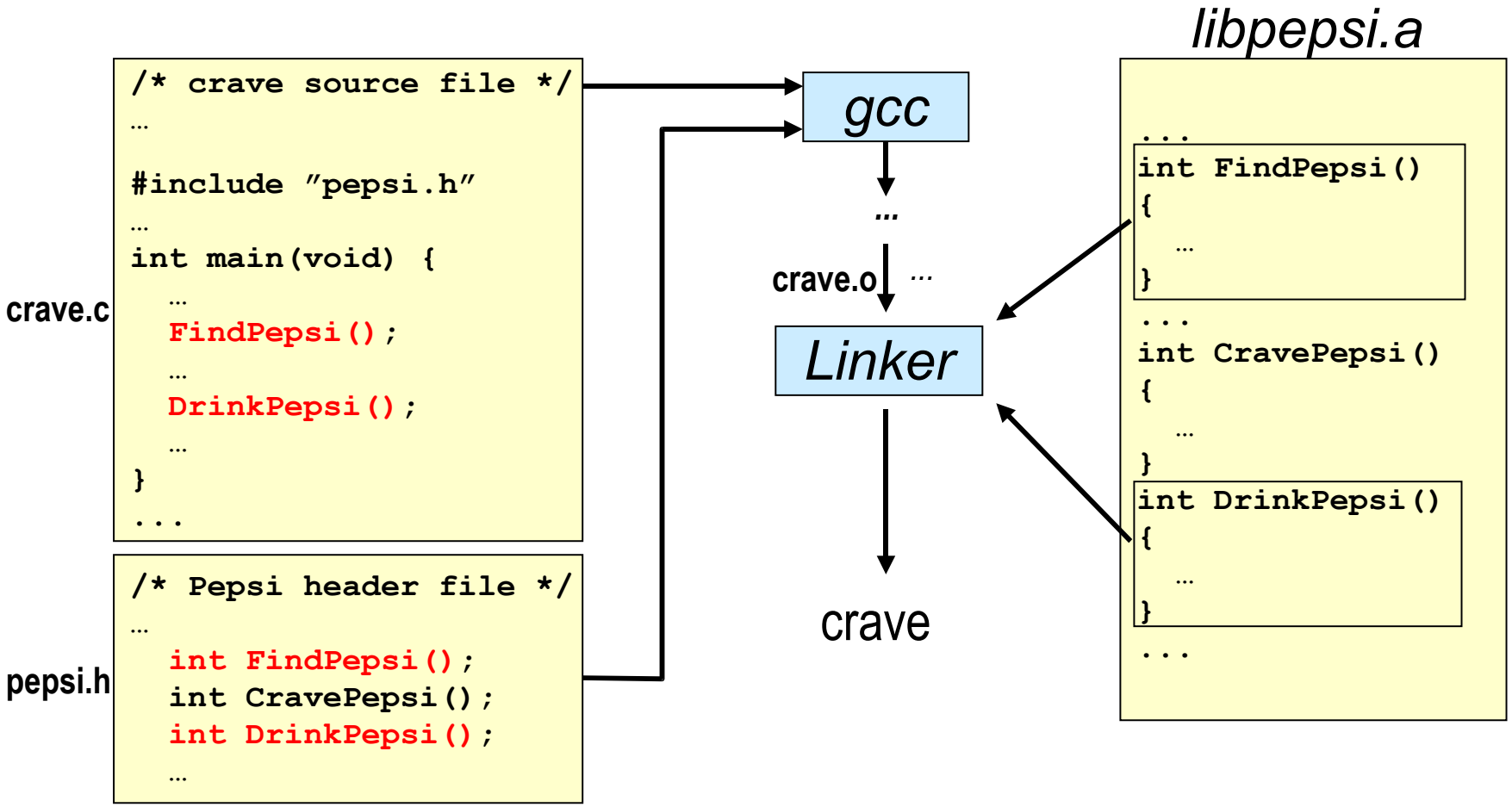
- Statyczne (*Archiwa*)
  - **`libname.a`** w systemie Unix; **`name.lib`** w systemie DOS/Windows
  - Tylko moduły do których są odwołania są dołączane podczas kompilacji
    - przeciwnie niż w przypadku plików **`.o`**
  - Linker kopiuje funkcje z biblioteki do pliku wykonywalnego
  - Uaktualnienie biblioteki wymaga rekompilacji programu

Biblioteki

- Dynamiczne (*Shared Object* or *Dynamic Link Library*)
  - **`libname.so`** w systemie Unix; **`name.dll`** w systemie DOS/Windows
  - Kod nie kopiowany do pliku wykonywalnego
    - Ładowany do pamięci w momencie uruchomienia programu
  - Mniejszy rozmiar pliku wykonywalnego
  - Można uaktualnić bibliotekę bez rekompilacji programu
  - Wada: nieco wolniejsze uruchomienie programu

Biblioteki

- Dołączenie biblioteki statycznej



Biblioteki standardowe

- Standard ANSI C definiuje zestaw funkcji które muszą być wspierane przez kompilatory zgodne ze standardem

- Standardowe pliki nagłówkowe ANSI zawierają prototypy funkcji bibliotecznych:
  stdio.h, stdlib.h, string.h, time.h, signal.h, math.h, …

- Standardowe biblioteki dołączane automatycznie podczas kompilacji

- Kompilatory mogą dostarczać dodatkowe biblioteki spoza standardu ANSI (np. graficzne)

- Unix: biblioteki standardowe często w katalogach /lib i /usr/lib

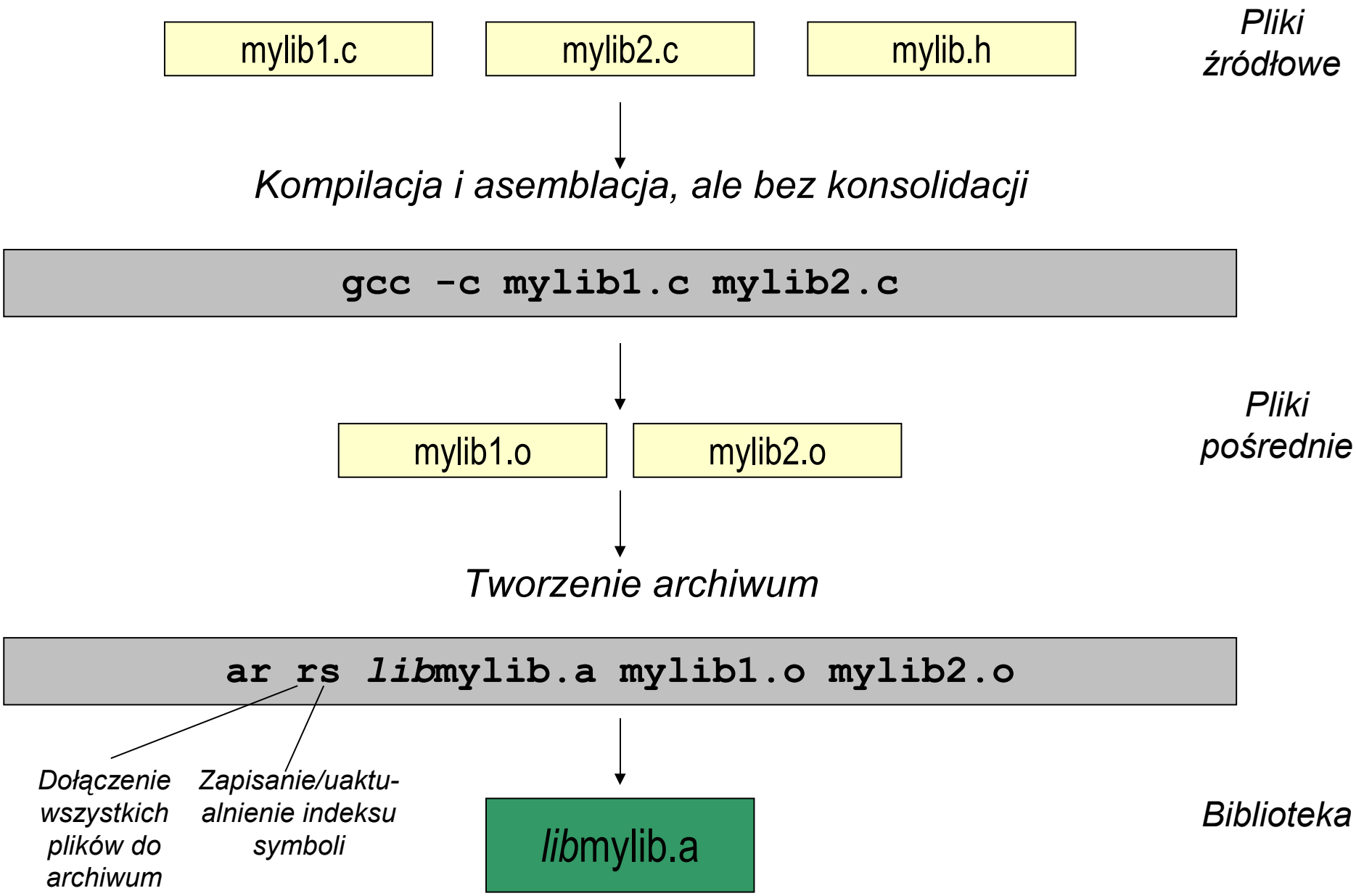
Tworzenie własnych bibliotek

- Dlaczego:
  - Kiedy mamy funkcje używane przez wiele różnych programów
  - Kiedy chcemy udostępnić nasze funkcje innym programistom

Tworzenie własnych bibliotek

- Jak:
  - gromadzimy funkcje w jednym lub wielu plikach .c
  - nie definiujemy funkcji **`main()`**
  - umieszczamy prototypy w pliku nagłówkowym
  - jeżeli używamy zmiennych globalnych niech bedą to zmienne statyczne
  - tworzymy bibliotekę przy pomocy polecenia **`ar`** (Unix)
- **`ar`**
  - narzędzie do tworzenia, modyfikacji, ekstrakcji z archiwum
  - archiwum = zaindeksowana kolekcja plików pośrednich
  - utrzymuje indeks symboli (funkcji, zmiennych) zdefiniowanych w plikach pośrednich dla szybszej pracy

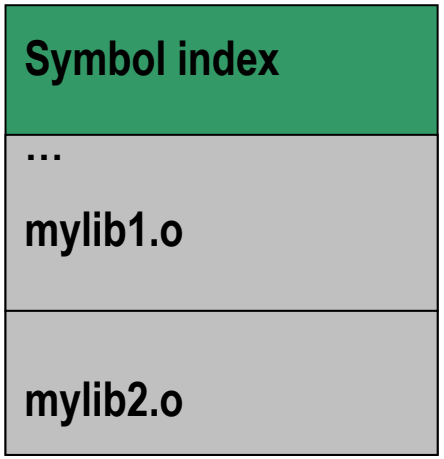
Używanie `ar`


mylib1.c
mylib2.c
mylib.h
Pliki źródłowe
Kompilacja i asemblacja, ale bez konsolidacji
gcc -c mylib1.c mylib2.c
mylib1.o
mylib2.o
Pliki pośrednie
Tworzenie archiwum
ar rs libmylib.a mylib1.o mylib2.o
Dołączenie wszystkich plików do archiwum
Zapisanie/uaktualnienie indeksu symboli
libmylib.a
Biblioteka

Używanie `ar`

- **`ar operation[modifier] archive [list of files]`**
  (np. **`ar rs libmylib.a mylib1.o mylib2.o`**)
- Najczęstsze operacje
  - **`d`** : usuń moduły
  - **`p [list]`**: wypisz podane moduły na stdout
  - **`r`** : wstaw lub zastąp
  - **`t`** : wypisz tablicę modułów
  - **`x`** : wypakuj moduły
- Możliwe dodatkowe modyfikatory do powyższych opcji
  - np. **`rs`** : wstaw + uaktualnij indeks
  - **`tv`** : dołącz czas utworzenia pliku, właściciela itd.
  - **`rsu`** : wstaw tylko uaktualnione moduły i uaktualnij indeks

libmylib.a



Opcje bibliotek w gcc

- **`-lname`**
  - użyj biblioteki **`libname.a`**
    - np.: **`gcc -o plot main.o plot_line.o -lm`**

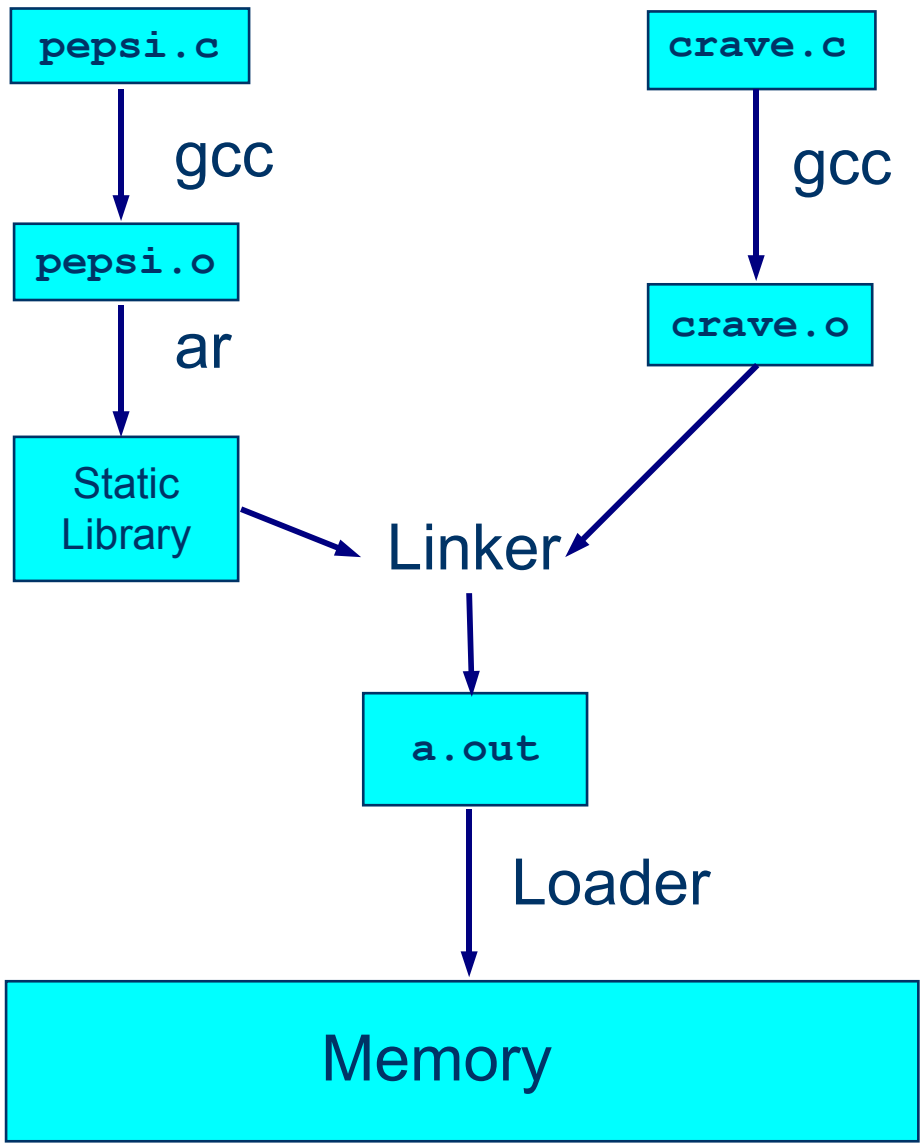
"użyj libm.a"

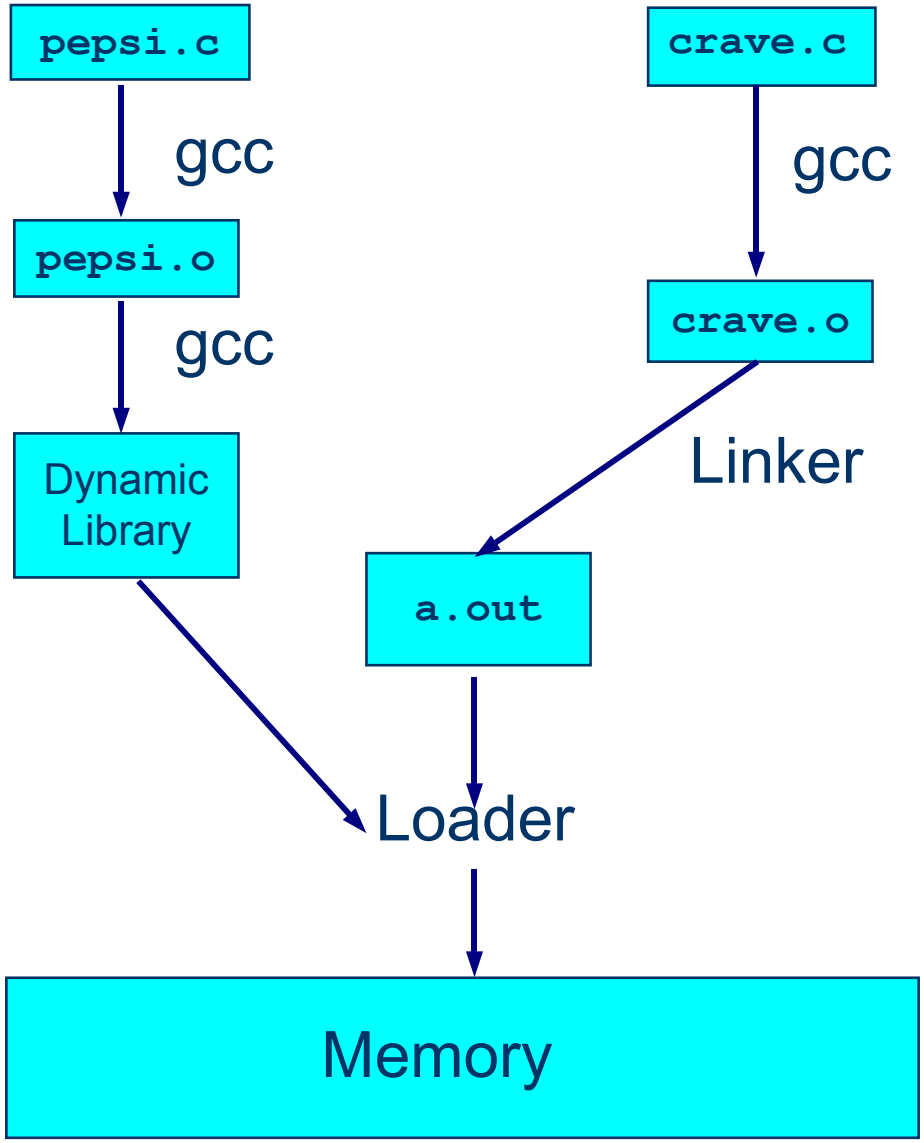
  - kolejność plików **`.o`** i **`-l`** jest istotna!
  - np.: **`gcc file1.c -lmylib file2.c`**
    może spowodować błąd linkera: **`Undefined symbol [file2.o]`**
    (W bibliotece wyszukiwane są tylko referencje dotąd nie rozwiązane)
    np.: **`gcc file1.c file2.c -lstop -lwalk`**
    może spowodować podobne błędy jeżeli if **`libwalk.a`** odwołuje się do funkcji w **`libstop.a`**

Opcje bibliotek w gcc

- **`-Ldir`**
  - dodaj dir do listy przeszukiwanych katalogów
  - np.: **`gcc -o crave -L/home/libs crave.c -lpepsi`**
  - Linker najpierw przeszukuje **`/home/libs`** a potem katalogi standardowe (**`/usr/lib`**, **`/lib`** w systemie Linux)
  - Pozwala na zastąpienie bibliotek standardowych własnymi wersjami
  - **`-nostdlib`** : nie używaj bibliotek standardowych
  - **`-static`** : uzywaj wyłącznie bibliotek statycznych
  - **`-shared`** : gdy to możliwe, używaj bibliotek dynamiczych
  - **`-u`*`symbol`*** : udawaj że symbol jest niezdefiniowany w celu wymuszenia załadowania modułu

Linking statyczny i dynamiczny





Tworzenie biblioteki dynamicznej

- Tworzenie biblioteki
  - `gcc -fPIC -Wall -pedantic -c findpepsi.c`
  - `gcc -fPIC -Wall -pedantic -c drinkpepsi.c`
  - `gcc -fPIC -Wall -pedantic -c cravepepsi.c`
  - `gcc -shared -Wl,-soname,libpepsi.so -o libpepsi.so findpepsi.o drinkpepsi.o cravepepsi.o`
- Użycie biblioteki
  - Linker poszukuje bibliotek dynamicznych w predefiniowanych katalogach systemowych
  - Aby linker poszukiwał biblioteki dynamicznej w innym katalogu, należy ustawić zmienną środowiska **`LD_LIBRARY_PATH`** na ten katalog
    - np. **`export LD_LIBRARY_PATH=.`**

Podsumowanie (konsolidacja + biblioteki)

- Konsolidacja
  - łączy kod z różnych plików w jeden plik wykonywalny
  - rozwiązuje zewnętrzne odwołania
- Biblioteki
  - dostarczają "usług" programom w miarę potrzeby
- Łącznie konsolidacja i biblioteki promują modularność i ponowne użycie kodu