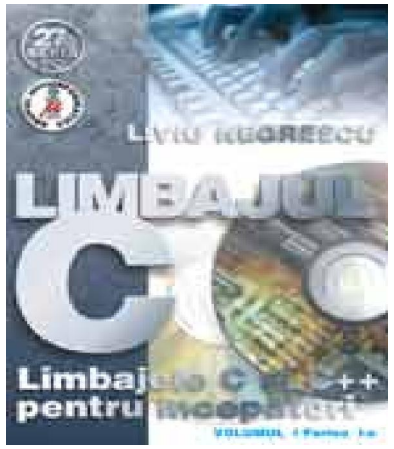
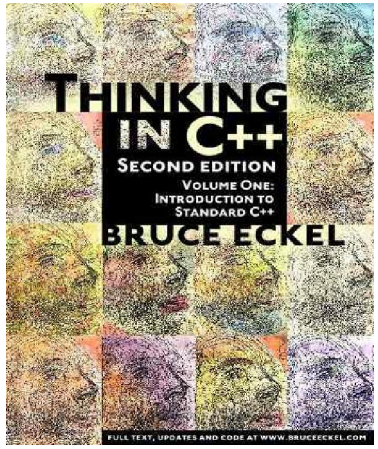
Introducere în Programarea Orientată Obiect (POO)

Bibliografie

- Liviu Negrescu, *Limbajele C şi C++ pentru începatori*, Vol. II, (editia XI), Editura Albastră, Cluj-Napoca, 2005.

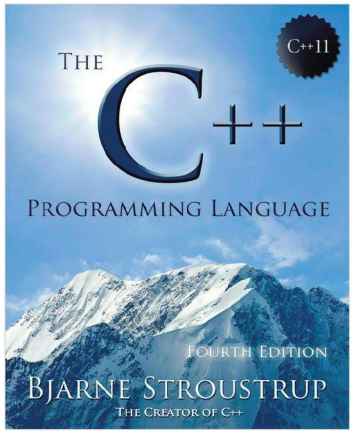


- Bruce Eckel, *Thinking in C++*, 2nd Edition, Prentice Hall 2000.
- M. Cosulschi, O. Mustafa, *Programare în C++. Concepte moderne şi aplicații*, PRO Universitaria, 2015.



Bibliografie

- Bjarne Stroustrup, *The C++ Programming Language*, Adisson-Wesley, 4th edition, 2013.
- J. Farrell, *Object-Oriented Programming using C++*, 4th edition, 2009.
- Y. Daniel Liang, *Introduction to Programming with C++*, Pearson, 2010.
- H.M. Deitel, P.J. Deitel, *C++: How to Program*, Prentice-Hall, 7th edition, 2010.



Detalii organizatorice

- Curs
  - Prezentarea teoriei generale a Programării Orientate Obiect.
- Laborator
  - Implementări practice ale teoriei prezentate la curs în C++.
  - Mediul de dezvoltare: *MS Visual Studio*, *DevC++*, *Code::Blocks*.
- Evaluare:
  - nota de laborator = NL
    - Teme
  - Examen final = NE;
  - nota finală: 50%NL + 50%NE;

Un prim program C++

```
#include <iostream>

using namespace std;

void schimb(int& x, int& y) {
  int z;
  z = x; x = y; y = z;
}

int main() {
  int a, b;

  cout << "a = "; cin >> a;
  cout << "b = "; cin >> b;

  schimb(a, b);

  cout << "a = " << a << "\nb = " << b << endl;

  return 0;
}
```

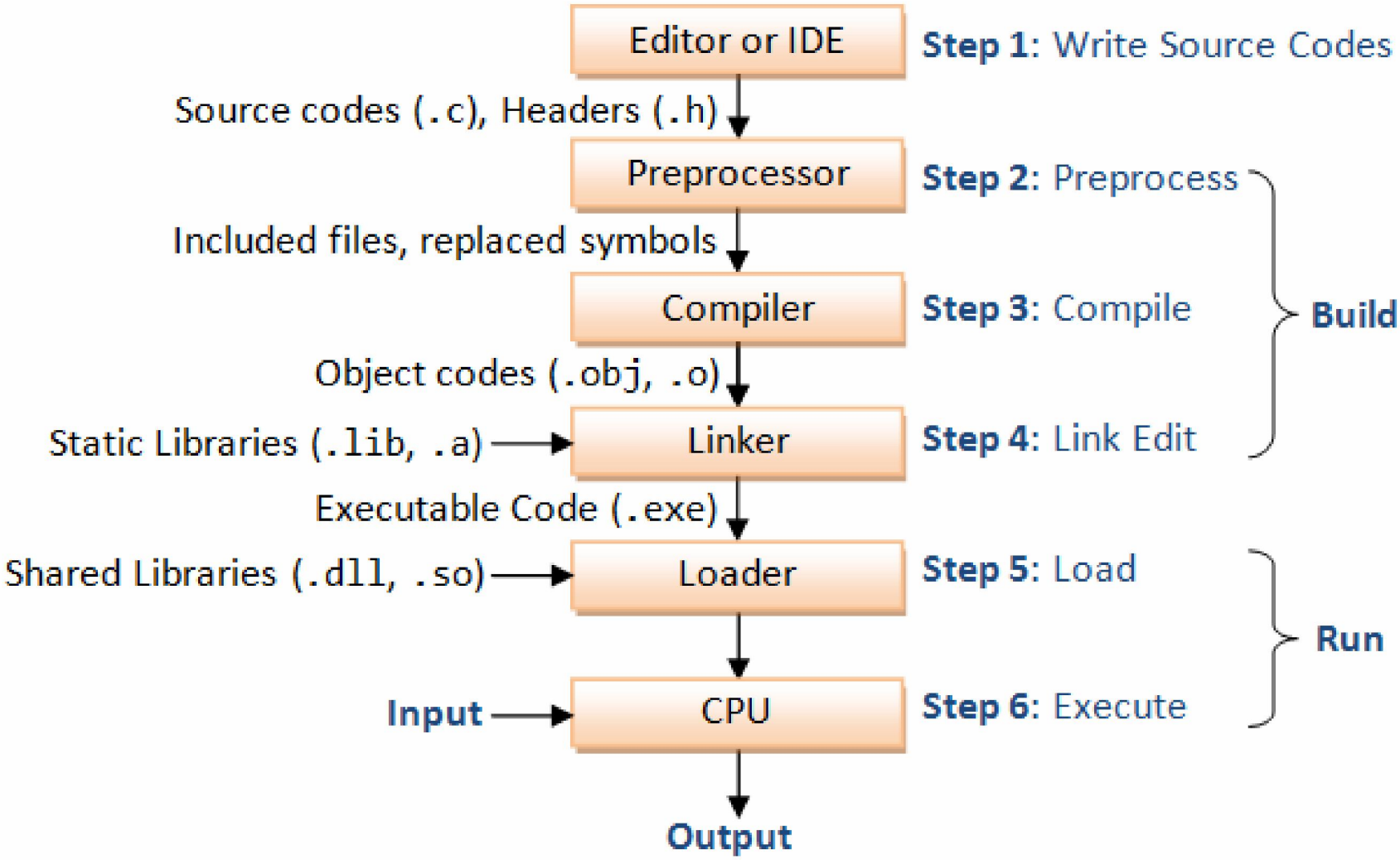
Limbajul C++

- limbaj folosit pe scară largă, atât în industrie, cât şi în domeniul academic (`https://www.tiobe.com/tiobe-index/`).
- este un limbaj de programare de nivel înalt, compilat, standardizat, aflat într-un proces de evoluție.
- multe dintre componentele critice ale platformelor Youtube, Google, Facebook, Amazon, Twitter, aplicații Microsoft, Adobe, sisteme de baze de date, sisteme fizice (mașini, roboți, rachete) sunt scrise în C++.
- unul dintre cele mai populare limbaje de programare.

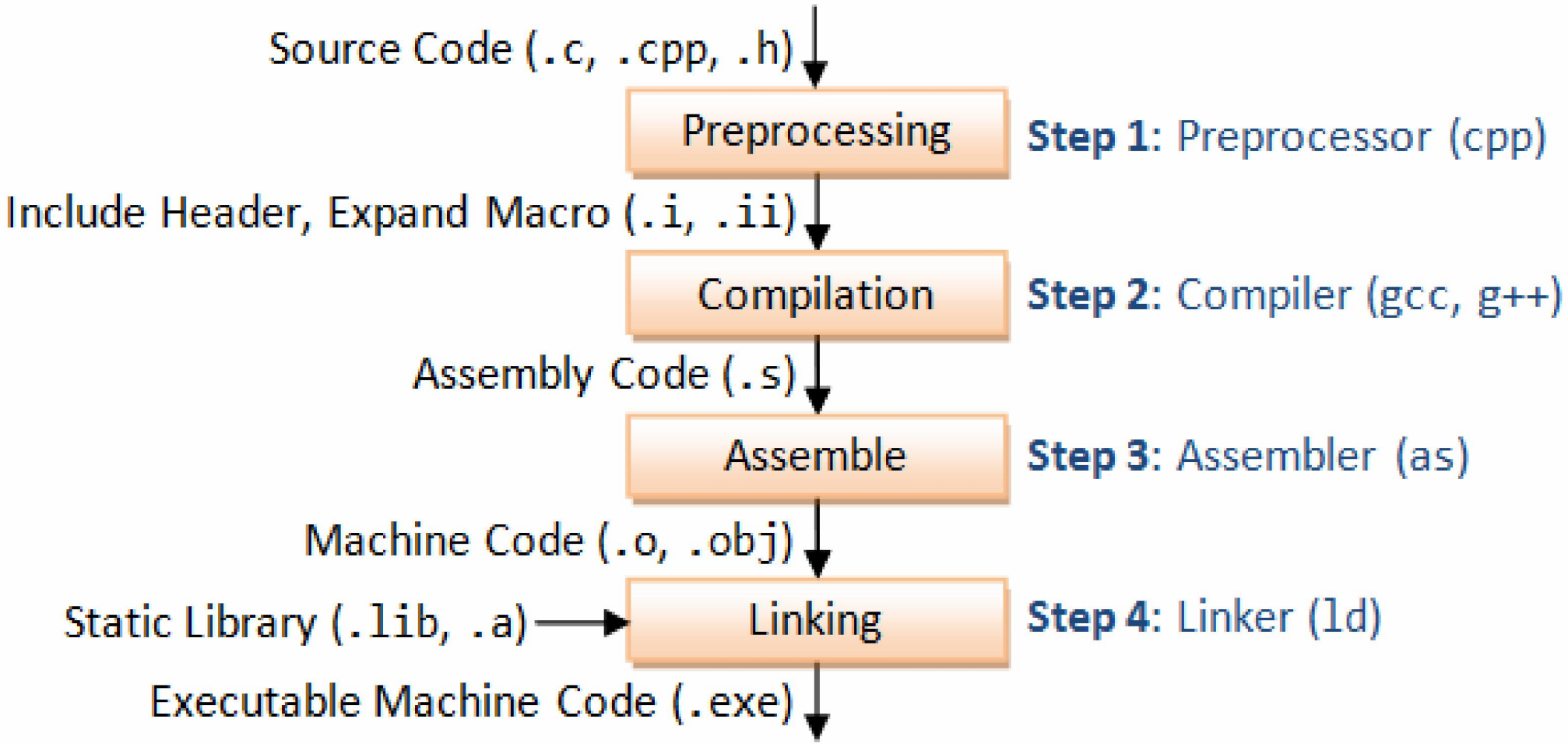
Istoric C++

- 1979 - Bjarne Stroustrup dezvoltă *C with Classes*.
- 1982 - Stroustrup lucrează la o versiune îmbunătățită a *C with Classes* denumită C++.
- 1985 – publică *The C++ Programming Language*, 1st edition.
- 1989 – este lansat C++ 2.0.
- 1992 – STL este implementată în C++.
- 1998 – C++98 (ISO/IEC 14882:1998) - RTTI (dynamic_cast, typeid), operatori de cast, mutable, bool, declarații în condiții, instanțieri de șabloane, șabloane de membri, export.
- 2003 – C++03
- 2011 – C++11
- 2017 – C++17
- 2020 – C++20.

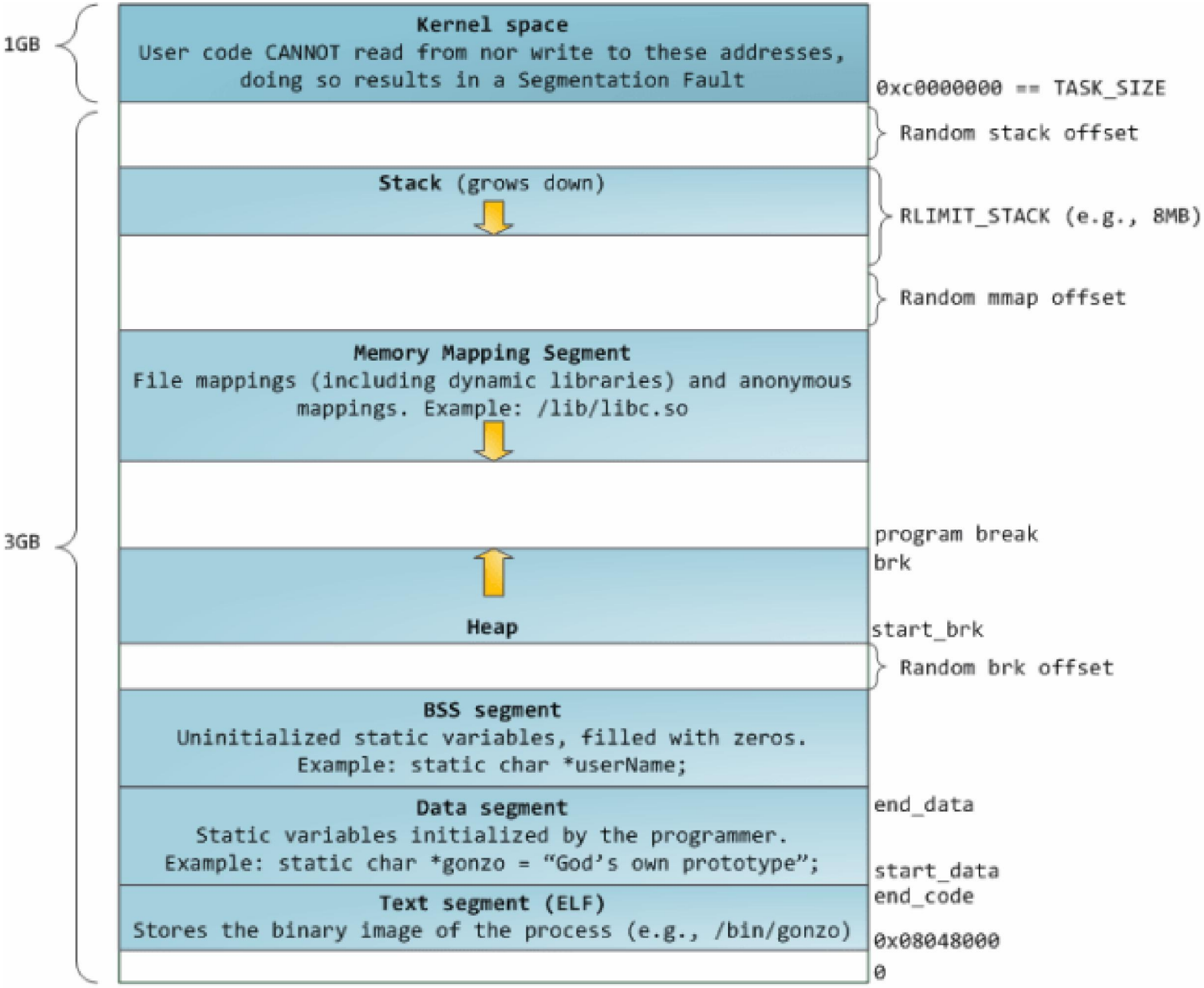
Dezvoltarea unui program în limbajul C/C++



Etape la compilarea unui program C/C++ cu gcc



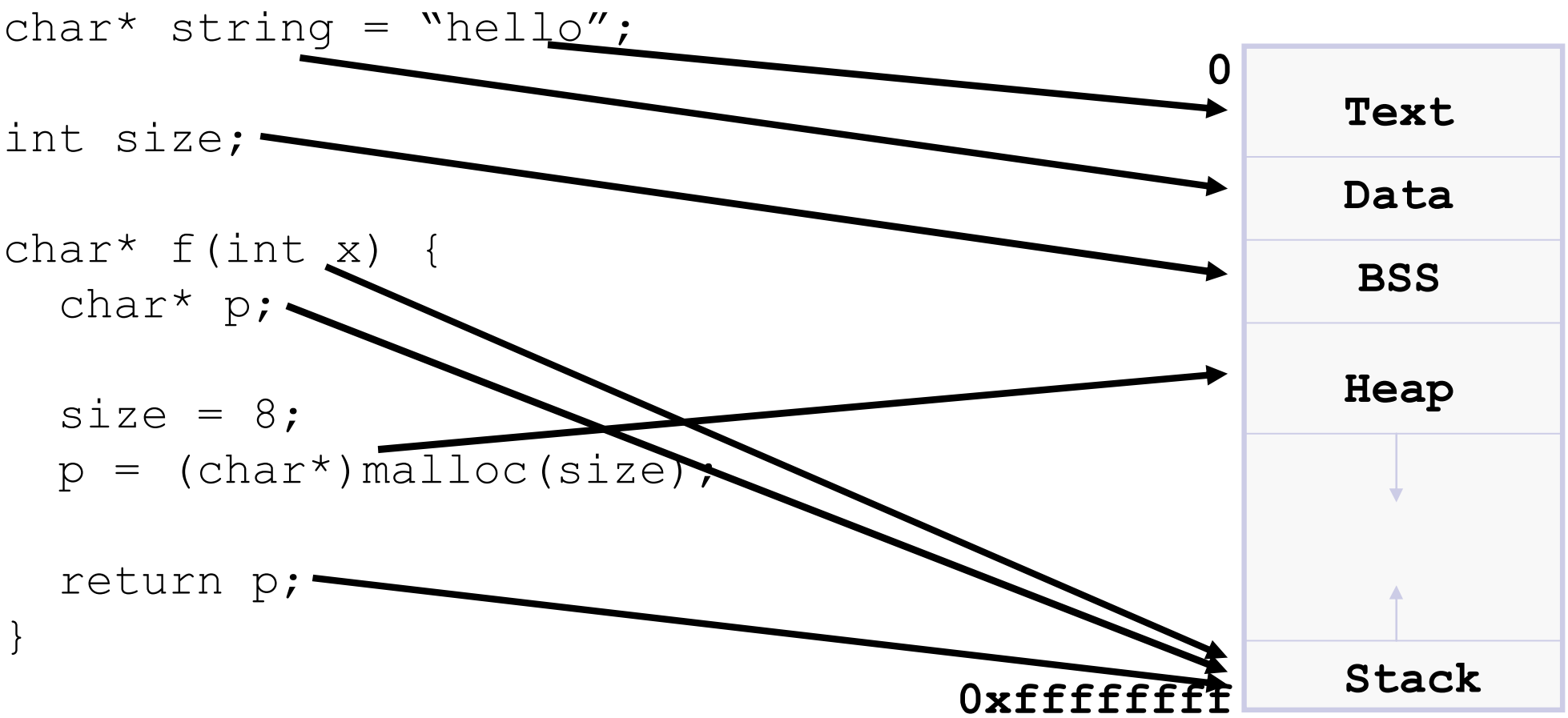
Structura memoriei unui program C/C++



Structura memoriei unui program C

- **Text** – codul programului, librării partajate şi constante.
- **Data** – variabile globale şi variabile locale statice iniţializate.
- **BSS** – variabile globale şi variabile locale statice neiniţializate.
- **Heap** – memoria alocată dinamic (memoria alocată în timpul rulării programului, de exemplu cu `malloc()`).
- **Stack** – oferă memorie temporară pe durata de viaţă a unei funcţii sau a unui bloc; păstrează parametrii unei funcţii şi variabilele locale.

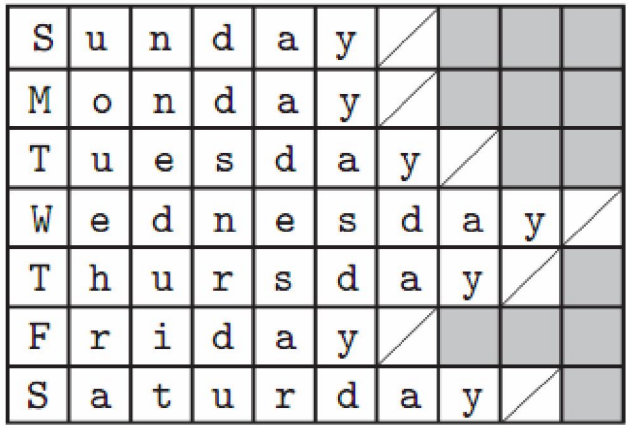
Structura memoriei unui program C

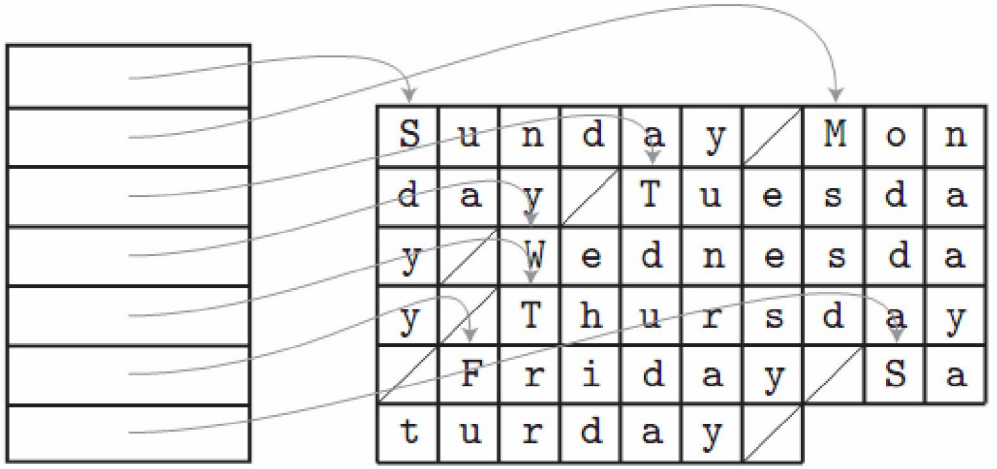


Structura memoriei unui program C

```
char days[][10] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```

```
char *days[] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```





Ce este Programarea Orientată Obiect?

- **Programarea Orientată Obiect (POO)** este o metodă de proiectare şi implementare în care programele sunt reprezentate sub forma unor colecții de obiecte ce interacționează între ele prin intermediul mesajelor.
- Limbaje de programare orientate obiect:
  - C++
  - C#
  - Java
  - …

Concepte de bază în POO

- Principalele concepte care stau la baza POO sunt:
  - **Abstractizarea**
  - **Încapsularea**
  - **Modularitatea**
  - **Ierarhizarea**
  - **Polimorfismul**.

ABSTRACTIZAREA

- **Abstractizarea** înseamnă *eliminarea sau ascunderea deliberată a unor detalii ale unui proces sau artefact* pentru a releva mai clar alte aspecte, detalii sau structura.
  - identifică similitudinile între diferite entități, situații sau procese din lumea reală, concentrează atenția asupra acestor aspecte comune şi ignoră pentru început detaliile.
- **Abstracțiunea**:
  - exprimă toate caracteristicile esențiale ale unui obiect care fac ca acesta să se distingă de alte obiecte;
  - oferă o definire precisă a granițelor conceptuale ale obiectelor din perspectiva unui privitor extern.

Abstractizarea - Exemplu

- Tipul abstract de date ”`Student`“:

```
typedef struct student {
  char nume[50];
  char facultate[30];
  int anStudii;
} Student;
```

- Instanțierea tipului abstract ”`Student`“:

```
Student s = {"Popescu Emil”, "Informatica", 1};
```

ÎNCAPSULAREA

- *Combinarea datelor şi metodelor într-o singură structură de date*, definind totodată modul în care obiectul şi restul programului pot referi datele din obiect.
- *Compartimentarea elementelor unei abstractizări în structură şi comportament*; încapsularea separă comportamentul (accesat prin interfață) de structură, definită prin implementare.
- *Obiectele nu pot schimba starea internă a altor obiecte în mod direct* (ci doar prin metode puse la dispoziție de obiectul respectiv).
- *Doar metodele proprii ale obiectului pot accesa starea acestuia*.
- Fiecare *tip de obiect expune o interfață* pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.

MODULARIZAREA

- *Modalitatea de a grupa abstracțiuni legate logic între ele*.
- Procesul de partiționare a unui program în componente individuale (module) ceea ce permite reducerea complexității programului prin definirea unor granițe bine stabilite şi documentate în program.
- *Gradul de cuplaj* (conectare între module) *trebuie să fie mic*, pentru ca modificările aduse unui modul să afecteze cât mai puține module. Pe de altă parte, clasele ce compun un modul trebuie să aibă legături strânse între ele pentru a justifica gruparea.

IERARHIZAREA

- Reprezintă o *ordonare a abstracțiunilor*.
- Principalele tipuri sunt:
  - **Moştenirea** (*ierarhia de clase*) - relație între clase în care o clasă împărtăşeşte structura şi comportarea definită în una sau mai multe clase (semantic implică o relație de tip "*is a*").
  - **Agregarea** (*ierarhia de obiecte*) - relație între două obiecte, în care unul dintre obiecte aparține celuilalt obiect. (semantic implică o relație de tip "*part of*").

MOŞTENIREA

- Reprezintă *procesul de definire şi creare a unor clase specializate plecând de la clase (generale)* care sunt deja definite.
- Permite construirea unor clase noi, ce păstrează caracteristicile şi comportamentul, deci datele şi funcțiile membru, de la una sau mai multe clase definite anterior, numite *clase de bază*, fiind posibilă redefinirea sau adăugarea unor date şi funcții noi.
- O clasă ce moşteneşte una sau mai multe clase de bază se numeşte *clasă derivată*.
- *Clasa derivată* se află întotdeauna pe un nivel imediat inferior celui corespunzător *clasei de bază*.

POLIMORFISMUL

- Proprietatea unor entități de:
  - *a se comporta diferit în funcție de starea lor;*
  - *a reacționa diferit la acelaşi mesaj*.
- Proprietate a unui limbaj OO de a permite manipularea unor obiecte diferite prin intermediul unei interfețe comune.
  - Există o relație de derivare între interfață şi celelalte clase.
- Oferă posibilitatea a procesa obiecte în mod diferit, în funcție de tipul sau de clasa lor, prin redefinirea unor metode pentru clasele derivate.

Obiect

- Un **obiect** este o *reprezentare a unei entități din lumea reală asupra căruia se poate întreprinde o acțiune sau care poate întreprinde o acțiune*.
- Un obiect este caracterizat de:
  - *nume*;
  - *atribute* (date):
    - valorile atributelor la un moment dat definesc o stare;
  - *metode* (servicii, operații).

Obiect

- A Real World Object : Dog
  - have *state* like
    1. *Name,*
    2. *Colour,*
    3. *Breed*
  - have *behaviours* like
    1. *Barking,*
    2. *Eating*

State	Value
Name	Tommy
Colour	Brown
Breed	Labrador

- A Real World Object : Car
  - have *state* like
    1. *Brand,*
    2. *Make,*
    3. *Engine CC*
  - have *behaviours* like
    1. *Accelerate,*
    2. *Brake,*
    3. *Change Gear*

State	Value
Brand	Maruti
Make	Swift
Engine CC	1248 CC

Obiect

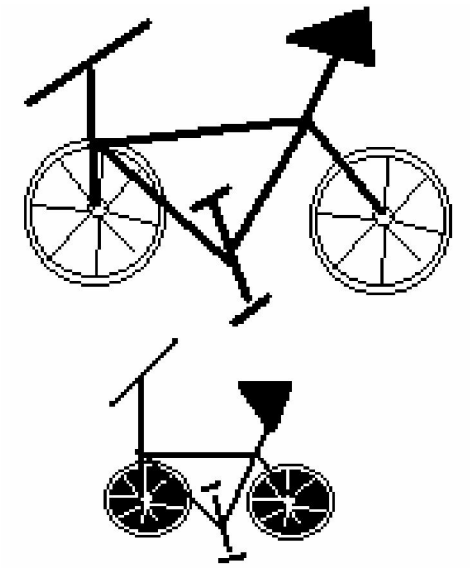
- A Programmatic object : Bank Account
  - have state like
    - *Acct No,*
    - *Acct Holder's Name,*
    - *Current Balance*
  - have behaviours like
    - *Deposit(),*
    - *Withdrawal()*

- A Programmatic object : Employee
  - have state like
    - *Employee ID,*
    - *Employee Name,*
    - *Basic Salary*
  - have behaviours like
    - *getData(),*
    - *putSalary()*

Clasa

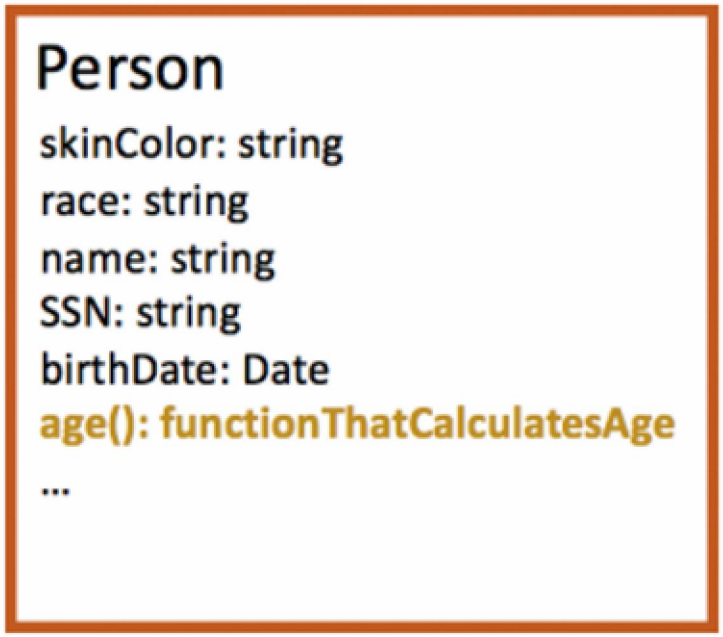
- O **clasă** este *o colecție de obiecte cu aceiaşi structură (caracteristici) şi acelaşi comportament (metode sau operații)*.

Obiecte - Biciclete



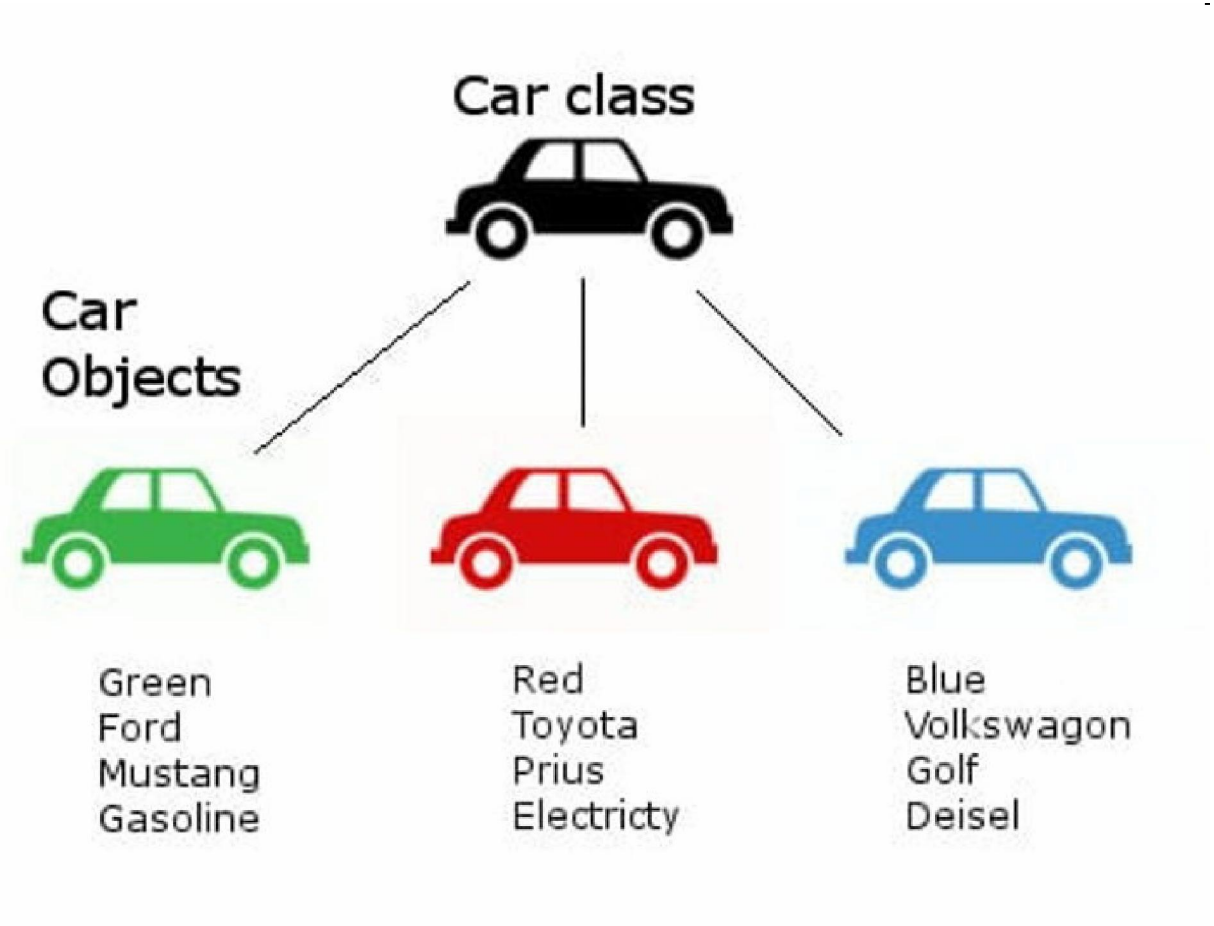
Clasa Bicicleta
- atribute
  - tip cadru
  - dimensiunea rotii
  - numar de viteze
- metode
  - acelereza
  - franeaza

Clasa



Clasa Persoana
- atribute
  - culoarea pielii
  - rasa
  - nume
  - CNP
  - data nasterii
- metode
  - calculeazaVarsta()

Clasa, obiecte



Identificarea atributelor şi metodelor

- În cadrul unei bănci un cont bancar are un titular, sold, o rată a dobânzii, număr de cont şi se pot efectua operații de depunere, extragere, actualizare sold.
- Extragerea atributelor:
  - titular, sold, rată a dobânzii, număr de cont.
- Extragerea metodelor:
  - depunere, extragere, actualizare sold.

Tipuri de date abstracte şi obiecte

- A doua definiție pentru obiecte şi clase:
  - O **clasă** este o *implementare a unui tip de date abstract*. Ea defineşte atribute şi metode ce implementează structura de date respectiv operațiile tipului de date abstract.
  - Un **obiect** este *o instanță a unei clase*. El este unic determinat de numele său şi defineşte o stare reprezentată de valorile atributelor sale la un anumit moment particular.

Sintaxa definirii unei clase

- Definirea unei clase constă din două părți distincte:
  - ***declararea*** clasei;
  - ***implementarea*** clasei respective.
- Sintaxa:

```
class IdNumeClasa {
  declaratii de date membru;
  declaratii/definitii de functii membru;
};
```

Protecția datelor şi funcțiilor membre

- **Funcțiile şi datele unei clase pot fi grupate din punct de vedere al dreptului de acces în:**
  - **private –** conține date şi funcții membre ce pot fi folosite doar de către celelalte funcții aparținând clasei respective.
  - **protected –** similară cu *private* dar care dă drepturi de acces şi funcțiilor membre ale claselor derivate din clasa respectivă.
  - **public –** drept de acces tuturor.

Declararea/definirea unei metode

Declarare:

```
class IdNumeClasa {
  tip idNumeMetoda(tip1 p1, …, tipn pn);
};
```

Definire:

```
tip IdNumeClasa::idNumeMetoda(tip1 p1, …, tipn pn) {
  //instructiuni
}
```

Exemplu

Declararea clasei `ContBancar`:

```
class ContBancar {
  private:
    char titular[100];
    char codIBAN[25];
    float sold;
  public:
    void init(const char _titular[], const char _codIBAN[], float
_sold);
    void depune(float suma);
    void retrage(float suma);
    float getSold();
    void  afis();
};
```

Exemplu

Implementarea clasei `ContBancar`:

```
void ContBancar::init(const char _titular[], const char _codIBAN[],
float _sold) {
  strcpy(titular, _titular);
  strcpy(codIBAN, _codIBAN);
  sold = _sold;
}
void ContBancar::depune(float suma) {
  sold = sold + suma;
}
void ContBancar::retrage(float suma) {
  sold = sold - suma;
}
```

Exemplu

Implementarea clasei `ContBancar`:

```
float ContBancar::getSold() {
  return sold;
}

void  ContBancar::afis() {
  printf("Titular: %s\n", titular);
  printf("Cod IBAN: %s\n", codIBAN);
  printf("Sold: %g\n", sold);
}
```

Sintaxa declarării unui obiect

```
class IdNumeClasa {
  // ….
} idOb1, …, idObN;
```

sau

```
IdNumeClasa idOb1, …, idObN;
```

Referirea la datele şi metodele membru

```
idOb.idDataMembru
idPointerOb->idDataMembru


idOb.idMetodaMembru(lista_de_parametri);
idPointerOb->idMetodaMembru(lista_de_parametri);
```

Exemplu

```
int main() {
  ContBancar c;
  c.init("Popescu", "RO49RNCB0080005630320001", 100);
  c.afis();

  printf("Depun 10 RON\n");
  c.depune(10);
  c.afis();

  printf("Extrag 15 RON\n");
  c.retrage(15);
  c.afis();
  return 0;
}
```

Referințe

- https://www.embhack.com/stages-of-compilation-in-c-program/
- http://cs.brown.edu/courses/csci1310/2020/assign/labs/lab2.html
- https://www.embhack.com/memory-layout-of-c-program/
- https://www.cs.swarthmore.edu/~newhall/unixhelp/compilecycle.html
- http://www.lmpt.univ-tours.fr/~volkov/C++.pdf
- https://en.cppreference.com/w/cpp/language/history
- https://www.tiobe.com/tiobe-index/