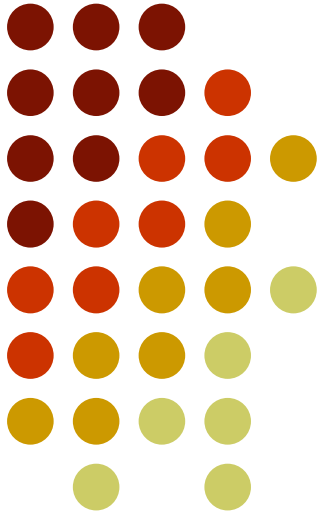
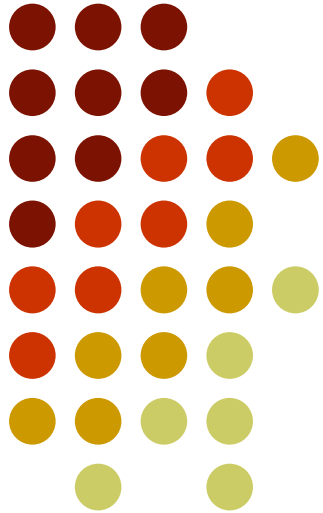
תכנות מונחה עצמים בשפת C++

אוהד ברזילי

אוניברסיטת תל אביב



עבודה עם קבצים וסדרתיות ב C++



זרמים

- קבצים הם מקרה פרטי של זרמים (steams) – מידע הזורם מ\אל מקור\יעד כלשהו
- במערכות הפעלה רבות כל ישות במערכת היא קובץ ולכן אוסף הפעולות שניתן לבצע על קובץ (interface) תקף גם לגבי ישויות מערכת ההפעלה האחרות
  - לדוגמא: קריאה ממקלדת, כתיבה למסך, תקשורת רשת התקני חומרה ועוד...

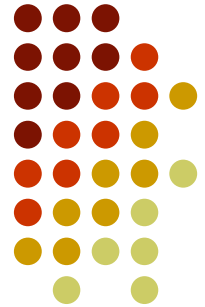
טיפול בקבצים

- בשפת C ניתן לעבוד עם קבצים ע"י מצביע לקובץ וסדרת פעולות (פונקציות גלובליות) עליו
- ב C++ ניצור עצם שייצג את הקובץ ונבצע עליו פעולות בעזרת שימוש במתודות שלו

```
FILE* pfile;
fwrite(pfile, array, size, …);
```

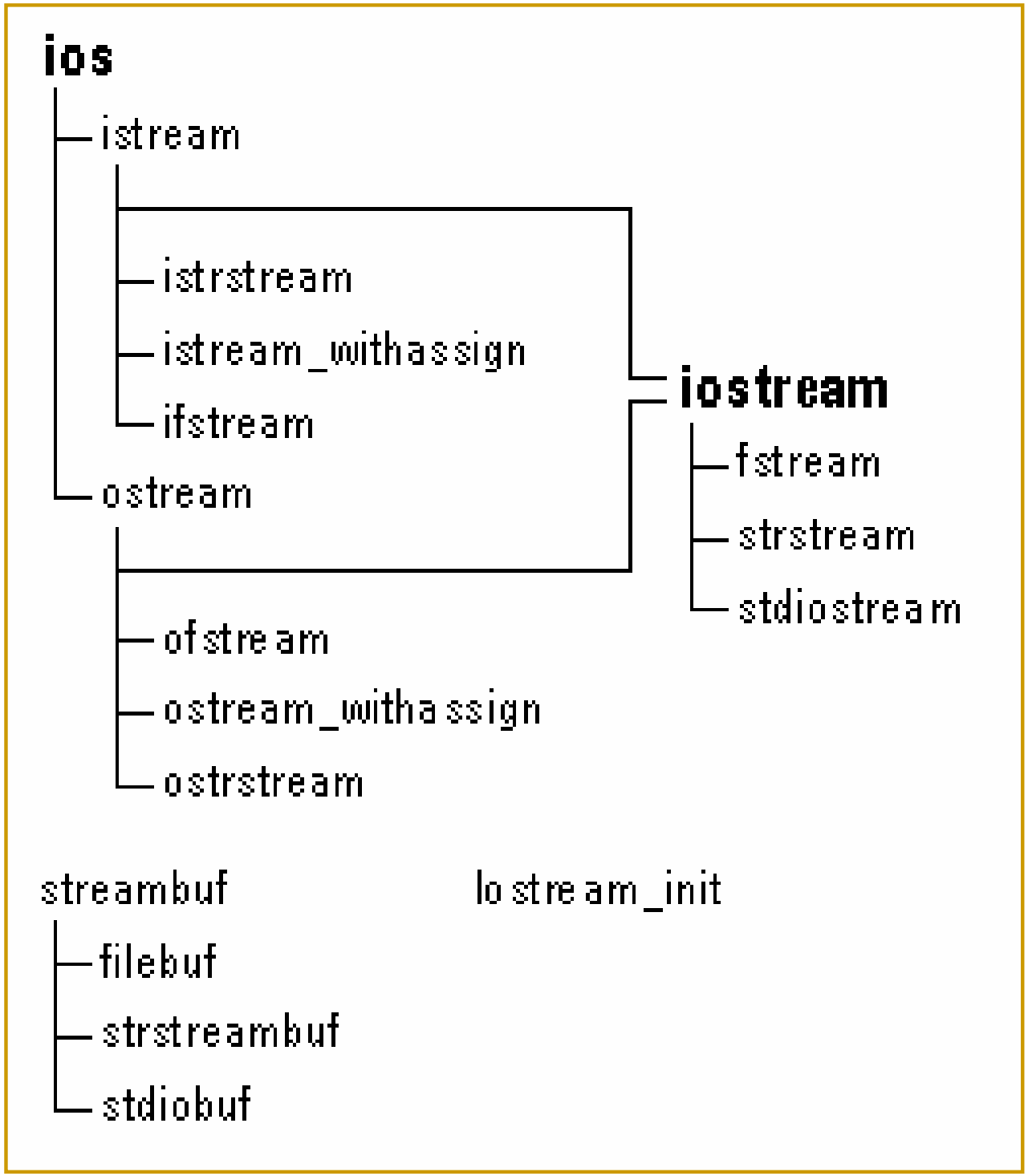
```
ofstream out_file(…);
out_file.write(array, size, …);
```

- למרות שפונקציות C עדיין זמינות לא נשתמש בהן

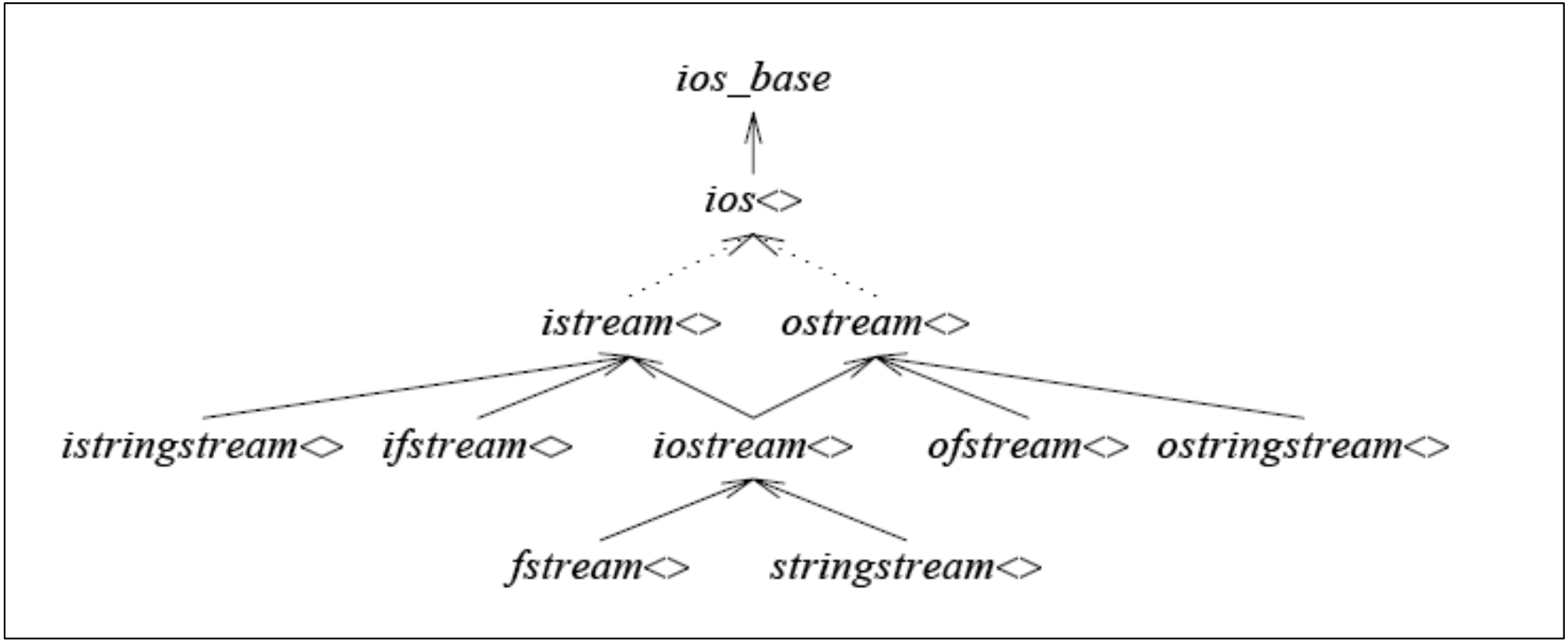


הספרייה התקנית

- הספרייה התקנית מספקת מחלקות לעבודה עם קבצים:
  - ofstream – כתיבה לקובץ
  - ifstream – קריאה מקובץ
  - fstream – קריאה וכתיבה לקובץ (ירושה מרובה)
  - מחלקות אלו הן חלק מהיררכיה ענפה לטיפול בזרמי קלט \ פלט (I/O) הכוללת בין השאר מחרוזות כזרמים
  - ההיררכיה מבוססת על תבניות שהטיפוסים הפורמלים בהן הם מחלקת התווים ומחלקת התכונות של מחלקת התווים

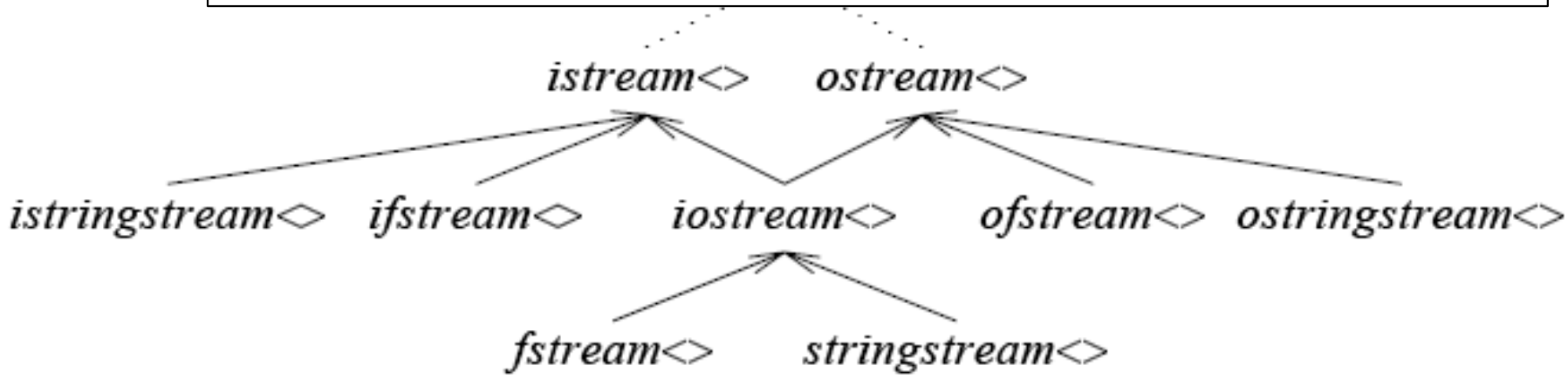

ios
istream
istrstream
istream_withassign
ifstream
ostream
iostream
fstream
strstream
stdiostream
ofstream
ostream_withassign
ostrstream
streambuf
filebuf
strstreambuf
stdiobuf
Iostream_init

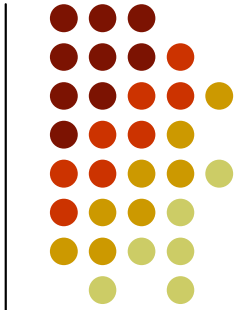
```
template <class Ch, class Tr = char_traits<Ch> >
class basic_ofstream : public basic_ostream<Ch, Tr> {
public:
        basic_ofstream();
        explicit basic_ofstream(const char* p, openmode m = out);

        basic_filebuf<Ch, Tr>* rdbuf() const;

        bool is_open() const;
        void open(const char* p, openmode m = out);
        void close();
};
```



```
template <class Ch, class Tr = char_traits<Ch> >
class basic_ofstream : public basic_ostream<Ch, Tr> {
public:
        basic_ofstream();
        explicit basic_ofstream(const char* p, openmode m = out);

        basic_filebuf<Ch, Tr>* rdbuf() const;

        bool is_open() const;
        void op
        void cl
};
```

```
typedef basic_ifstream<char> ifstream;
typedef basic_ofstream<char> ofstream;
typedef basic_fstream<char> fstream;

typedef basic_ifstream<wchar_t> wifstream;
typedef basic_ofstream<wchar_t> wofstream;
typedef basic_fstream<wchar_t> wfstream;
```

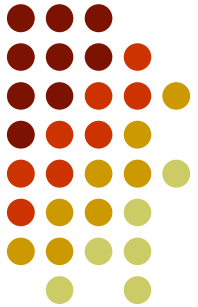




cout

הערה:

- העצם cout מהמחלקה ostream מוגדר במרחב השמות std כ- extern (גלובלי במרחב השמות) ומאותחל לשלוח פלט למסך. לשם כך אנו מכלילים את הכותרת <iostream>
- המחלקה ostream העמיסה את האופרטור >> לכל הטיפוסים הבסיסיים (ולחלק מהמחלקות התקניות)
- בדר"כ נשמור על מוסכמה זו בעבודה עם קבצים



פתיחת קובץ

- ניתן לפתוח קובץ ע"י המתודה open או ישירות בבנאי:

```
void ofstream::open(const char* filename, int mode);
void ifstream::open(const char* filename, int mode);
```

  - **filename** – שם הקובץ
  - **mode** – צורת הפתיחה (ניתן לבחור יותר מאופציה אחת ע"י |)
    - **ios::app** – שרשור
    - **ios::ate** – פתח עם הסמן בסוף הקובץ
    - **ios::in / ios::out** – ברירות המחדל
    - **ios:nocreate / ios::noreplace** – פתח רק אם קיים \ לא קיים
    - **ios::trunc** – פתח קובץ ריק
    - **ios::binary** – פתח קובץ בינארי (ברירת מחדל: טקסט)

- בסיום העבודה עם הקובץ – close()

שאילתות

- is_open() – האם הקובץ מפתח כהלכה? (ניתן גם להשתמש באופרטור המועמס '!' כמו ב C)

```
FILE* pfile = fopen(…);
if (!pfile) …
```

```
ofstream out_file(…);
if (!out_file) …
```

- rd_state() – מחזיר משתנה מטיפוס ios::iostate עם חיווי כללי על הקובץ:
  - ios::goodbit – תקין
  - ios::eofbit – הסמן הגיע לסוף הקובץ
  - ios::faildbit – הפעולה הבאה תכשל
  - ios::badbit – קובץ לא תקין – לא ניתן להמשיך

דוגמא

```
void f()
{
        ios_base::iostate  s = cin.rdstate();   // returns a set of iostate bits

        if (s & ios_base::badbit) {
                // cin characters possibly lost
        }
        // ...
        cin.setstate(ios_base::failbit);
        // ...
}
```

- ניתן גם להשתמש במתודות `bad()` , `fail()` `good()` , `eof()`
- לאחר שגילינו שגיאה יש לאפס את שדות השגיאה ע"י המתודה `clear()`

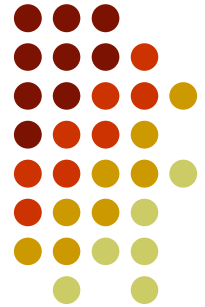
הזזת הראש

- הזזת הראש מתבצעת ע"י מתן מספר תווים ונקודת ייחוס (ios::beg , ios::cur , ios::end):
  - seekp() – הזזת הראש הכותב (put)
  - seekg() – הזזת הראש הקורא (get)

```
int f(ofstream& fout)
{
        fout.seekp(10);
        fout << '#';
        fout.seekp(-2, ios_base::cur);
        fout << '*';
}
```

מיקום הראש

- הראשים הקוראים כותבים מקודמים אוטומטית לאחר כל פעולת קריאה \ כתיבה
- מיקומם נגיש ע"י
  - tellg() – מיקום הראש הקורא (get)
  - tellp() – מיקום הראש הכותב (put)



קריאה וכתיבה לקובצי טקסט

- כתיבה:
  - put() – כותב תו בודד לקובץ
  - << לכתיבת כל דבר אחר (כולל תווים)
- קריאה:
  - get() – קורא תו בודד מקובץ
  - getline() – קריאה של שורה שלמה (עד גודל מקסימלי או תו מסיים)
  - >> לקריאה כללית

קריאה וכתיבה לקובצי טקסט
דוגמא

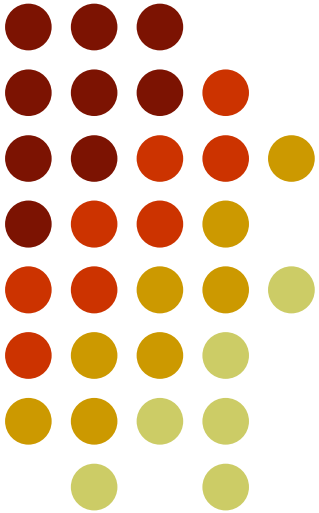
```
#include<fstream.h>

int main()

{
    // Writing to file
    ofstream OutFile("my_file.txt");
    OutFile<<"Hello "<<5<<endl;
    OutFile.close();


    int number;
    char dummy[15];


    // Reading from file
    ifstream InFile("my_file.txt");
    InFile>>dummy>>number;


    InFile.seekg(0);
    InFile.getline(dummy, sizeof(dummy));
    InFile.close();

}
```

קריאה וכתיבה לקבצים בינרים

- כתיבה של n בתים:
  - `write (const unsigned char* buffer, int n);`
- קריאה של n בתים (למערך שהוקצה מראש):
  - `read (unsighed char* buffer, int num);`
  - המתודה `int gcount()` מחזירה את מספר הבתים שנראו בפועל

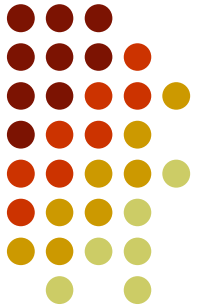
```
#include<fstream.h>

int main()

{
   int array[] = {10,23,3,7,9,11,253};
   ofstream OutBinaryFile("my_b_file.txt“, ios::out | ios::binary);
   OutBinaryFile.write((char*) array, sizeof(array));
   OutBinaryFile.close();

}
```

סדרתיות (serialization)



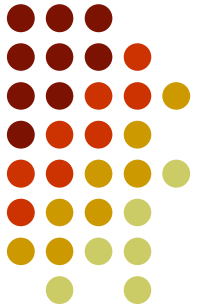
סדרתיות (serialization)

- היכולת של עצם ל"סדרת" את עצמו
- עצם שיודע זאת ניתן לשמור בקובץ, לשלוח ברשת, להציג על המסך ועוד...
- הרעיון הכללי פשוט: לאט ובזהירות
- כדי לשמור struct ב-C היינו עושים זאת שדה-שדה
- אין טעם לשמור מצביע, במקום זאת נשמור את העצם המוצבע ואת גודלו (בסדר הפוך)



שמירת עצמים לקובץ

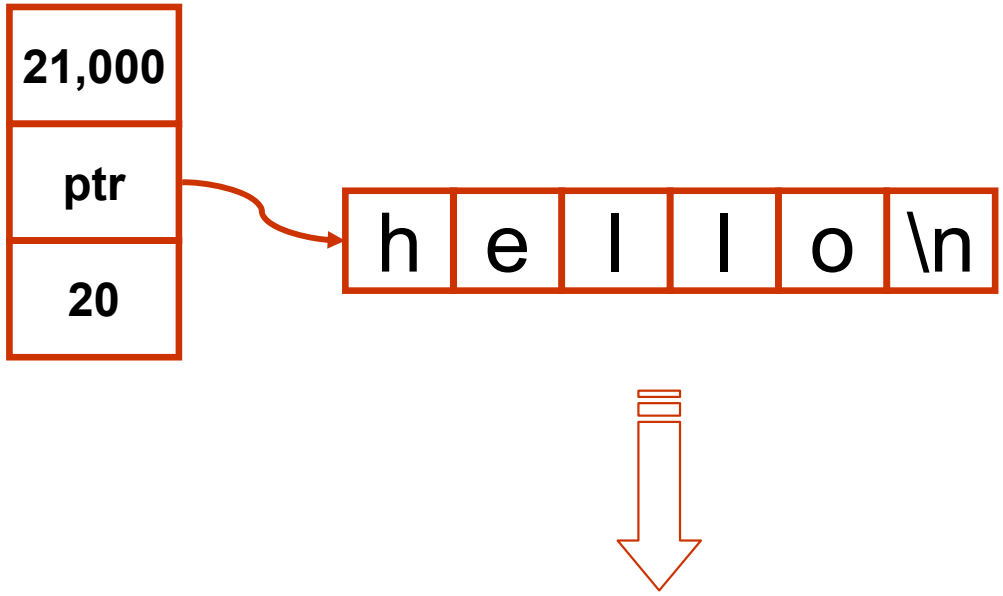
- עבור כל מחלקה נגדיר מתודה וירטואלית Save (בשפות אחרות מקובל השם serialize):
  - המתודה תקבל כפרמטר שם של זרם (קובץ במקרה שלנו)
  - המתודה תשמור את העצם שדה-שדה, מצביעים יתורגמו ל: גודל+תוכן (נניח בינתיים שכל העצמים שאנו שומרים מהווים עץ - אף עצם אינו מוצבע פעמיים, אין הצבעות הדדיות וכו')
  - אין צורך לשמור מצביעים לטבלה הוירטואלית או לעצמים משותפים של מחלקת בסיס וירטואלית
  - מחלקה נגזרת תקרא קודם למתודת ה Save של מחלקת הבסיס ורק אז תשמור את עצמה

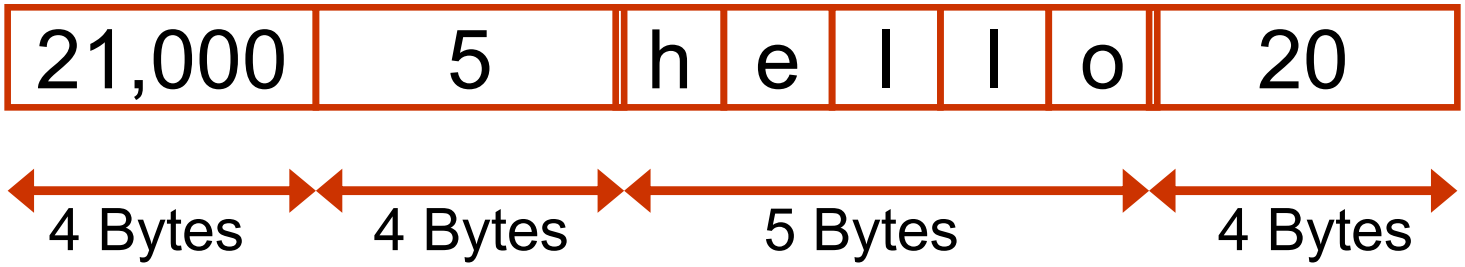


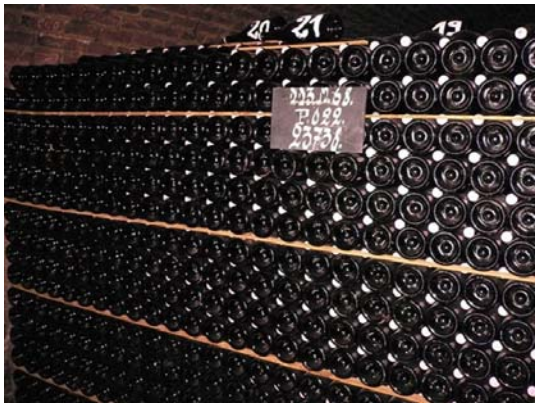
טעינת עצמים מקובץ

- עבור כל מחלקה נגדיר:
  - בנאי טעינה המקבל כארגומנט טיפוס ifstream ובונה עצם בלוגיקה הפוכה לזו של מתודת השמירה
  - מתודה וירטואלית Load (בשפות אחרות מקובל השם unserialize): הטוענת נתונים לתוך עצם קיים
- שדה המוקצה דינאמית יוקצה ע"י בנאי הטעינה ומתודת ה- Load

שמירה לקובץ בינארי





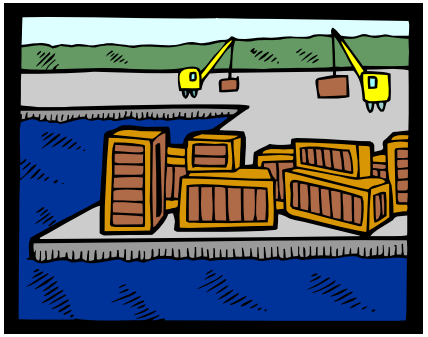


שמירת מיכלים

- אנו מעוניינים לשמור את כל העצמים מתוך מיכל פולימורפי כלשהו (לכולם מחלקת בסיס משותפת)
- נגדיר את המתודה Save כוירטואלית במחלקת הבסיס
- נגדיר מתודת Save למיכל עצמו (אם הוא סטנדרטי נירש ממנו \ נכיל אותו \ פונקציה גלובלית וכו') השומרת את כל אבריו איבר-איבר

טעינת מיכלים

- איך נממש את מתודת ה Load ? איזו מתודת Load נפעיל ?
- עלינו כבר בתהליך השמירה לשמור לפני כל עצם מציין של המחלקה שלו
- פונקציית ה Load תקרא את המציין (קוד, מחרוזת וכו') ותפעיל בנאי של המחלקה הנכונה

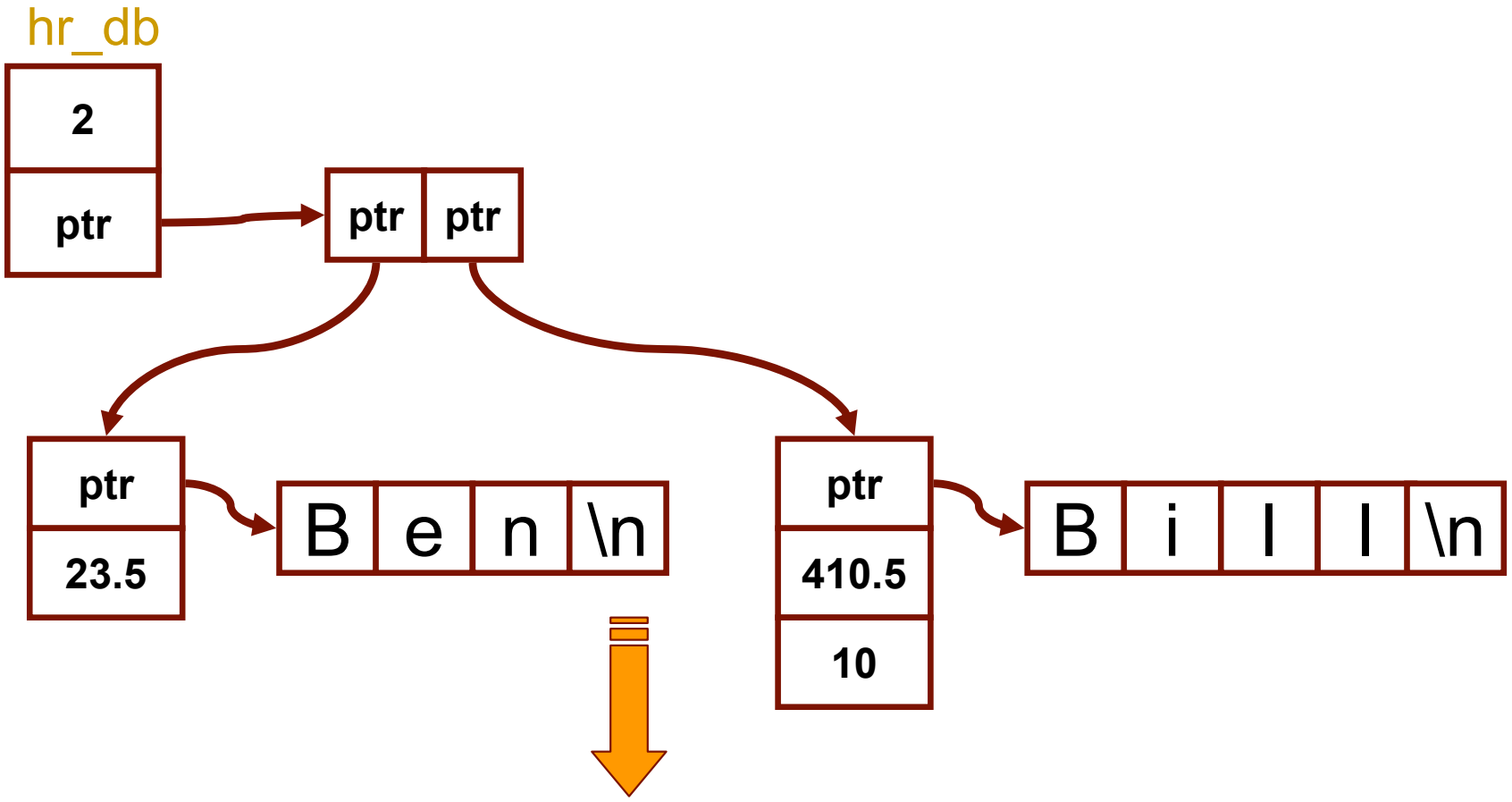


דוגמא

- ברצוננו לתחזק בסיס נתונים של עובדי חברה עם הפעולות: input / print / save / load
- עובד בחברה מוגדר כך:

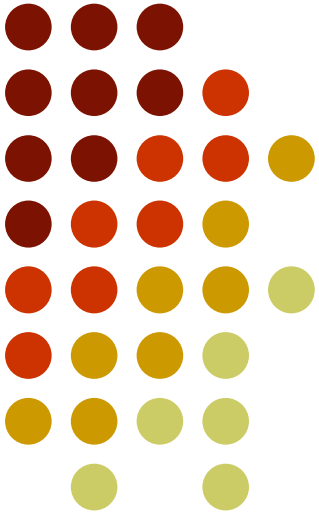
```
class employee
{
  char *Name;
  float Salary;
};
```

- למנהל (שיורש מעובד) תכונה נוספת: `int Level`



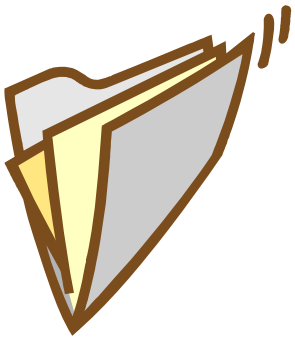
- **הדוגמא המלאה (של אלחנן בורנשטיין) בקובץ באתר**

נושאים מתקדמים בשמירה סדרתית של נתונים



קובץ בינארי או טקסט?

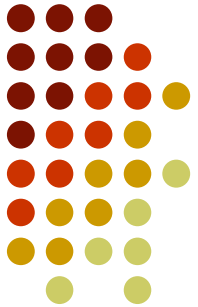
- קובץ הטקסט:
  - קריא יותר ונוח לעבודה 'אנושית'
  - המשתמש לא מודע לגודל לפורמט של ייצוג הנתונים בזכרון
  - עדיף כאשר רוב הקלט מבוסס מחרוזות
- קובץ בינארי:
  - יעיל יותר בזמן עיבוד
  - חוסך את בעיית ההפרדה בין שדות
  - עדיף בעבודה עם מספרים גדולים



פרמטרים המקשים על סדרתיות

- כאשר העצמים הנשמרים אינם שייכים לאותה מחלקה
- כאשר העצמים הנשמרים מכילים מצביעים:
  - היוצרים עץ (ללא מסלולים שונים לאותו צומת)
  - היצרים גרף חסר מעגלים (בעיית השיתוף)
  - היוצרים גרף (המכיל אולי מעגלים ומסלולים כפולים)

- בדוגמת העובדים פתרנו רק את בעיית העץ!



טעינת עצמים ממחלקות שונות

- איך ניצור עצם מהמחלקה הנכונה ללא שימוש ב switch על מזהה המחלקה?
- נשתמש ב"בנאי וירטואלי"
  - נגדיר עבור מחלקת הבסיס של כל עץ ירושה מתודה וירטואלית טהורה בשם create. למשל עבור ההיררכיה של Shape נגדיר:

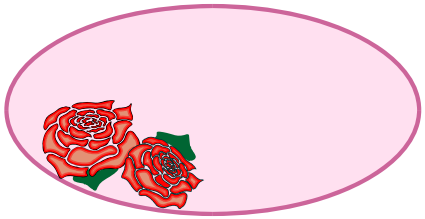
```
Shape *create(std::istream&) const=0;
```

  - בכל אחת מהמחלקות היורשות נממש את המתודה שתחזיר עצם חדש מהמחלקה שבה היא נמצאת

טעינת עצמים ממחלקות שונות

- למשל עבור המחלקה אליפסה נגדיר:

```
Shape *Ellipse::create(std::istream& istr) const
{
   return new Ellipse(istr);
}
```

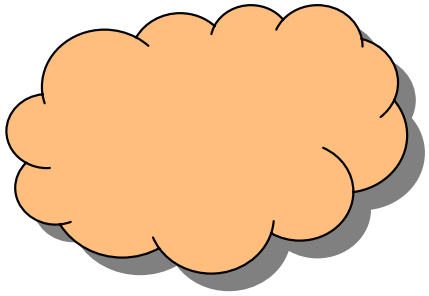


- כדי ליצור קבצים באלגנטיות, ניצור מיפוי (map) בין המזהה של כל מחלקה ובין נציג של המחלקה (המכונה prototype)
- למשל עבור המיפוי `static map<string , Shape*>` נכניס את הזוג `"Ellipse"` עם `new Ellipse()`.

טעינת עצמים ממחלקות שונות

- המתודה שקוראת את המחלקות מהקובץ (Load או unserialize) תקרא את המזהה ולפיו תפעיל את פונקצית ה create של הנציג המתאים במיפוי:

```
Shape *Shape::unserialize(std::istream& istr)
{
   // read code...

   if (theMap.count(className) == 0)
       throw ...something...

   return theMap[className]->create(istr);
}
```



טעינת עצמים ממחלקות שונות

- ואריציה על השיטה משתמשת ב"בנאי עם שם":
  - נגדיר בתוך Shape:

```
typedef Shape* (*Factory)(std::istream&)
```

  - נגדיר את המיפוי כך:

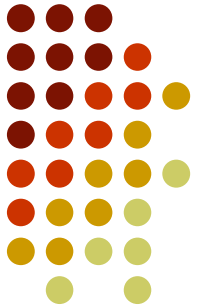
```
std::map<std::string,Factory>
```

  - נאכלס את המיפוי בפעולה במקום בנציג:

```
theMap["Ellipse"] = Ellipse::create
```

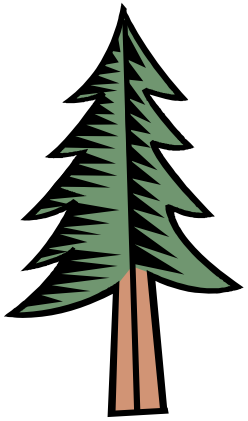
  - במקום לקרוא ל create על הנציג, נפעיל את הנציג:

```
theMap[className](istr)
```



סריקת עץ מצביעים

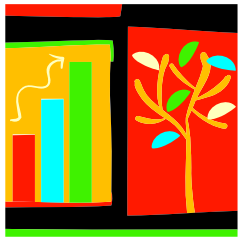
- ישנן תבניות עיצוב אשר מכלילות שיטה זו (Composite , Visitor או Decorator) שיילמדו בהמשך הקורס
- החשוב הוא להיות עקבי בדרך שבה נשמרים העצמים ובדרך שבה הם נקראים
  - למשל DFS: אם העצם שאנחנו שומרים מכיל int a, מצביע בשם b, float c ועוד מצביע ל- d:
    - נכתוב את a
    - נמשיך רקורסיבית (נצלול) ל- b
    - נכתוב את c
    - נמשיך רקורסיבית (נצלול) ל- d



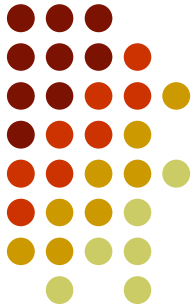
סריקת עץ מצביעים

- תהליך הקריאה יהיה הפוך:
  - נקרא נתון לתוך a
  - נקצה עצם עבור b ונקרא לפעולת הקריאה שלו
  - נקרא נתון לתוך c
  - נקצה עצם עבור d ונקרא לפעולת הקריאה שלו

- שימוש במצביעים חכמים (auto_ptr) ידאג לשחרור הזכרון שכבר הוקצה במקרה של חריגה



שמירת גרף עצמים



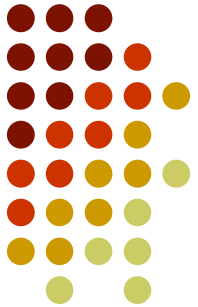
- נשתמש במעבר כפול על גרף המצביעים (two-pass algorithm) ובמיפוי בין עצמים והמזהה שלהם
  - לדוגמא: `std::map<Node*,unsigned> oidMap`

- במעבר הראשון על גרף העצמים נמלא את הטבלה:
  - עבור כל עצם אם הוא איננו בטבלה נוסיף אותו ביחד עם מזהה יחודי ונמשיך רקורסיבית לעצמים המוצבעים ממנו
  - מזהה ייחודי אפשרי הוא גודל המיפוי הנוכחי:

```
unsigned n = oidMap.size();
oidMap[nodePtr] = n;
```

שמירת גרף עצמים

- במעבר השני נעבור על אברי המיפוי ונשמור אותם בזה אחר זה, ראשית את המזהה ואחר כך את השדות
- כאשר בתהליך שמירת תוכנו של העצם נתקל במצביע לעצם אחר, במקום להמשיך רקורסיבית לעצם זה נרשום רק את המזהה שלו
  - למשל עבור המצביע: `Node* child` נשמור את המזהה `oidMap[child]`
- המיפוי אינו נחוץ לתהליך הטעינה של העצמים



טעינת גרף עצמים

- נטען את העצמים ע"י מעבר כפול על הקובץ (קיימות דרכים יעילות יותר אך פשוטות פחות)
- במעבר הראשון נאכלס מערך עצמים מהטיפוס המצופה (נניח בינתיים שכל העצמים מאותו הטיפוס)
  - למשל: `std::vector<Node*> v`
- כל המצביעים בתוך עצמים אלו יאותחלו להיות NULL
- במעבר השני נכוון את כל המצביעים הפנימיים:
  - למשל אם `v[3].child` מכיל את הערך 5 נבצע את ההשמה `v[3].child = v[5]` או בדומה (אם השדה private)
- לאחר סיום המעבר השני הוקטור המקומי מיותר