מונחה עצמים

C++

סיכום החומר
והכנה למבחן

Scanned by CamScanner

אובייקטים, מחלקות, מתודות וספריות

יתרונות שפת מונחה עצמים:

← אורגון טוב יותר של הקוד
← קוד קטן יותר
← שחצור קודים שנכתבו כבר
← עובדה פשוט יותר
← פשוט יותר לפתח

מחלקה class

המקום בו מאוגד כל המידע - מבנה משתנים ומתודות. הדרה של שליט חלק.
כשש מחלקה מתוך מחלקה הדשה לשתנים נגישים באמצעה מחלה.

חברי מחלקה - Data Members - אובייקטים ומתודות

משתני המחלקה שמכילים את המידע, וכולם להיות מסוגים פרמיטיביים
או מסוגים מחלקה אחרת. יכולים להיות Public / Private / protected

מתודות

פונקציות במחלקה המאפשרת גישה למשתני המחלקה ולמידע הפנימי.
יכולות להיות Public / Private / protected

ספריה iostream.h

ספרה סטנדרטית המכילה אובייקטים של קלט cin ופלט cout
ומעמד של קלט פלט לעסק
לא רוצים לבנות מחלקה משלנו מינהרה ofstream

Namespace.std

רשימים מתחולת הטבעות ומשמעותה שאפשר לתשתמש מכל מה שנמצא ב std
מבנה מחלקה, אובייקטים ופונקציה.
std שם המשפחה של iostream

Scanned by CamScanner

משתני ברירת מחדל

יש אפשרות לקבוע ערכי ברירת מחדל על ארגומנטים של פונקציה שמגדירה ואז נשלח ערך לאחד הארגומנטים הפונקציה יודעת מה ברירת המחדל שלו

קביעת הערכים מתבצעת בהכרזת המחלקה

מתודות get, set

מתודות בסיסיות בכל מחלקה אשר מאפשרות:

get - להחזיר את משתני המחלקה, כי לרוב הם יהיו private ואי אפשר יהיה לגשת אליהם.

set - אתחול משתני המחלקה.

העברה והחזרה - Value / by Reference

← תמיד נשלח class על מה שמופיע & , כיוון שהוא טיפוס שדולג המרכיב אובייקטים ומתודות, וכיוון שרק כדאי נוכל לגשת שונו עליו.

← פונקציה יכולה לקבל ערך או כתובת ולהחזיר ערך / כתובת

void f (int & a) int & f (int x)

• וגם דרך להתעסק עם * כדי לשלח ערך כמו גם להחזרה &.

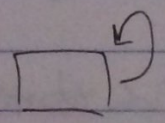
פונקציות inline

כאשר מגדירים פונקציות כ inline, כל מקום שהפונקציה מופיע בקוד הקומפיילר יחליף את הקריאה לפונקציה במקום שלה.

הקומפיילר יכול לבצע את זה שבתוכה או שלא.

פונקציות קצרות נכתבות עם את ההגדרה ועם את המימוש בקובץ h.

this



פוינטר עצמי למחלקה

const

כשלוחצים ארגומנט כ const זה אומר שהוא מוגן - לא נשתמש לשנות את ערכו.

מאפשר עם לשלוח משתנים שלא רוצים המרה כ const

Scanned by CamScanner

Constructor and Destructor

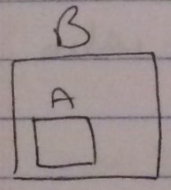
Construct

הקונסטרקטור הוא פונקציה שיקרא לאחר שיווצרים אובייקט.
הוא למעשה הבנאי - , אין לו ערך החזרה.
← default const
זה למעשה בנאי דיפולטיבי, אינו מקבל ארגומנטים
← אפשר להוסיף בנאי כרצונך המקבל ארגומנטים כמשתנים וחוקה על פיהם.
← לפעמים נצטרך לאתחל או לעדכן משתנים לפני הכנסה לבנאי,
ולכן נעשה את זה בשורת האתחול, כמו לגשת משתנה במחלקה B
מאפיינים מחלקה A.
עשים זאת ב-cpp בלבד.

משתמש עם
נגישות אובייקט אחרי

Destructor

← כקונבנציה לקונסטרקטור שדאשל קוראים למעשה ל destr
למעשה הוא משחרר משאבים שהפסתי.
destr של אובייקטים שקולים מתבצע בסדר ההורשה.
← destr פועל בדרך הפוכה מ constr כלומר



Destructor	constructor
- משחרר את B	- בונה את A
- משחרר את A	- בונה את B

העתקה

← העתקה רדודה - משמש הקצאה דינמית, כדי למנוע כפילות כתובת עם של פוינטרים
← העתקה עמוקה - באמצעות copy constructor
מחזה אשר מקבלת ארגומנט מאותו סוג המחלקה ומחזיר המחזה
עבור העתקה של משתנה משתנה.
כי בעת ולדרוש מחלקה חדשה אנו צריכים לפרוס את כל הפרטים הפשוטים עליה
זה לא משתנה הפרמיטיבי שהמחלקה מכיר.

Scanned by CamScanner

Exception, FriendShip and More

Exception#

לפעמים מתקבל פלט לא תקין כמו למשל: מס שלוי המבוקש מרחק
והמשך הקטע הקוד וכל להמשיך להתקדם למרות הבעיה שהתקבלה, לצורך
הארכיטקטורה על השימושים הללו נוצר מנגנון throw-try-catch
וזהו למעשה הדרך שלי להתמודד עם שימושים בצמן ריצה, שלא אוכל לצפות להן.
← איך זה עובד?
נשאל שאלה עבור המשתנה או הקלט, במידה ונמצא ל throw המשך
הקטע הקוד יפסק, לא נמצא אליו.
במקרה זאת נצרך ל-catch, אשר מבלוק זה תטופל הבעיה.
• יכולים להיות כמה בלוקים של catch המתאימים לטיפוסים השונים
שיעלו לבוק.
אם יש כמה catch על אותו הטיפוס נבדק אל הראשון שמופיע.
אם אין catch אחרי throw הבא משהו ובא מהתכנית.
• catch שמקבל לו כפרמטרים (...) הוא default, כלומר יכול לתפוס
כל מה שיגיע לו.
• ב catch ניתן לטפל בבעיה באמצעות אובייקטים, מחרוזת וכו'.
← שמרקים משהו במהלך catch קיומו בעיה - יש פרמטרים שנולדו ובאלה
אחרים שלא ומובהם דם ה destructor, לכן צריך לדאוג לשחרור משאבים
כי אין destructor שיעשה לזה.

explicit

משמעותו שניתן להשתמש ב constructor אין רק לבנאי האובייקט, לא נשמש
אותו לאופרטור =

const

const int func(const int x) const

↓ מה שהפונקציה מחזירה לא ניתן לשינוי

↓ המשתנה שהפונקציה מקבלת לא ניתן לשינוי

↓ הפונקציה לא יכולה לשנות את המחלקה שהפעילה אותה

Scanned by CamScanner

Mutable - שמתרחש כאשר לפני משתנה זה אומר שהוא ניתן לשינוי,
יש אם הוא נמצא בתוך מחלקה שהוגדרה כ const.

← משתני const חייבים להיות מוגדרים כ init.line

משתנים סטטיים

משתנה סטטי הוא בבלוק בו הוא נולד.
ברגע שיש משתנה סטטי כ data member עבור יהיה לזה אבל כל האובייקטים
מסוגים המחלקה.
• כוכבית שימוש שרובים למנוע ולעדכן שדה אחיד אבל כל המשתנים מהסוגים.
• מתודה שמשנה משתנה סטטי, צריכה להיות סטטית עם

friend

כשמגדירים מחלקה אחת כ friend של השניה זה מאפשר לה לגשת
ל data mem של המחלקה ללא שימוש בפונקציות get
• רק פונקציות גלובליות יש טעם להגדיר כ friend, לשם עושים כאן
בעומס << , >>
כיוון שבאמת לפונקציות גלובליות אין גישה ל data Memb.

פונקציה חברה של מחלקה אין לה גישה ל Data Member

enums

מאפשר להגדיר מילים שמוגדרות כמו "define"
כמעט הם יכולים לקבל ערכים, ואפשר להגדיר במחלקה להיות enums.

```
enum Days {
    SUN     index0
    Mon     index1
    tue     index2
}
```

Scanned by CamScanner

אופרטורים

מימוש אופרטור נעשה ע"י הוספת המילה operator בתחילת המילה
דרך לשים לב מה שולחים לאופרטור ומה חוזר מהאופרטור.

← למה צריך?

כיוון שאנו לא מעדיפים ילדים טיפוס חדש המורכב ממספר משתנים הלקוח פרמטרים
הלקוח לא, אם נרצה לבצע פעולה בין אובייקטים מהטיפוס החדש
המתשב לא יודע מה בוודאי יתהה שרוצה.

← דוגמא:

- בשביל אופרטורים לתשב על בניה של אופרטור עבור int
- כמעט כל פעם לא יהיה const על אופרטור
- לא יהיה אופרטור שמחזיר void
- אופרטור ההשמה = מתבצע בדרך כלל copy const
- רשימה אחת לאופרטורים:

פעולות חשבון: + , - , * , % , / , ++ , -- , += , -=
תנאים: == , > , < , >= , <= , != , && , ||
ביטוי: << הזרמה ביט ימינה, >> הזרמה ביט שמאלה, & , |
אחרים: () , [] , ! (casting)

- באופרטורים >> , << מחזירים ostream ומכן ניתן לשרשר באותה
שורה מספר הקלטים

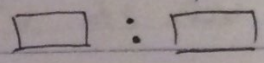
Scanned by CamScanner

ירושה

← לפעמים המחלקה בה אנו משתמשים לא מספיקה לי ואני רוצה להוסיף דברים
מבלי לשנות את הקוד של המחלקה כי: - הוא עובד טוב ולא אני רוצה להפיל באגים
- יכול להיות שרוצה מהפרוייקט
משתמשים במחלקה כמו שהיא.

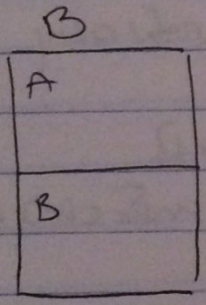
← מחלקה הירושה מחלקה מקבלת בהגדרה: את כל התכונות של המחלקה
האמא - כל השיטות והמשתנים.
הקוד של המחלקה המקורית מופיע פעם אחת
יתרון: • לא מייצר באגים חדשים
• רחבה טובה

סימון



← כל שירשם ירושה המחלקה עדיין יהיה סקלרי ויש לה יתרונות.
שימוש בירושה הוא כדי לומר זהו סוג של משהו אחר
↓
אותו ירשתי.

דוגמא: B ירש את A



אז ב B יש שני חלקים החלק ה A שלו
החלק ה B שלו

← כאשר מגדירים מתודה במחלקה החדשה העקרונות באותו שם כמו במחלקה
שירשנו מתקיימת דריסה - לוקחים מהמחלקה שלנו.
← constructor and destructor אינם עוברים בירושה.
אוברטור = הממומש במחלקה הירושה צריך להיות ממומש במחלקה
שירשם ממנה.
← 3 סוגים של ירושה: Public / Protected / Private
← יש לה מתודה m בתוך A ש B ירש:
בתוך B A::m()
בתוך B A&a=*this ואז פונים לפי משתנים שהוא מעדיף החלק הA של B

Scanned by CamScanner

פולימורפיזם

מאפשר ליצור סידים שונים של טיפוסים שצריכים יש מבנה משותף אחיד
לדוגמא: צורה
מרחק צורה נניח לדוגה משולש, עיגול, מלבן, ריבוע וכו'.

Static & Dynamic Binding

קשור בין הפונקציה לכתובת שלה בזיכרון
← Static - זה ה default בשפה, פירוש המתודה נקבעת מהמחלקה אותה יצרנו.
← Dynamic - מאפשר לשנות למתודה שרצה ולא מהמחלקה אבה יצרנו עושים זאת ע"י Virtual

Virtual

מאפשר לשנות למתודה המחלקה שרצה בזמן ריצה ← אלדורתם דינרי
למעשה זה מאפשר לנו ליצור את אלגוריתם כללי שמשנה את התנהגותו בזמן ריצה בהתאם לסוג האובייקט עימו מתמשים.
← לרוב בפולימורפיזם נצטרך להגדיר את הconstructor כוירטואלי
← Virtual ברירת מחדל - פירוש אם במחלקה הבאה הגדרנו מתודה כוירטואלי ובמחלקת הבן לא רשמנו זה אוטומטי מתייחסים כvirtual
← טבלה וירטואלית
נרכז בה את הפונקציות הוירטואליות מאיפה כל אחת נקבעת בהתאם לירושה.

מחלקה אבסטרקטית

← מחלקה אשר לא ניתן ליצור מופעים מטיפוסה.
← הרעיון שיש מתודה אחת שהיא abstract פירוש לא ממומשת = 0
המחלקה נקראת אבסטרקטית
מחלקה הורשת מחלקה אבסטרקטית חייבת לממש את כל המתודות ה abstract
דוגמא: מחלקה אבסטרקטית - תרומה
מחלקה בנק - רמקה נכנה סמל, תרומה לאלה בסך וכו'..
אין דבר כזה אובייקט תרומה מסל עצמו!

typeid.name - מחזיר לי את שם הטיפוס ובכך ניתן לבדוק בין משתנים.

Scanned by CamScanner

כתיבה וקריאה לקבצים

← קובץ מאפשר לשמור את המצב בזמן ריצה לחסכון אותו בפעם מועדת שנרצה
ofstream - משמש לכתיבה
ifstream - משמש לקריאה

← פתיחת קובץ נעשה בדרך עי קונסטרקטור ומתודת המחלקה
is_open מתודה שמחזירה T/F אם נפתח הקובץ או לא
בעזרת בטל התוכנית סודם את הקובץ כדי שלא להשאיר לכתוב שם

קובץ בינארי

בקובץ בינארי נשמרים מספרי הטיפוס המוגדרים את השתנה ולאחר מכן
השתנה נשמר באופן בינארי (אפסים ואחדים)
הקובץ הבא למעלה כדאי כמה ומאפשר לנו לחסוך מקום
← סריליזציה
מתודה שמאפשרת לשמור ולהעלות מידע של אובייקט הכל נשמר באופן בינארי
לא משתנה שעבורם מתבצעת הקבצה דינאמי

Scanned by CamScanner

ירושה מורחבת

איזכון ירושה

← Private - אם ירשנו מחלקה שהוא Private כל המידע במחלקה
הופך לחיל פרטי עם אם סדרך המחלקה הזו מודל אחרת.

← Protected - מאפשר לשמר את המחלקה שירשתי כפי חלק

	Private	Protected	Public
Private	Private	Private	Private
Protected	Private	Private	Protected
Public	Privat	Protected	Public

בעיה בירושה מרובה

A ← B, A ← C, B ← D, C ← D

⊛ בכמות איכון

במקרה זה ירשנו פעמיים את A
הבעיה תהיה בקריאה בהמשך לA, לאיזה חלק A פונים?!
↓
הפתרון יהיה ירושה וירטואלית
כל המחלקה שיורשת את אותו הלא ירשו אותו באיפון וירטואלי
ובנוסף מתווסף פוינטר לטיפוס שירשנו
בעת ירושה למחלקה הורשת יהיו כפולות של פוינטרים, אבל נבא לראיין
בעיה כי הם מצביעים לאותו אב קדמון.

Scanned by CamScanner

Template

פונקציה Template

זהו מנגנון אוטומטי המאפשר לשלב טיפוס מבלי לכתוב בכל פעם את אותה הפעולה.

לדמה שכבר הכרנו: • virtual • define

← דוגמא:

```
template <class T>
T& max (T&a, T&b) {
    return (a<b)?b:a;
}
```

→ שורה שחייבת להיות לפני כל פונקציה שמוגדרת כ template

מה שקורה בפועל שנעמן את הפונקציה נשלח משתנים שהם מטיפוס מוגדר ובזמן הקומפילציה הקומפיילר יתאים את פיסת הקוד הכללי לטיפוס ששלחנו בכל מקום שלפני T יהיה רשום הטיפוס ששלחנו.

← בדרך שמסייעת את הפונקציה פועלת אחת עם סוג טיפוס הזו ושמרה וגם נשלח שוב בזמן עם אותו טיפוס זה לא ישוכפל מרגל.
כלומר מספר הפעמים שהפונקציה נכתבת הוא כמספר סוגי הטיפוסים.

← הפונקציה נכתבת בקובץ h כיוון שהוא נבנה בזמן הקומפילציה

← קיימת הגבלה סמוך בכתיבת פונקציות template כמו לשל מימוש אובייקטורים עבור משתנים פרמיטיביים
אבל אם הפונקציה פועלת עם טיפוסים שאינו בהם לדוגמא אופרטורים.

← כשצמנים פונקציה עם המשתנה שאינו רוצה הכתיבה func<int>(5)

← ההבדל בין template לאלגוריתם רגיל:
• בעת שמתבצע אלגוריתם שמתאים לשפה מסוימת בלבד הוא כתוב ומשתמש במילת virtual.
• כאלגוריתם שיכול להתאים לשפות שונות ומשתמש ב template

Scanned by CamScanner

	פולימורפיזם	Template
גודל הקוד	הקוד מופעל פעם אחת בהגדרת המחלקה / הבסיס	הקוד המקומפל מתפרס ומשוכפל בהתאם לסוגי הטיפוסים
זמן ריצה	Dyamic func binding איטי יותר - נעשה בזמן ריצה	Static func binding מהיר יותר - נעשה בזמן הקומפילציה
שימוש	יותר גמיש - אפשר לרשת מלק מהגדרת ולשנות מה שלא מתאים.	פחות גמיש - אין מימוש של ירושה שחוסך כתיבה של קוד

מחלקת template

נשתמש כאשר יש לי אותה מחלקה שמבצעת את אותו התהליך כמו מחלקה אחרת.

← מעל ההגדרה נכתב: template <class T>

← הגדרה על אובייקט ב main: Myclass <int> MyObject (100)

שם המחלקה | טיפוס המחלקה | שם אובייקט | פרמטר לקונסטרקטור

← לפני כל מחלקה צריך לציין template<class T>

← אפשר להגדיר יותר ממחלקה אחת template <class T, class G>

← המטרה היא ליצור מבנים חדשים שיתאימו לטיפוסים שונים כמו: מחסנית, רשימה, וקטור.

← ירושה של מחלקה template ניתן לעשות:

הגדרה מפורשת של טיפוס המחלקה

ירושה template ללא הגדרה ספציפית של טיפוס

Scanned by CamScanner

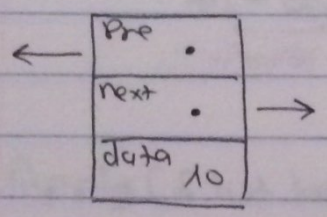
אלגוריתם ומבנים נרחבים

STL = Standart Template Lib...

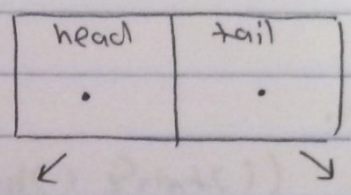
iterator

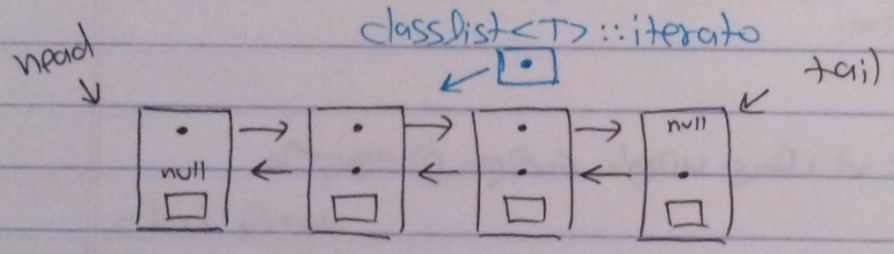
האיטרטור הוא למעשה מעין פוינטר "מעורבב"
עבור מחלקת איטרטור המכילה מימושים עבור פוינטר כדי אופרטורים מתאימים
אשר מותאמים לטבעה המתרחש עמו עובדים - לכל מבנה מחלקת איטרטור שונה.
← ניתן ליצור מספר איטרטורים כדי שגם מחלקה
להחזיר מספר אובייקטים שלה.
← דוגמא טובה - רשימה מקושרת:

class item<T>



class list<T>





★ ניתן לראות שאיטרטור הוא מעין "פוינטר חכם": כלומר מאפשר לו להתנהג
ולעבוד עם כל מבנה נתונים במערך.
בעצמו אינו יוצר תלויים במבנה הנתונים.

כל מבנה נתונים דרוש להמציא אותו אופן דרך שמאפשרת לעבור על כל האובייקטים
במבנה מבלי לחשוף אותו ← איטרטור זו הדרך.

Scanned by CamScanner

Object Function

העברת פונקציה כפרמטרים.
למעשה בדומה מדבר struct / מחלקה אשר מממשת מחלקה תבניתית
שמקבלת פונקציה שאותה היא מפעילה על כל השתנים, לכן היא יכולה
מוגדרת כ-template

← דוגמא:

```
template <class iterator, class operation>
void Apply (iterator begin, iterator end, const operation& f)
    for (; begin != end; ++begin)
        f(*begin)
```

פונקציה

פרמטר לפונקציה

דוגמא לשימוש:

```
Apply(f.list.begin(), f.list.end(), Print())
```

אובייקט זמני
וחסר שם

אז

① הצהרה על שמה מטפס המחלקה של הפונקציה

```
class Print
Print MyPri
```

② שימוש : MyPri

← עבור פונקציה רגילה דרך לממש אופרטור סוגרים ()

Scanned by CamScanner