שפות תכנות – הקדמה

מהי שפת תכנות?
כלי לשוני בעל תחביר רשמי (סינטקס) ומשמעות (סמנטיקה).
מוצר מלאכותי שמעוצבן ביודעין כך שיוכל להתממש על מחשב.
עולם תפיסתי (קונספטואלי) למחשבה על תכנות.

בלשנות התכנות
כמו בשפה רגילה: תחביר ומשמעות.
בשפה טבעית טווח ההבעה גדול בהרבה – מדויק יותר ברור ושנון יותר.

מה נדרש משפת תכנות?
אוניברסליות – לכל בעיה צריך להיות פתרון. (למשל באמצעות רקורסיה מפורשת).
בר יישום – ללא סימנים מתמטיים מסובכים וללא שפה טבעית.
יעילות – נתון לויכוח מתמיד – איזו שפה יעילה יותר.
השפה צריכה יכולת ביטוי טובה – למתכנת צריך להיות קל לרשום כל רעיון באמצעות השפה. יחד עם זאת, השפה צריכה להיות יעילה (למשל מימוש באמצעות רקורסיה עלול להיות לעיתים מאוד לא יעיל, כמו למשל בחישוב סדרת פיבונצ'י).
פשטות – כמה שפחות כללים בשפה – לפעמים על חשבון הנוחות – למשל בפסקל, יש שלושה סוגי לולאות.
אחידות ועקביות הרעיונות – למה למשל, לולאת FOR בפסקל, מאפשרת לרשום בתוכה הוראה אחת בעוד שלולאת REPEAT יכולה לקבל מספר הוראות?
הפשטה – השפה צריכה לאפשר להסיר תבניות חוזרות. (אין הגיון בכתיבת אותה פונקציה פעמיים, פעם אחת עבור מספרים שלמים ופעם שניה עבור מספרים ממשיים).
בהירות לאנושות – מדוע הסימנים "=" ו "==" ב C כל כך דומים אף על פי שהמשמעות שלהם כל כך שונה?
הסתרת מידע ומודולריות – כאשר עובדים בצוות תכנות גדול, חשוב שיהיה אפשר לחלק את העבודה לחלקים נפרדים – מחלקות למשל, כאשר כל מחלקה עומדת בפני עצמה.
בטיחות – הקומפיילר יכול לזהות שגיאות שכתב המתכנת.

מדוע RANGER 3 לא הגיעה לירח?
הגדרות מילוניות גרועות – אין חשיבות לרווחים.
אין הצהרת משתנים – הקומפיילר מקצה משתנים באופן אוטומטי בלי הצהרה מפורשת עליהם מצד המתכנת.
טיפוסי משתנים מרומזים – הקומפיילר מנתח בעצמו את מה שכתב המתכנת ומסיק מכך לאילו טיפוסים התכוון המשורר.
מרחב מחייה של משתנים – המשתנים לא מתים בסוף הבלוק שבו הוגדרו ומוכרים בכל מקום.
שליטה עלובה בפירוט המבנים - ???
מחסור בכלי אבחון – דיבאגרים מוצלחים לדוגמה.

מה מאפיין שפת תכנות?
תחביר רשמי ומשמעות.
רעיונות:
- כיצד היא מטפלת בערכים (VALUES).
- כיצד היא מזהה טיפוסים (TYPING).
- הישויות שלה לאיחסון ערכים. (STORAGE).
- האמצעים שלה לאיחסון ערכים. (COMMANDS).
- כיצד היא מנהלת את השליטה (SEQUENCERS).
- כיצד היא מספחת שמות לערכים (BINDING).
- כיצד היא מאפשרת הכללה (גנריות) (ABSTRACTION).

פארדיגמה (דוגמאות):
- שפות ציוויות (IMPERATIVE PROGRAMMING). – פורטרן, קובול, סי, פסקל, אלגול.
- תכנות בו זמנית (CONCURRENT PROGRAMMING). – ADA, OCCAM, PAR-C
- תכנות מונחה עצמים – Small talk, Self, C++, Objective-C, Object Pascal, Beta, CLOS
- תכנות פונקציונלי – Lisp, Scheme, Miranada, Ml
- תכנות לוגי – Prolog, Prolog-dialects, Turbo Prolog, Icon
- תכנות מאולץ – CLP, DLP

<u>טיפוסים וערכים:</u>

<u>סוגי טיפוסים וערכים:</u>

- פרמיטיביים, (INT ,REAL), מרוכבים (מחלקות למשל) ורקורסיביים (למשל רשימה מקושרת).
- סדרת ייצוג תאורתית.
- הצגה רב איברית של טיפוסים.

<u>ערכים:</u>
ערך הוא ישות שקיימת בזמן החישוב.
הגדרה תפעולית (בפסקל): כל מה שניתן להעביר כארגומנט לפונקציה או לפרוצדורה.

<u>דרכים לחלק ערכים למחלקות בשפה:</u>

- על פי המבנה שלהם (כיצד הם מוגדרים).
- על פי הפונקציונליות שלהם (כיצד הם מתופעלים).

<u>מבני ערכים:</u>

- ערכים פרמיטיביים: כל מה שלא מורכב מערכים אחרים: TRUE ,FALSE, אותיות, מספרים שלמים וממשיים ומצביעים.
- ערכים מרוכבים: רשומות (RECORDS), מערכים, סדרות וקבצים.

קבוצת הערכים שהשפה יכולה להשתמש בהם מובנית <u>מבסיס</u> שהוא הערכים <u>הפרמיטיביים</u> ומכל חיבור ביניים באמצעות ערכים מרוכבים. מבחינה מושגית – יש אין סוף ערכים מרוכבים.
הדרכים ליצור את הערכים המרוכבים בשפה, בדרך כלל <u>בלתי תלויות</u> במימוש שלה.

<u>תפעול ערכים:</u>
הפעולות שניתן לבצע על ערכים (מלבד העברתם לשגרות):

- השמתם בתוך משתנים ואיחזור ערכם מתוך המתשנים.
- שימוש בערכים ליצירת ערכים מרוכבים.
- החזרת ערך מתוך שגרה.
- שימוש בערך בתוך חישוב ערך של ביטוי.

<u>ערך שאפשר לבצע עליו את כל הפעולות הנ"ל נקרא "ערך ממחלקה ראשונה".</u>

<u>ערכים בפסקל:</u>
<u>ערכי מחלקה ראשונה:</u> רק ערכים פרמיטיביים – ערכי אמת, אותיות, ערכים מספריים (ENUM), שלמים, ממשיים ומצביעים.
<u>ערכי מחלקה שניה:</u> ניתן להעביר כארגומנטים לשגרות, אך לא לאחסן, להחזיר או להתשתמש כרכיבים בערכים אחרים: references to variables וגם פונקציות ופרוצדורות.
<u>ערכים מרוכבים</u>: רשומות, מערכים, סדרות וקבצים – לא ניתן להחזיר!

<u>ערכים ב ML</u>.
כל הערכים ב ML הם ממחלקה ראשונה:
<u>ערכים פרמיטיביים</u> – ערכי אמת, שלמים, ממשיים ומחרוזות.
<u>ערכים מרוכבים</u> – רשומות, TUPLES, תבניות, רשימות ומערכים.
<u>פונקציות מופשטות</u>
<u>Refernces to variables</u>

<u>מה ניתן לבצע ב ML שלא ניתן לבצע בפסקל?</u>
ניתן לכתוב פונקציה שמקבלת פונקציה F : A -> A ומחזירה הרכבה של F על עצמה.
ליצור רשומה שמכילה בתוכה שתי פונקציות.

<u>ביטויים:</u>
ביטוי הוא חלק מתוכנית שמתורגם לערך בזמן החישוב.
לדוגמה: 3.1415, '%', 'SHIR' (Pascal).
Sqr(4)
If leap (thisyear) then 29 else 28 – בשפת ML.

הצורך בטיפוסים:
בשפת מכונה כל הערכים הם בסך הכל ביטים של אפס ואחד. הם חסרי טיפוסים.
בשפות אסמבלי (כמו ה11 – PDP שנלמד באת"מ), כבר יש הבדל בין כתובות לבין נתונים.
- הסכום של שתי כתובות עדין לא מוגדר כהלכה: הדבקת תוויות שמוסיפות מידע לערכים בזמן ריצה.
- בעיה: בדרך כלל מתעסקים רק בטיפוסים שהוגדרו מראש אך לא עם טיפוסים שהוגדרו ע"י המשתמש.

שימו לב: טיפוס הוא גם תכונה של:
- ביטוי – מאילו טיפוסים הערכים מחושבים.
- תאי זיכרון – היכן ערכים בעלי טיפוסים מאוחסנים.

שפות גבוהות: מתאימות טיפוסים גם לביטויים בזמן הידור (static typing) וגם בזמן ריצה (dynamic typing).

טיפוסים כסדרות של ערכים:
אם ערך V הוא מטיפוס T אז V הוא בקבוצה ש T מגדירה.
אם ביטוי E הוא מטיפוס T אז התוצאה של חישוב E היא ערך מטיפוס T.
אבל: לא ככל קבוצה של ערכים מתאימה לטיפוס!
לדוגמה {יום שישי, 13, "אוקטובר"} לא מתאימה לשום טיפוס בשום שפה.
{שקר, אמת} מתאימה לטיפוס בהרבה שפות.

מסקנה: ההגדרה של קבוצה מטיפוס היא החלטה מעשית, המבוססת על מטרה של השפה ולא על התכונות של הערכים שלה.

טיפוסים:
טיפוסים פרמיטיביים: כאלו שמכילים רק ערכים פרמיטיביים.
טיפוסים מרוכבים: טיפוסים שיש להם הרבה ערכים מרוכבים.
טיפוסים רקורסיביים: טיפוסים שיש להם הרבה ערכים שמורכבים גם מערכים מאותו טיפוס וגם מערכים מטיפוסים בסיסיים אחרים. (למשל רשימה מקושרת שבנויה מטיפוס שמכיל בתוכו רשומה שכוללת ערך ומצביע לאיבר הבא ברשימה).

טיפוסים פרמיטיביים: לא ניתן לפרק אותם לתת ערכים (כפי שניתן למשל לבצע על רשומות).
מתחלקים לשתי קטיגוריות:
- אלמנטריים – כל מה שמובנה בשפה.
- לא אלמנטריים – כל מה שמוגדר ע"י המשתמש. למשל טיפוס שיכול להחזיק את שמות החודשים. או בפסקל, תת טווח – למשל טיפוס שיכול להכיל מספרים שלמים, אך רק בטווח 1 עד 31.

הטיפוסים האלמנטריים הפרמיטיביים של השפה אומרים הרבה על היעוד המכוון מראש של שפת התכנות:
פורטרן – חישובים מתמטיים. לכן יש בה דיוק של מספרים ממשיים ושל מספרים מרוכבים.
קובול – עיבוד נתונים. לכן יש בהם מחרוזות ומספרים עשרוניים.
סנובול – עיבוד מחרוזות. מחרוזות אורכי משתנים. ???

מקור לבלבול: שפות שונות משתמשות בשמות שונים עבור אותם טיפוסים פרמיטיביים.
פסקל: Boolean, Integer, Real
ML: bool, int, real

מקור לקושי: מימוש שונה של השפה עלול להשתמש בקבוצות שונות של ערכים עבור אותו טיפוס.

ההבדל בין שמות לערכים: הערך המספרי שלהם והזהות שלהם שונה:
המזהה **דצמבר** - ניתן להגדירו מחדש. אולם הערך **דצמבר** תמיד יהיה "זה שבא אחרי (נובמבר)".

טיפוסים פרמיטיביים בעלי יחס סדר:
טיפוסים מסודרים – טיפוס שיש סדר כולל ביחס בין ערכים שלו. שימושי בתור איטרטור וביטוי מותנה.
טיפוסים ללא סדר – מרוכבים (למשל בפורטרן).
טיפוסים לא רציפים (בני מנייה) – טיפוס מסודר שניתן להתאים אחד לאחד, את הערכים שלו, לערכים טבעיים. למשל, ערך בולאני, CHAR, INTEGER.
טיפוסים מסודרים לא רציפים – מחרוזות וממשיים.

פסקל: ניתן להשתמש רק בטיפוסים בני מנייה בהרצה של לולאה או במיקום בתוך מערך (מסיבה הגיונית) וכמו כן ב CASE (מסיבה שרירותית).
C++: ב TEMPLATE ניתן להשתמש רק בטיפוסים בני מנייה.

טיפוסים פרמיטיביים למחצה:
סוג ראשון:
- אטומיים: ערכים שלא ניתן לפרק לערכים מטיפוסים קטנים יוצר.
- לא אלמנטריים: מוגדרים ע"י המתכנת. לדוגמה: ENUM ב C.

סוג שני (טיפוסים פסאודו פרמיטיביים): די נדיר למצוא כאלה.
- לא אטומיים: ניתן לפרק אותם לטיפוסים קטנים יותר.
- אלמנטריים: מובנים בתוך שפת התכנות. (למשל STRINGS בטורבו פסקל שניתן לחלקם לתווים בודדים).

מחרוזות:
סדרה של תויים. שימושי כטיפוס נתונים. נתמך בצורה כזאת או אחרת בכל שפות התכנות המודרניות.
ML: טיפוס פרמיטיבי מכל אורך. ניתן להשוות בין שתי מחרוזות, לפרק ולשרשר. הכל בנוי בשפה.
פסקל ואדה: מערך של תוים. כל הפעולות שניתן לבצע על מערכים זמינות. החסרון: כל פעולות המחרוזות חייבות להיות מוגדרות בתנאים של מחרוזות קבועות מראש ???
אלגול 68: כמו פסקל ואדה, אך עם מערכים גמישים, שניתן לשנות את אורכם בזמן ריצה.
C: מערכים גמישים למחצה: מחרוזות ליטרלים.
פרולוג: רשימה של תווים. ניתן לבחור רק את התו הראשון. כל שאר הפעולות על מחרוזות צריכות הגדרה מפורשת.

ערכים מרוכבים:
ניתן ליצור טיפוסים מרוכבים מתוך טיפוסים פרמיטיביים ומתוך טיפוסים מרוכבים שהוגדרו לפניהם.
TUPELS, רשומות, משתנים, איחודים, מערכים, קבוצות, מחרוזות, רשימות, עצים, קבצים סדרתיים, קבצים ישירים, יחסים וכדומה.

ערכים ממחלקה ראשונה וממחלקה שניה:
ערך שכל פעולה ניתן ליישום עליו נקרא ערך ממחלקה ראשונה.
בפסקל, פונקציות ופרוצדורות הן ערכים ממחלקה שניה. זאת מכיוון שאף על פי שניתן להעביר אותן כפרמטרים לשגרות, לא ניתן להקצות אותן או להשתמש בהן כרכיבים בטיפוסים מרוכבים.
ברוב השפות הציוויות (פסקל, סי וכו') הן ממחלקה שניה, במטרה להימנע ממספר בעיות מימוש.
לעומתן, ב ML ובמירנדה, נמנעים מההבדלים הללו (ולכן ניתן לעשות עם פונקציות כמעט הכל).

עקרון שלמות הטיפוסים:
המחלקה הראשונה והשניה הן שני ניגודים. לדוגמה, פסקל לא מרשה לפונקציה להחזיר ערך מרוכב, בזמן שC מאפשרת להחזיר מבנים (רשומות), אך לא מערכים.
עקרון שלמות הטיפוסים: שום פעולה לא צריכה להיות מוגבלת באופן שרירותי לטיפוסים של הערכים שמעורבים בה.

האם ערכים מרוכבים הם ממחלקה ראשונה? (רשומות, מערכים וכו')
אין עקרון שימנע מהם להיות ממחלקה ראשונה, אך בהתחשב במימוש לפעמים זה נמנע מהם.
פסקל: ערכים מרוכבים:
- ניתן להעבירים כפרמטרים, גם על פי ערכם וגם By reference.
- ניתן לאחסנם במשתנים.
- לא ניתן להגדיר כקבועים.
- לא ניתן להשתמש כערכי החזרה של פונקציות.

C הישנה:
- לא ניתן להעביר כפרמטרים (בסי אפשר להעביר פרמטרים רק בערכם). השיטה להעביר ערכים מרוכבים היא באמצעות העברת מצביע אליהם.
- ניתן לאחסן ערכים מרוכבים בתוך משתנים.
- לא ניתן ליצור קבועים מרוכבים.
- לא ניתן להחזיר אותם מפונקציות.

ANSI C: מתנסה לתקן הרבה מהבעיות של C. עדין יש רק העברת פרמטרים על פי ערכם.
- כעת ניתן להעביר מבנים על פי ערכם. יש קומפיילרים (כמו של בורלנד) שמאפשרים למתכנת להתקין דגל אזהרה כך שכל פעם שמשתמשים באופציה הזאת, הקומפיילר יזהיר אותו.
- מערכים: עדין לא ניתן להעביר על פי ערכם או להחזיר אותם. לא ניתן לתקן זאת בגלל השקילות בין מערכים ומצביעים ב C.
- קבועים: מילת ה CONST החדשה לא נועדה לקביעת שמות לערכים. היא מיועדת בשביל להכריז על מקום שבו ישמר מידע שלא ישתנה.
- טריק: ניתן להעביר מערכים ע"פ ערכים ע"י כך ששמים אותם בתוך מבנה.

רפרנסים מתנדנדים Dangling Refernces
לדוגמה – קביעת רפרנס למשתנה מקומי מתוך מצביע גלובלי. לאחר היציאה מהפונקציה שבה הוגדר המשתנה המקומי, המצביע הגלובלי יצביע ל"זבל".
פסקל פותרת את הבעיה הזאת בכך שהיא לא מרשה הצבעה אל משתנה מקומי מתוך אף משתנה אחר.

לו היה ניתן לשמור פונקציות בתוך משתנים בפסקל, אז היה ניתן לציור רפרנס מתנדנד אף על פי שלא היתה הגדרה מפורשת של הצבעה על משתנה מקומי. פשוט היתה העתקה של פונקציה מקומית שכוללת בתוכה שימוש במשתנה מקומי שכבר לא קיים.

C: ניתן לשמור פונקציות בתוך משתנים, אבל אין פונקציות מקוננות.
GNU C: מאפשרת קינון פונקציות. האחריות של המתכנת.
++C: חייבות להיות פונקציות מקוננות!
- מחלקות כוללות פונקציות חברות.
- מחלקות הן בדיוק כמו מבנים.
- ניתן להגדיר מבנים בכל מקום, אפילו בתוך פונקציות.
- פונקציות חברות מוגדרות בתוך פונקציות והן מקוננות.
- ניתן להגדיר מחלקות עם פונקציות חברות בתוכן, בתוך כל פונקציה חברה, כך שקינון שרירותי אפשרי.
- הפתרון: פונקציות מקוננות לא רשאיות לגשת אוטומטית למשתנים של הפונקציה העוטפת.

GNU C++: אין פונקציות מקוננות רגילות.

מסקנות:
- התרה של פונקציות מקוננות וגם פונקציות שהן ערכים ממחלקה ראשונה עלולה להוביל לרפרנסים מתנדנדים שיתגלו רק בזמן ריצה.
- הפתרון הוא באמצעות הגבלות בתחביר (הגבלות סינטקטיות): פונקציות אינן ערכים ממחלקה ראשונה (כמו בפסקל) או שאין פונקציות מקוננות (C) או שאין לפונקציות המקוננות גישה למשתנים של הפונקציה העוטפת (C++).
- פתרון מסוג אחר: יש רק משתנים גלובלים (כמו שפות פונקציוליות).

ביטויים: ביטוי הוא חלק מתוכנית שניתן לחשב אותו ולחלץ ממנו ערך.
ליטרלים הם הביטויים הפשוטים ביותר – ערכים ברורים ביותר מטיפוס מסויים.
פסקל: שלמים: 365, ממשיים: 3.1415, תוים '?', מחרוזות: 'שפות תכנות זה קורס מעצבן'.
ב SMALLTALK 7/9 הוא ליטרל (אובייקט של מחלקת שברים).

ביטוי יכול לגשת לערך של משתנה או של קבוע שמי.
פסקל: const pi = 3.1415; var r: real;
ואז הביטוי r * pi * 2 יערב גישה גם לקבועים וגם למשתנים.
השפה צריכה גם לאפשר גישה לרכיבים של ערכים מרוכבים. למשל בפסקל V[i] וגם V.I משמשים לגישה למשתנים, אבל אין ציון זהה לגישה לקבועים. ניתן לגשת לתו מסוים במחרוזת השמורה בתוך משתנה באמצעות הסוגרים המרובעים, אך לא ניתן לבצע זאת עבור מחרוזת השמורה בקבוע CONST.
באדה ובטורבו פסקל הבעיה נפתרה – ניתן לגשת באותה צורה גם לקבועים וגם למשתנים.

ביטוי מצרפי: ביטוי שמייצר ערך מרוכב. הערכים הפנימיים נקבעים באמצעות תת-ביטויים.
ML: TUPLE, רשומות ורשימות.
C ואדה: מערכים (רק באיתחול) ורשומות.

פסקל: בלתי אפשרי.
C++: אפשרי רק אם הוגדר קונסטרקטור במפורש עבור המחלקה / מבנה.

ביטויים מותנים:
ישנם מספר תת ביטויים, אולם בכל פעם, רק אחד מהם נבחר וערכו מחושב. לא קיים בפסקל ובפורטרן ובאדה.
ב C זה מוגבל לאופרטור :?.
ב ML השימוש בזה נפוץ, בכל פונקציה כמעט.
באלגול 68 ניתן אפילו לקבוע מי יקבל את ערך הביטוי: (if u>v then v else u) := u+v;

קריאות לפונקציות:
ברוב השפות שם הפונקציה צריך להיות מזהה. בשפות בהן הפונקציות הן ממחלקה ראשונה כמו ML:
(if x>y tehn sin else cos) (x)
אופרטור הוא בדיוק כמו פונקציה, מלבד העובדה שהוא נרשם בין שני ארגומנטים.
בפסקל וב C OVERLOADING מוגבל רק למספר אופרטורים ופונקציות שמובנים בשפה.
ב C++, אדה, ML: קריאה לאופרטורים היא בדיוק כמו קריאה לפונקציה. לכן זה קל ללמידה ומאפשר למתכנת להגדיר מחדש אופרטורים.

פעולות על מחרוזות:
כמה פעולות נפוצות: העתקה (=:), מבחן השוואה (=), סדר מילוני (<), שרשור (+).
כמה פעולות פחות נפוצות: חזרה: "ho"*3 = "hohoho"
חישוב אורך, מציאת התו בעל הערך המינימאלי במחרוזת.
המרה מטיפוסים אחרים למחרוזות היא חשובה מכיוון שהוצאת פלט היא בדרך כלל של מחרוזת, כך שכל טיפוס צריך המרה למחרוזת.
פעולות מסובכות יותר: חילוץ תת מחרוזת, חיפוש תו במחרוזת, חיפוש מחרוזת בתוך מחרוזת.
חילוץ מתוך מחרוזת של טיפוסים שונים, כמו SSCANF של C.
המרה ממחרוזת אחת למערך ע"י בחירת תו מפצל.