

שיטות מתמטיות בגיאופיזיקה – מבוא לסביבת העבודה

מטרה: המסמך נועד לתת רקע והסברים על סביבת העבודה בה תכתבו תוכנות במהלך הקורס, על מנת להקל עליכם בעבודה.

המסמך רשום בלשון זכר, אך מיועד לכל המינים.
המסמך גם כתוב בעברית, אך מיועד לדוברי כל השפות.

אתר הקורס:

<http://www.tau.ac.il/~mosher/0341-2218-01>

עוזר הוראה:

שם: Pavel Sinitsyn

חדר: 601

טלפון: 6083

[-emailpavelsin@post.tau.ac.il](mailto:pavelsin@post.tau.ac.il)

מה זה יוניקס?

יוניקס היא בסה"כ מערכת הפעלה (להלן מ"ה). מ"ה היא תוכנה אשר מטרתה לנהל את משאבי המחשב באופן כזה שכל המשתמשים במחשב יקבלו גישה יעילה והוגנת עד כמה שניתן למשאבים. משאבים הם לדוגמא – דיסק, תקשורת, מעבד וזיכרון. מבחינת המשתמש, מ"ה היא נותנת השירותים. כאשר אני רוצה להריץ תכנית, לכתוב קובץ לדיסק או להדפיס מסמך, אני "מבקש" ממ"ה שתבצע לי את הפעולה.

השוני בין מ"ה יוניקס לבין מה שרוב המשתמשים הרגילים בעולם מכירים (קרי – windows), הוא שמה שמייקרוסופט ייצרו איננו מ"ה לפי הגדרה. העובדה ש-windows מצליחה רוב הזמן להפעיל את החומרה של המחשב, בד"כ בלי להתרסק אם לא מבקשים ממנה יותר מדי, וכן יכולתה העסקית של החברה להרוג את מתחריה כך שמכירותיה עולות על איכות מוצריה, אינן מכשירות את windows להיות מ"ה. לא ניכנס לזה.

מערכת היוניקס הראשונה נכתבה במעבדות של Bell laboratories בשנת 1969 ע"י בחור נחמד בשם Ken Thompson, שבסה"כ רצה להריץ תכנית שעושה סימולציה של תנועת כוכבי לכת בלי להתעסק עם המחשבים המפלצתיים של IBM. הוא ייצר אותה עבור מחשב "קטן" של חברת Digital בשם PDP-7. מאותה נקודה התחילה המהפכה שהובילה להשתלטות של "המחשבים הקטנים", בהתחלה רק באוניברסיטאות, ואח"כ לשאר העולם. היכולת להריץ מ"ה יעילה על מחשב קטן יחסית למה שהיה אז (IBM הייתה היצרנית הגדולה של ה-MainFrames – מחשבים ענקים, כמו שרואים בתמונות מפעם) נתנה לתכניתנים סביבה נוחה וקלה לתפעול. ישנם הרבה סוגים (המכונים flavors) של מערכות יוניקס, שנכתבו ע"י חברות שונות עבור החומרה הספציפית של המחשבים שלהן. רשימה חלקית – HPUX, Sun Solaris, IBM-AIX, SGI, Unisys, – Sco-Unix.

אז מה זה לינוקס?

לינוקס הוא אחד מה-"טעמים" של היוניקס. הוא נוצר לראשונה ע"י בחור פיני טוב-לב בשם Linus Torvalds, ומכאן נגזר שמה של מ"ה. בניגוד להרבה מערכות הפעלה, לינוקס ייחודית בכך שהיא Open-Source, כלומר התוכנה של מ"ה פתוחה לכולם. כל אחד יכול להשיג אותה, ולראות איך היא כתובה. יתרון אחד גדול של שיטה זו הוא שהרבה אנשים יכולים בהתנדבות לתרום דברים למ"ה, לכתוב רכיבים חדשים המתממשים עם מ"ה, ובעיקר – למצוא באגים. אה, כן – הכי נחמד שזה חינם, או לפחות לא עולה מאות שקלים להתקנה של משהו שלא עובד טוב.

תפיסת השימוש במערכת יוניקס

הגורמים הראשיים שדרבנו את פיתוח יוניקס, הם הצורך ליצור מ"ה ותוכנה תומכת שיהיו פשוטים, אלגנטיים, וקלים לשימוש. אלגנטיות בהקשר זה פירושה בד"כ סגנון תכנות טוב, וניהול חסכוני של המשאבים. תכנות אלו של עיצוב הנחו את מפתחי יוניקס, לקראת העקרונות הבאים:

1. דאג לכך שכל תכנית תעשה דבר אחד, בדרך הטובה ביותר. תכניות פשוטות אלו קרויות לעיתים קרובות כלים (tools).
2. תכנן שהפלט של כל תכנית יהפוך לקלט של תכנית אחרת שאיננה ידועה עדיין. גישה זו תגרום לכך שכלים פשוטים יתחברו זה לזה כדי לבצע עבודות ומשימות מורכבות.
3. אל תהסס לבנות תכניות חדשות כדי לבצע עבודה כלשהי. ספריית הכלים הולכת וגדלה ונותנת לך שירותי עזרה.

התוצאות של עקרונות אלו היא שאנשים אומרים לפעמים כי מערכות יוניקס מגשימות את האמירה של שומאכר ש-"קטן הוא יפה". כל תכנית יוניקס היא קומפקטית, קלה לשימוש ומבצעת ביעילות את משימתה.

למה הולך להיות לכם קשה בקורס

מספר סיבות:

1. אתם לא יודעים איך לתכנת. קורס "תכנות לפיזיקאים" שניתן בשנה א' מכין אתכם לתכנות בערך כמו שטירונות 02 מכינה חייל לירות במשהו. מי שכן יודע לתכנת מתבקש לעזור לחבריו.
2. החומר בקורס הוא לעיתים מסובך להבנה, ואין הרבה זמן לחזרות.
3. יש הרבה עומס מקורסים אחרים. זה הסמסטר הקשה ביותר של התואר.
4. התרגילים גוזלים זמן רב גם ממי שכן יודע לתכנת.
5. סביבת העבודה היא קשה לרוב האנשים הרגילים להיות משתמשים פשוטים של windows. בעיקר אנשים מופתעים מכך שהמקלדת יותר מהירה מהעכבר.

למזלכם, שפת התכנות היא C, שהיא השפה הכי טובה בעולם. למעשה, מ"ה יוניקס כתובה בעצמה ב-C. זו שפה עם תיעוד רב, והרבה ספרי הדרכה בעברית. הספר הבסיסי והנוח ביותר הוא: "The C programming language", Kernighan & Ritchie, 2nd ed.

איפה עובדים

התכניות שתכתבו צריכות לרוץ על גבי מחשבי הלינוקס שבכיתת המחשבים של גיאופיזיקה (הידועה בכינויה "שולחן עגול"). יש שם 6 עמדות לינוקס, ששמותיהן class1219-1 עד class1219-6. הכניסה למחשבים עם המשתמש האוניברסיטאי שלכם (כמו שנכנסים ל-webmail). התכניות צריכות להימצא בספריית הבית שלכם, תחת ספרייה שתייצרו לכל תרגיל. רצוי מאוד מאוד (מאוד) לא לכתוב תכניות בבית, להעביר אותן במייל לשרתי הלינוקס, ואז לצפות שהן יתקמפלו וירוצו. הסיבות הטיפשיות לכך הן:

1. מייקרוסופט החליטו שבסוף כל שורה של טקסט בקובץ הם מוסיפים תו מיוחד, אשר לא משתמשים בו בקבצי טקסט של אף מערכת הפעלה אחרת. זה עלול לבלבל את הקומפיילר (מיד נסביר מה זה קומפיילר) שינסה לקרוא את הקובץ שהבאתם מהבית.
2. הקומפיילר שיש לכם בבית עלול לא להתריע על שגיאות שהקומפיילר שקיים על הלינוקסים כן יתריע.
3. ישנן ספריות דינאמיות שצריכות להתלנגג' (מיד נסביר מה זה לינקוג') אל התכנית שלכם בזמן ריצה. למרבה הפלא, ב-windows הן נמצאות במקומות לא סטנדרטיים.

מה שכדאי לעשות הוא לעבוד בשולחן העגול, או להתחבר מהבית אל המחשבים בשולחן עגול. **יש הסבר על גישה אל המחשבים מחוץ לאוניברסיטה באתר הקורס.**

אם בכל זאת אתם מעדיפים להביא קבצים שאותם כתבתם בבית דרך המייל, תפעלו באופן הבא: היכנסו לאחת מעמדות הלינוקס. יש למטה במסך אייקון של כדור שמחובר לעכבר. תקליקו עליו. זה יפתח את firefox, שהוא ה-internet browser. דרכו גשו לחשבון המייל אליו שלחתם את הקובץ, והורידו את הקובץ אל המחשב. הקובץ ככל הנראה ירד אל ספרייה בתוך ספריית הבית שלכם, שנקראת Desktop. העבירו את הקובץ אל הספרייה בה אתם רוצים שהוא יימצא (אל תעבדו בתוך ספריית Desktop, כי כל מה ששם נמצא על המסך, וזה יעשה לכם בלגאן). כעת נסו לקמפל, וקוו לטוב.

איך עובדים

קודם כל מתיישבים ולוקחים נשימה עמוקה. לאחר מכן עושים login לאחת מעמדות הלינוקס בשולחן העגול (class1219-1 עד class1219-6). אופן הכניסה – רושמים שם, ורושמים סיסמא. אם זה לא מצליח אפשר לשאול את שרה או ציפי שיושבות בצד השני של המסדרון לעזור לכם. אולי יש לכם בעיה עם הגדרת היוזר. לאחר הכניסה, עולה הסביבה החלונאית KDE, היא שונה מהסביבה החלונאית של windows במראה (ובעיקר ביכולות), אך אנחנו לא נדדקק להרבה שירותים מסביבה זו. כל מה שצריך לעשות עכשיו הוא לפתוח terminal: יש אייקון מוזר בפניה השמאלית התחתונה של המסך, שהוא לצורך העניין המקבילה של כפתור start ב-windows. לוחצים עליו, ובתפריט שנפתח בוחרים terminal -> system tools. המסך שנפתח הוא הכלי העיקרי באמצעותו נתממשק עם המערכת. דרך הטרמינל מקלידים פקודות למערכת כגון ls (מציג את רשימת הקבצים), cat (מראה את תוכנם של קבצים), cd (מעבר לספרייה אחרת) והרצה של התכניות הנפלאות שכתבנו. כן – ממש צריך לכתוב מה אנחנו רוצים

שיתבצע. אני מודע לזה שזה מעט שונה מהפעילות אליה רובכם רגילים (קליקים עם העכבר, עריכה של מסמכים בלבד), אבל תאמינו לי – זה הרבה יותר מהיר, הרבה יותר מדויק, והרבה יותר **חזק**. כאשר נפתח הטרמינל, אתם "נמצאים" בספריית הבית. הריצו את הפקודה `pwd` ותיווכחו. תקבלו שורה שנראית משהו כמו `/a/home/cc/students/space/<my user name>`. מערכת הקבצים של יוניקס היא מאוד פשוטה. התיאור הציורי הוא של עץ קבצים. ראש העץ הוא ה-"/". תחתיו נמצאות ספריות וקבצים, כגון `/usr`, `/etc`, `/home`. תחת כל ספרייה יכולות לשבת עוד ספריות או קבצים (יעני `usr/bla1/bla2/...`). ספריית הבית שלנו היא מיקום מסוים בעץ הקבצים, שבו יש לנו הרשאות לכתוב, לקרוא ולהריץ קבצים, ולייצר עוד ספריות. המיקום של קובץ או של ספרייה נקרא גם `PATH`. בעקרון, כאשר עובדים במערכת יוניקסית חשוב תמיד לדעת איפה נמצאים, במובן של באיזו ספרייה אני נמצא (מה ה-`Present Working Directory`, או `pwd`). לכן, אם תשימו לב – ב-`prompt` שלכם נמצא ה-`pwd` כדי להקל עליכם. הסימול של ספריית הבית הוא `~`.

רצוי שכל תרגיל יישב בספרייה משלו, הנמצאת בתוך ספריית הבית. יעני – תרגיל 1 יימצא ב-`~/ex1/`. זה יסייע לכם לשמור על סדר בספריית הבית.

הערה חשובה – יוניקס הוא `case sensitive`, כלומר `a` שונה מ-`A`. אם תכתבו `Ls` במקום `ls`, התגובה שתקבלו היא: `Ls: command not found`. כנ"ל לגבי שמות משתנים בתכניות שתכתבו, אז שימו לב.

רשימת פקודות שימושיות

הפקודה **החשובה** ביותר ביוניקס היא **man** (קיצור של `manual`). הפקודה נותנת הסבר על כל פקודה אחרת. לדוגמא – `man ls` יראה את הדף שמכיל את ההסברים לגבי הפקודה `ls`. `man` מציגה את המידע באמצעות הפקודה `less`, והניווט בתוך הקובץ המוצג היא כדלהלן:

תזוזה של שורה אחת כלפי מטה – `j`

תזוזה של שורה אחת כלפי מעלה – `k`

תזוזה של דף שלם כלפי מטה – `enter`

תזוזה של דף שלם כלפי מעלה – `b`

תזוזה לסוף הדף – `G`

תזוזה לתחילת הדף – `g`

ההסבר הוא לצערנו מה שנקרא – `expert friendly`. כלומר ההסבר מאוד עוזר למי שמכיר את הפקודה, יודע איך לקרוא `man page`, ורק צריך להיזכר במשהו. למרות זאת, זה הכלי השימושי הטוב ביותר שיש לכם כדי ללמוד איך להשתמש בפקודות בסביבת `Unix`, וכמו כן איך להשתמש בפונקציות של `C`. נגזר מכך שהפקודה השנייה בחשיבותה היא `man man`.

ב-`man page` ישנם כמה חלקים:

`NAME` – מראה את שם הפקודה, ותיאור קצרצר של פעולתה.

SYNOPSIS – מתאר את האופנים השונים בהם ניתן להפעיל את הפקודה, ואת כל האופציות והארגומנטים שניתן להעביר לפקודה. האופציות נקראות גם סוויטצ'ים, ובד"כ הן מינוס יחד עם אות מסוימת, למשל `ls -l`. הסוויטצ'ים משפיעים על אופן פעולת הפקודה. במקרה שעשינו `man` על פונקציה של `C`, בחלק הזה גם יופיעו ה-`includes` שהיא דורשת. DESCRIPTION – תיאור ארוך וממצה של הפקודה, התיאור של כל סוויטצ' ואיך הוא משפיע. EXAMPLES – דוגמאות לשימוש בפקודה. RETURN VALUE – במקרה שעשינו `man` על פונקציה של `C`, כאן יתוארו הערכים שהיא יכולה להחזיר. SEE ALSO – רשימה של פקודות או פונקציות הקשורות לפקודה עליה עשינו `man`.

רצוי מאוד להתרגל לפורמט של ה-`man page`. זה יחסוך לכם ולי (ולמי מכם שכבר תכנת פעם) שאלות רבות.

למרות כל זאת, מצורפת כאן רשימה קצרה של פקודות שימושיות. אין לראות ברשימה זו תחליף לשימוש ב-`man`, או בספרות.

- `ls` – מראה את רשימת הקבצים והספריות בספרייה הנוכחית בה אני נמצא כרגע (ה-`pwd`).
- ניתן להריץ גם לגבי ספרייה או קובץ אחרים, כלומר `ls mydir/mydir2`.
- `ll` – זה קיצור של `ls -l`. מראה את רשימת הספריות והקבצים אבל עם פרטים חשובים נוספים מלבד שמם, כגון ההרשאות על הקובץ/ספרייה, הבעלים שלה, זמן השינוי האחרון, והגודל בבתים.
- `cd` – כניסה לתוך הספרייה הרשומה כארגומנט לפקודה. כאן המקום לציין שיש כזה דבר שנקרא `PATH` אבסולוטי, ו-`PATH` יחסי. `PATH` אבסולוטי הוא כל מה שמתחיל ב-`/`, למשל `PATH/dir1/dir2/file1`. יחסי הוא כל מה שלא מתחיל ב-`/`. הפירוש במילה יחסי הוא "יחסית ל-`pwd`". כלומר כשאני עושה `cd dir1`, הכוונה שלי היא שאני רוצה להיכנס לספרייה בשם `dir1` שנמצאת מיד מתחת לספרייה הנוכחית בה אני נמצא. אם אין ספרייה בשם הזה מיד מתחת ל-`pwd` שלי, התגובה הצודקת שנקבל היא: `dir1: no such file or directory`. עוד דבר נחמד – הכינוי של הספרייה שנמצאת מיד מעליי הוא `".."`. כלומר, על מנת לעלות כלפי מעלה בעץ הקבצים, עושים `cd ..` דוגמא – אם ישנן שתי ספריות מתחתיי בשמות `dir1` ו-`dir2`, ואני נכנס לתוך `dir1` באמצעות `cd dir1`, אני יכול להגיע ל-`dir2` כך: `cd ../dir2`. הכינוי לספרייה הנוכחית הוא `"."`. זה אומר שאם אני עושה `cd ..`, לא עשיתי שום דבר.
- `pwd` – למרבה הפלא, פקודה זו מציגה לי את ה-`PATH` האבסולוטי בו אני נמצא כרגע.
- `cat` – קיצור של `concatenate` (שרשור). פולט למסך את התוכן של כל הקבצים שרשמתי לו כארגומנטים בשורת הפקודה, בשטף אחד גדול. לא שימושי אם אלו קבצים גדולים, כל מה שנראה זה איזה מטריקס על המסך.
- `less` – אותו דבר כמו `cat`, אבל הפלט מוצג עמוד אחרי עמוד. הניווט בפלט כפי שצוין למעלה לגבי `man`.

- which – מראה את ה-PATH האבסולוטי של פקודה מסוימת. למשל – cp which פולט bin/cp/. כך אני יכול לדעת בדיוק איזו פקודה אני מריץ.
- cp – העתקה של הקובץ ששמו רשום כארגומנט הראשון אל קובץ ששמו רשום כארגומנט השני. במידה והארגומנט השני הוא שם (או PATH) של ספרייה, הקובץ יועתק באותו שם אל תוך אותה הספרייה. דוגמאות: cp myfile1 myfile2 , cp mydir1/myfile mydir2
- mv – מעביר את הקובץ ששמו רשום כארגומנט הראשון אל שם קובץ הרשום כארגומנט שני. במידה והארגומנט השני הוא שם (או PATH) של ספרייה, הקובץ יועבר באותו שם אל תוך אותה הספרייה.
- rm – מוחק את הקובץ או רשימת הקבצים הנתונים לו כארגומנטים. לא, אין ביוניקס recycle bin. מה שמחקתם – נמחק. זה דורש יותר תשומת לב מהמשתמש, אבל מתוך ניסיון – אחרי שכמה פעמים מוחקים בטעות דברים חשובים ביותר, לומדים לשים לב.
- mkdir – יצירת ספרייה.
- rmdir – מחיקת ספרייה. פועל רק אם הספרייה ריקה. אתם מוזמנים להסתכל ב-man page של rm על מנת לראות איך למחוק ספרייה שלמה על תוכנה.
- wc – מראה כמה שורות, מילים ותווים יש בקובץ. תעשו man כדי לראות מה זה כל שדה בפלט.
- grep – פקודה עם הרבה הרבה סוויטצ'ים, ובעלת יכולות מגוונות. באופן השימוש הפשוט ביותר, היא בודקת האם המחרוזת שניתנת לה כארגומנט ראשון מופיעה בקובץ או בקבצים שניתנים לה כארגומנט שני, שלישי וכו'. אם המחרוזת מופיעה בהם, היא פולטת את השורה בה מופיעה המחרוזת. אפשר להפעיל אותה כך שתראה באיזה קובץ נמצאה המחרוזת (איך? לא זוכר. תסתכלו ב-man). הדבר יכול להיות שימושי כאשר אני לא זוכר שם של קובץ, אבל כן זוכר קטע מתוכן הקובץ.
- gcc – פקודת הקימפול (הסבר בהמשך).
- chmod – שינוי הרשאות של קובץ או ספרייה הניתנים כארגומנט.
- id – מראה את המזהה המספרי שלי (user id) ואת המזהה המספרי של הקבוצה שלי (group id).

הסבר קצר על הרשאות:

כל ישות ביוניקס היא קובץ (זה מאוד מקל על כתיבת תוכנות שמפעילות התקנים). לכל ישות יש בעלים – משתמש מסוים שהישות שייכת לו, וכן יש לה הרשאות שימוש. הרשאות השימוש הן: קריאה (r), כתיבה (w) והרצה (x) של אותו הקובץ. במקרה שזו ספרייה, הפירוש של הרשאת קריאה היא היכולת לעשות ls באותה הספרייה, כתיבה היא היכולת לייצר בספרייה קבצים, והרצה היא היכולת לעשות cd לתוך הספרייה. כידוע, לכל משתמש יש שם. כמו כן, כל משתמש שייך לקבוצה. ההרשאות על ישות הן קריאה/כתיבה/הרצה עבור המשתמש שהוא בעל הקובץ, קריאה/כתיבה/הרצה עבור חברי הקבוצה

של המשתמש שהוא בעל הקובץ, וקריאה/כתיבה/הרצה עבור כל שאר המשתמשים. עכשיו אפשר גם להבין את השדה הראשון שמקבלים בפלט של `ls -ls` – אלו הן ההרשאות. דוגמא לשדה הראשון בפלט של `ls` על קובץ עם הרשאות קריאה/כתיבה/הרצה לבעל הקובץ, קריאה/כתיבה לחברי קבוצתו, וקריאה בלבד לכל השאר: `-rwxrwx-r-` דוגמא לשדה הראשון בפלט של `ls` על ספרייה עם הרשאות קריאה/כתיבה/הרצה לבעל הקובץ, ללא הרשאות לחברי קבוצתו, וקריאה/כתיבה/הרצה לכל השאר: `drwx---rwx` שימו לב שעבור ספרייה האות הראשונה היא `d`, שמציין לי שזה איננו קובץ רגיל, אלא ספרייה. הפקודה `chmod` משנה הרשאות על קבצים. מן הסתם, אני מסוגל לשנות הרשאות רק על קבצים/ספריות שבבעלותי מלכתחילה.

הוספת הרשאות ריצה על קובץ לי ולחברי קבוצתי: `chmod ug+x myfile`
הוספת הרשאות קריאה וכתיבה על קובץ לכולם: `chmod ugo+rw`
הסרת הרשאות כתיבה והרצה מקובץ לשאר העולם: `chmod o-wx`

ההרשאה על ספריית הבית שלכם היא קריאה/כתיבה/הרצה עבורכם בלבד, וזהו. אין הרשאות לחברי הקבוצה שלכם, על מנת שחבריכם לקורס לא יוכלו לגשת לקבצים שלכם. זו אחת הדרישות לקבלת ציון על התרגילים. אל תנסו לשנות את ההרשאות על ספריית הבית שלכם במהלך הקורס.

מה, שאני אקמפל?

אני מניח שאתם כן יודעים את הבסיס של כתיבת תוכנית `c`. כלומר שאתם מבינים מה כתוב פה:

```
int power2( float );
int main() {
    int a,i;
    float f=2;
    char c='Q';
    char *s="to the power of";

    for ( i=1; i<13; ++i ) {
        f=power2(f);
        printf("2 %s %d is %f\n", s, i, f);
    }
}
```

```
int power2( float z ) {
    return powf(z,2);
}
```

אבל - כתיבת התוכנית היא רק חצי הסיפור. על מנת שהמחשב יוכל להריץ את התוכנית, עליה להיות כתובה בשפה המובנת לו. אנחנו צריכים משהו שיודע לקרוא שפת תכנות (שנקראת ה- `source`

(code), ולתרגם אותה למה שנקרא object code, שהוא קובץ בינארי (יעני רצף ביטים), המכיל פקודות מכונה שאותן המעבד מסוגל להריץ. היצור המוזר הזה נקרא קומפילר. אופן השימוש הכי נאיבי בקומפילר (לאחר שכתבתי ושמרתי קובץ בשם myprog.c):
gcc myprog.c

במידה והקימפול הצליח, יוצר קובץ בשם a.out, שהוא קובץ ההרצה. הרבה יותר טוב לעשות:

```
gcc myprog.c -o myprog
```

זה יגרום לכך שקובץ ההרצה לא ייקרא a.out, אלא myprog. זה הרבה יותר אינטואיטיבי שה-executable (קובץ ההרצה) ייקרא בדיוק באותו שם כמו ה-source code, רק ללא ה-"c". אם אנחנו משתמשים בפונקציות מתמטיות, נצטרך גם לצרף ללינקוג' את הספרייה הדינאמית שמכילה את הפונקציות המתמטיות. עושים את זה כך:

```
gcc myprog.c -o myprog -lm
```

עכשיו אנחנו רוצים לכתוב myprog, ואנחנו מצפים שהעסק פשוט ירוץ. לא כך הוא תמיד. ראשית כל, על מנת שמ"ה תדע היכן יושב הקובץ שאותו רוצים להריץ, ישנו משתנה סביבה בשם PATH. המשתנה הזה מכיל את ה-PATH של ספריות שבהן יש קבצי הרצה. לפעמים, משתנה הסביבה הזה לא מכיל בתוכו את ". ", שהיא הכינוי למיקום הנוכחי שלי. כדי להימנע מעצבים מיותרים, הריצו myprog/. זה אומר שאני רוצה להריץ משהו שנמצא כעת בספרייה "הזו" ונקרא myprog.

אוף, למה זה לא מתקמפל?

שגיאות קומפילציה הן השגיאות המעצבנות והנפוצות ביותר בתכנות, אך גם הקלות ביותר לתיקון. העצבים נובעים מכך שלרוב אנחנו מרגישים שאנחנו היינו בסדר, עבדנו קשה, והמחשב הבן-זונה לא רוצה לקמפל לנו את התכנית.

לא כך הוא.

שגיאת הקומפילציה נובעת כמעט תמיד משגיאות טיפשיות שלנו. ניתן להתגבר על רובן ע"י כתיבה מסודרת של הקוד, שימוש בשמות משתנים ברורים (לא קצרים מדי, ולא ארוכים מדי), והזכר נכונה של הטקסט.

הטריק הכי טוב שצריך לזכור הוא לקמפל הרבה במהלך כתיבת התכנית. כלל אצבע – כל 7 או 8 שורות קוד, לשמור את הקובץ ולקמפל. כך כמות שגיאות הקומפילציה תהיה קטנה יותר. אין משהו מפחיד יותר מלראות פלט של 100 שורות של שגיאות קומפילציה לאחר הרצת gcc, אחרי שעובדים 4 שעות רצוף וכבר אין כוח לעשות תיקונים.

לגבי תיקון השגיאות – השיטה היא לעבוד מההתחלה לסוף. כל שגיאת קומפילציה מופיעה עם מספר השורה שבה היא נתגלתה. כדאי להתחיל לתקן מהשורה הראשונה לגביה מופיעה שגיאה. תקרא טוב טוב את לשון הודעת השגיאה, תראו שאתם מבינים בערך מה הבעיה. לעיתים רבות, תיקון של שגיאה בשורה אחת מבטלת את כל שאר שגיאות הקומפילציה. אחרי כל תיקון של שגיאה בודדת – לשמור ולקמפל שוב. לא להתעצבן!

אופ, למה זה לא רץ?

לא כל תכנית רצה במכה הראשונה. ישנן שגיאות של תכנות שאותן הקומפיילר איננו מסוגל לזהות. קורה לא מעט שהתכנית שלנו כתובה כך שהיא מנסה בזמן הריצה לבצע דבר לא חוקי (כגון לכתוב לשטח בזיכרון המחשב שאיננו מוקצה לה), ובמצב כזה מ"ה דואגת להעיף אותה מריצה. המצב הזה נקרא תעופה. עבור יוניקס, התעופה היא כמעט תמיד בגלל ניסיון לגשת לשטח זיכרון שאיננו מוקצה לתכנית, או כתוצאה מחלוקה באפס. מה שהמשתמש רואה היא הודעת Segmentation Fault, או Zero Divide, שנפלטת אל המסך. התעופה יכולה להתרחש בכל נקודה במהלך הריצה של התכנית. על מנת לזהות את הקטע הבעייתי בתכנית אשר גורם לתעופה (שגם היא, אגב, כולה באשמתנו ולא באשמת המחשב, אז תפסיקו להרביץ למסך), הדרך הנאיבית והפשוטה ביותר היא לרשום הדפסות בתוך ה-source code שלנו שיצינו לנו היכן התכנית "נמצאת" במהלך הריצה. הדפסות אלו נקראות הדפסות debug, כי הן עוזרות לנו להתגבר על הבאגים בתכנית. למשל – כותבים:

```
fprintf (stderr, "in the beginning\n");
```

בתחילת התכנית. כותבים:

```
fprintf (stderr, "in the middle\n");
```

באמצע התכנית. וכותבים:

```
fprintf (stderr, "almost finished\n");
```

לקראת סוף התכנית.

לפי ההדפסה האחרונה שרואים במהלך הריצה של התכנית, ניתן לזהות באיזה קטע היא עפה, ואז אפשר לכתוב עוד הדפסות debug בסביבה של אותו קטע הקוד. אפשר גם להדפיס ערכים של משתנים, על מנת לעקוב אחרי השתנותם במהלך התכנית. ברגע שהבנו איזה קטע קוד גורם לתעופה, ניתן להתחיל להבין למה היא מתרחשת. במירב המקרים, אתם תזהו שהתעופה היא כתוצאה מניסיון כתיבה למערך מעבר לגבולותיו. תעופה מאוד פופולארית בקרב תכניתנים מתחילים היא להריץ scanf בצורה לא נכונה, כך שהקלט מוכנס לתוך משתנים, ולא לתוך הכתובת של המשתנים כפי שצריך. לדוגמא, סביר להניח שזה יעוף:

```
;(scanf("%d %f", myint, myfloat
```

```
;(scanf("%d %f", &myint, &myfloat
```

וזוה כיוון ש-scanf מנסה להכניס את הקלט לכתובת מסוימת בזיכרון.

צרה קלאסית נוספת שמתרחשת לעיתים קרובות היא לולאה אינסופית. במצב הזה התכנית לא עפה, אך היא לא מחזירה שום פלט, ותמשיך לרוץ עד אינסוף אם לא נעצור אותה בכוח. הדרך לעצירת התכנית היא ע"י הקשת ctrl-c בטרמינל ממנו הרצתי אותה (אלא אם ההרצה הייתה ברקע, כלומר כתבנו ./myprog).

לאחר שגרמנו לכך שהתכנית לא תעוף ולא תרוץ אינסוף זמן (בעגה המקצועית – דיבגנו אותה), עדיין צריך לוודא שהתכנית עושה מה שהיא צריכה לעשות. בקורס הזה, הפירוש הוא שהיא מחשבת את המספרים הנכונים. ברוב התרגילים יהיה פתרון אנליטי שאפשר יהיה להשוות אליו.

עריכת קבצים

קודם כל, תזכרו שכאשר כותבים תכנית c, שם הקובץ צריך להיגמר ב- ".c".

אפשר להשתמש בעורך הנחמד והפשוט בסביבת KDE בשם gedit.
עורך נוסף, לגביו יש הסברים באתר (בקישור a short memory refresh), הוא xemacs.
ניתן להשתמש בשניהם בעמדות הלינוקס בשולחן העגול.
ההפעלה פשוט ע"י הקשת <file <xemacs או <gedit file.
הבעיה עם שני העורכים האלו היא שהם גראפיים, והם פועלים מעל פרוטוקול גראפי שנקרא X. זה
בעייתי כיוון שאין לכם בבית את התוכנה המסוגלת להציג חלונות שעובדים מעל הפרוטוקול הזה, לכן
בבית לא תוכלו להשתמש בהם.
כאשר עובדים מהבית, אפשר להפעיל את emacs במצב טקסטואלי. לצורך כך, הריצו קודם כל את
הפקודה:

```
unsetenv DISPLAY
```

ואח"כ את הפקודה: emacs myprog.c
ישנו גם עורך טקסטואלי וותיק וטוב שנקרא vi, אך השימוש בו קשה למתחילים.
העצה הכי טובה שאני יכול לתת היא לעבוד השולחן העגול. כך גם תוכלו להיעזר אחד בשני. אין כמו
לילה של תכנות לייצר גיבוש חברתי.

ענייני קלט/פלט

לכל תכנית שמריצים מיוחסים שלושה "צינורות" דרכם מידע יכול לזרום אליהן ומהן החוצה.
הצינור הראשון הוא צינור הקלט, ונקרא גם standard input, או stdin.
הצינור השני הוא צינור הפלט, ונקרא גם standard output, או stdout.
הצינור השלישי הוא צינור פלט שמיועד לשגיאות תוכנה, ונקרא גם standard error, או stderr.

כל תכנית יכולה לבחור לקרוא דברים מה-stdin, ולפלוט דברים לתוך stdout. העניין השימושי כאן
הוא: אני, בתור זה שמריץ את התכנית, יכול להחליט לאן יחובר הקצה שני של צינור. כלומר – אני
יכול להפנות את צינור הקלט או הפלט לקובץ, או לתוך צינור הקלט או הפלט של תכנית אחרת.
דוגמא: cat myfile > myfile2. פעולה זו גורמת לכך שצינור הפלט של התכנית cat יופנה לקובץ
בשם myfile2. התוצאה של הפעולה תהיה יצירת קובץ בשם myfile2 שתוכנו יהיה זהה ל-myfile.
הפעולה הזו תבצע את אותו הדבר בדיוק: cat < myfile > myfile2. צינור הקלט של התכנית cat
מופנה אל הקובץ myfile, אותו היא קוראת, ומעבירה לתוך צינור הפלט, המופנה אל הקובץ myfile2.
נאמר וכתבתי תכנית, שאמורה לקבל רשימה של פרמטרים מקובץ. אינני חייב לפתוח את הקובץ
בתוך התכנית. התכנית ממילא מחזיקה קצה אחד של צינור קלט פתוח. אני יכול לבקש ממנה לקרוא
ממנו באמצעות scanf. את הצד השני שלו אני יכול לחבר אל קובץ פרמטרים שכתבתי מבעוד מועד,
ע"י כך שאריץ: myprog < params/.
אם ארצה שפלט התכנית יישפך לקובץ, אריץ: myprog < params > myoutput/.

בתור ברירת מחדל, ה-stdin וה-stdout מחוברים שניהם למסך. נסו להריץ cat בלי כלום. כל מה
שתכתבו למסך ייפלט למסך, לאחר הקשת enter.

Pipe: ביוניקס ישנו יצור נחמד, שכל תפקידו בחיים הוא לקרוא מצינור הקלט שלו ולפלוט את אותו הקלט לתוך צינור הפלט שלו. לכן התכנית הזו נקראת (pipe (duh). הסימון שלה הוא | . משתמשים בו בשורת הפקודה כך: ./myoutput/ > ./myprog2 | ./myprog < params . הפירוש של שורה זו הוא הרצה של תכנית myprog הקוראת מתוך קובץ params, הפלט שלה מופנה לתוך הקלט של התכנית myprog2, שעושה איזושהי פעולה על הפלט של myprog, ופולטת לתוך הקובץ myoutput.

דוגמא שימושית: cat myfile | grep Bush . פקודה זו תפלוט למסך (שאליו מחובר צינור stdout בתור ברירת מחדל) את כל השורות בקובץ myfile שבהן רשום שמו של הליצן הידוע. כמובן שהייתי יכול לציין את שם הקובץ בשורת הפקודה ע"י grep Bush myfile, ולא להשתמש ב-pipe. מאידך, כוחו הגדול של ה-pipe בא כאשר רוצים לחבר הרבה תכניות יחד, כדי לבצע פעולה אחת. לדוגמא, הרצף הבא עובר על הקובץ etc/passwd/ ובודק אם לא הוגדרו בטעות שני משתמשים בעלי אותו שם:

```
“ cat /etc/passwd | cut -f 1 -d ':' | sort | uniq -c | grep -v “ 1
```

זה מראה את הכוח הגדול שאני יכול לקבל מכך שכל תכנית עושה דבר אחד בלבד (והיטב), כאשר מ"ה מאפשרת לי לקשר ביניהן, וגם מדגים את עקרון KISS שכדאי לכם ליישם בתכנותיכם: .Keep It Simple, Stupid.

נכתב ע"י אוהד ברק