המחשב

המחשב בנוי על שני רכיבים עיקריים:

1) hardware - חומרה, כל המחשב, הכול עצמם.
2) software - תוכנה, התוכניות המפעילות את המחשב.

חומרה

החומרה בנויה מכמה מרכיבים עיקריים:

1) CPU* - יחידת העיבוד. בה המחשב מבצע את הפעולות
2) RAM* - הזיכרון שעובד עם המעבד. עובד על מתח.
3) הדיסק - זיכרון משני השומר בתוכו את הקבצים קבוע, גם אם מכבים את המחשב (בניגוד ל-RAM שנמחק בכיבוי).

central Processing Unit

* Random access memory

תוכנה

התוכנה מאפשרת למחשב לבצע את הפעולות הרצויות. התוכנה המרכזית המנהלת את המחשב היא מערכת ההפעלה. היא גם שולחת פקודות ומעבירה את הפקודות מהדיסק ל-RAM ול-CPU וגם מקבלת פקודות מהחומרות.

שפת תכנות

המחשב עובד רק עפ"י שפה בינארית. לכן, כדי שנוכל לתת פקודות למחשב, יש צורך בשפה עילית כלשהי שתנקוט את הפקודה שנכתוב ותמיר אותה לשפת מכונה. לשפת תכנות יש syntax שזה התחביר וה-semantic שזו ההבנה של המחשב. לכל שפה יש קומפיילר שהוא המתרגם.

התהליך של הארת הדורות משפת התוכנה לשפה המכונית נקרא קומפילציה. (מה מתשב/מערכת הפעלה מקומפל בדורה אותן)

סוגי בעיות בשפה

1) בעיות Syntax - כאשר הקומפיילר נתקל בקוד שאינו שייך לשפת התכנות. במקרה כזה, הוא נעצר ומודיע היכן הוא נתקע. לכן קל יותר לפתור בעיות כאלו.

2) בעיות לוגיות - כאשר כותבים קוד בשפת התכנות, אלא שלו לא הפקודה שרצינו. הקומפיילר כן יבצע את הפקודה אך לא הפקודה שרצינו. כדי לפתור בעיות כאלו - צריך להבין בעצמנו איפה טעינו, ולכתוב פקודה מתאימה.

סביבת עבודה

התוכנה Visual Studio היא בעצם סביבת עבודה. כלומר, כדי לבצע את התכנות יש צורך בקומפיילר, בפונט תימות בו כותבים, במתקן שגיאות, במסך שומר את התוכנית וכדו'. בתוכנה הזו רוכזו את כולם במקום אחד.

קוד ופלט בתכנות

קוד - מה שהמתשמש קולט לתוכו מהר שאותנו כותבים.

פלט - מה שהמתשמש מודאג החוצה אל המסך.

התוכנה היא כזו שהופכת את הקוד לפלט. היא קולטת את מה שאותנו הכנסנו ומעבדת את הפקודה לפי המתשמש בעזרת פלט. הפרק לעשות כאות היא ע"י אלגוריתם.

אלגוריתם

אלגוריתם הוא פרק הפתרון, הוא רצף של פקודות שעל יפה אנו מצפים אל הפתרון. יש מה פרקים לפתור בעיות מקום אורכים ותקים קבלים. המטרה שלנו הוא לפתור את הבעיה ע"י האלגוריתם הנכון ולבצע אותו. (איך בשל מ מגיון של יעילות, לו משה אום הפרק תהיה אורכה הפקר שתהיה נכונה כולל, האלגוריתם חייב להות מוכן ע"י הקומפיילר עד שהוא יכל לבצע אותו, ולכן צריך לפרק את האלגוריתם למה שיותר שבים עפ"י רמה שהקומפיילר יבין. חשוב להבין שהקומפיילר מבין רק מה שקבענו בתכנות, אם לו כתבנו - כאילו לא קיים.

המחשב מבצע פקודות לפי איך שאנחנו מורים לו. הוא לא מחשב מעצמו. לכן, אנחנו צריכים לחשוב מ הפרק הכי קצר שתגיע אל הפתרון ע"י שנעבור קיר מסוים שחוצר מ עצמו ולבנכם אותו לפקודות מעשות.

סוגי פקודות למחשב

1) פקודות בסיסיות פשוטות - לבקש מהמחשב לבצע פקודה, כמו קלט, פלט, חיבור, חיסור וכפ' משנק הפקודות שהוא יודע לבצע, לפי התוכנה.

2) סלקציה ושאלות בלוגיות - לבקש מהמחשב לבחור בין כן או לא, ~~לפי התנאים~~ לפי המידע שברשותו.

3) לולאות - לבקש ממנו לחזור כמה פעמים מ פקודות.

* בתכנות, בשונה מאתמטיקה, המשתנים אינם בעלי ערך קבוע, כלומר אם אני מאפיין X=8 אז במתמטיקה ה-X תמיד יהיה 8. לעומת זאת, בתכנות המחשב עקב לפי שבים בהמלבה. אם אני אפיין ל X=8, אז כל עוד לא הפרתי אותה היא בגודל 8, אבל אם אני חלה לשנות את ההגדרה ואכתוב לו אח"כ ל X=9, מבחינת המחשב X=9 (בניגוד למתמטיקה, זה לא יהיה נכון).

<u>חשוב להבין</u> שחוזרים לאפיין את המשתנה לפני שמגדירים לו ערך. אם לא נגדיר את המשתנה, המחשב לא יפעל לכן לחלב את המספר שבקשנו.

סמן התכנית

אם נקח את הפקודה: `cout<< "Hello_World"<<endl;`

הפלט יהיה:

```
Hello World
Press any key to continue...
```

אם אני אכתוב את זה בשלוש פקודות נפרדות

```
cout << "Hello";   ⇒   Hello
cout << "_World";   ⇒   Hello_World
cout << endl;   ⇒   הפלט שנעלם.
```

הפלט יהיה זהה. הסיבה היא שסמן של התכנית לא 55 בלי פקודה ולכן אחרי שהוא יפיס את הפקודה הראשונה הוא יעבור במקום ויקרא את הפקודה הבאה.

בעיות חלוקה (Errors)

ישנם שני סוגי בעיות שהתוכנית תשבר בגורמה שונה:

1) בעיות בהגדרת משתנה - אם לא הגדרנו בתוכנה את הערך של המשתנה, הקומפיילר יתקע ולא ירץ את התוכנית. כי הוא לא מכיר את המשתנה.

2) בעיות בהצבת ערכים - כשהגדרנו משתנה אבל לא הכנסנו ערך. התוכנית תרוץ כי הקומפיילר מכיר את המשתנים, אולם הוא תיתקע בהפסקה כי הוא לא יודע מה הערך.

* אם נתעלם מהבעיה (מס' 2) התוכנית תמשיך לרוץ אולם תבצע אינטרוול מספר מוכר. הסיבה הוא כי אין פתר כדה ביק בתוכנית. אם לא הגדרנו ערך, הקומפיילר יחפש ערכים קודמים שהוצבו באותו משתנה ואותם הוא יפסיס.

חלוקה וקבועה

במתמטיקה - אם נכתוב $x=x-1$ הפתר יהיה לא נכון ולא אפשרי לעומת זאת, בתכנות הפתר אפשרי ואף מתבצע. כאן שמאל = בתכנות פרושו הצבה, אנחנו לוקחים מספר ומצבים אותו במשתנה X. וכמובן שחלבה אפשר לשנות.

אם נרצה לקבוע שלמשתנה יהיה ערך קבוע - נשתמש ב ==.

שני סוגי חילוק

1) מנה - (div) כפי לפעמים כמה פעמים נכנס משתנה במשתנה - /

2) שארית - (modulo) לפעמים את השארית מהחילוק - % (המון אחוז)

כללים בכתיבת משתנה

(1) לא מתחילים בסימן תחתון (underscore) - _Abc X

(2) לא מתחילים במספר. 123Ab X

(3) אם המשתנה הוא יותר מאותיות אחת בעלת משמעות, יש להפריד בין המילים בשתי דרכים:

א. הדרך היותר מקובלת היא לכתוב כל תחילת מילה באות גדולה, למשל: numberOne, ProductPrice. (camel notation)

ב. בעזרת קו-תחתון, כמו: lovely_day, Number_2

סוגי משתנים

int - מאפיין משתנים שלמים. a= 3; a=17

float - מאפיין מספרים עשרוניים. a=1.73 , 0.75

char - מאפיין תווים a= 'b', 'co'

כתיבת קבצים

בשביל לכתוב לקובץ, צריך לבצע אחד מפעולות:

(1) צריך להפיק לו שהוא כרם

(2) לתת לו שם.

(3) להפיק לו כיוון מרמה מאיפה לאיפה.

(4) להכניס אל לחוצאו נתונים (תווי בצמה).

למשל, כשאני רוצה להשתמש בפקודות פלט/ קלט, אני צריך להשתמש בספריה בה נמצאות פקודות הפלט והקלט, ולכן אני מפיק לתוכנה להשתמש בספריה (שלה כבר יש שם וכיוון).

קובץ לכתיבה

```
#include <iostream>
```
בחירת ספרית קלט/פלט

```
#include <fstream>
```
בחירת ספרית קבצים

```
using namespace std;
int main()
{
  ofstream(1) fout(2)("yyy.doc")(3)
```

(1) אני מאפיין כמה of שמונה: Out file, פומר הוא יוצא מן התנכות ומאז לספריית הקבצים fstream.

(2) אני מאפיין את שם הקבצ (אפשר לבחור מ שם).

(3) אני מאפיין את הקובץ אליו אני רוצה לחאז.

```
  fout(1) << "HelloWorld"(2) << endl;
```

(4) אני רוצה להשתמש בקבצ שבתוכו, שהוא יקח את מה שבקש ממנו ושלח אותו אל ~~הספרה~~ הקובץ שהאיתו.

(5) אני מגדיר את המילים שהכנסתי בקבצ הנ"ל, פומר: שלחתי את המילים האלו לקובץ שנמצא בספריית הקבצים.

```
  cout << "Hello_World" << endl;
```

במקרה הזה אני בוחר את הקבצ cout שמאפיין (ע"י הספרה <iostream>) כקבצ מהתנכות למסך, ושולח אליו את המילים שלום עולם. התוצאה תהיה הפסקה על המסך.

```
  return 0;
}
```

תוכנות הפוטאו שכתבנו (fout) היו שיוכר קובץ בספריית הקבצים של התוכנה visual studio) בשם: doc.yyy שכאשר נפתח אותו נראה את המילים שלום עולם. הקובץ יפתח בקובץ word בכל הסיומת.

סיומות של קבצים

סיומת של קובץ היא למעשה הוספה לארכת ההפעלה לפתוח את הקובץ באמצעות התוכנה המתאימה. למשל: הקובץ שלנו doc.yyy הוא קובץ שהסיומת שלו מורה לארכת ההפעלה לפתוח אותו באמצעות winword.exe (קובץ הפעלה של הורד. אנחנו יכולים לפתוח את הקובץ גם בכתבן, אם נשנה את הסיומת ל txt- (כל סוג הקובץ שתנה לסוג של פקס הישמע) או אם נעשה- פתיחה באמצעות, ופתח כתבן.

- מבחינתנו, לסיומת אין שום משמעות.

* הערה: לא לחתפבל. סומת doc. לא אומרת שכה קובץ ורד, אלא שהקובץ יפתח באמצעות ורד, באותה מידה- סיומת ppt. אומרת שהקובץ יפתח באמצעות Visual Studio.

סוגי הקופות (המשך)

אינטרבל לולאה

1) **כל עוד** - לולאה הבנויה על בסיס שהתנאי יבוצע כל עוד כל עוד שהתנאי מתקיים. (מבצע 0 עד n)
2) **בצע עד** - התנאי יבוצע כל עוד מסוימות עד שהתנאי שהגדרנו יתקיים. (מבצע פעם אחת גם אם לפי התנאי)

- שני הסוגים הם למעשה אותו דבר רק בהסתכלות שונה.

סלקציה

בחירה. מבצעים בפני התנאים שאלה מנתה והתאים יבוצע בהתאם לשאלה. (כמו הסקלה לא תבוצע כמה פעמים).

אופרטורים בוליאנים בסלקציה

$<, >, <=, >=, ==, !=$

מאופרטורים המחברים: (לא) ! , (או) || , (וגם) &&

שני סוגי לולאות

1) **לולאת מונה** - לולאה המופרדת כמה פעמים.
2) **לולאת תנאי** - מבצעת פעולות. עד ש.

חלקי לולאה

1) **תנאי ההתחלה** - להגדיר איך הלולאה מתחילה.
2) **תנאי הלולאה** - השאלה האם הלולאה מתקיימת.
3) **קידום הלולאה** - שבתוך הלולאה תהיה אפשרות להקפיץ (למשל במונה, לספר הלאה) ולהגיע אל תנאי העצירה.

לולאות כל עוד

```
int a=7
while (a<15)
{
cout << a <<endl;
a++;
}
```

קביעת תנאי התחלה
קביעת תנאי עצירה
בגוף הפעולה
קידום הלולאה

המקומפיילר יבדוק האם התנאי קיים, אם כן-יבצע.

לולאות בצע עד

```
int a=10;
do
{
cout<<a<<endl;
a++;
}
While (a<11);
```

קביעת תנאי בצע
ביצוע הלולאה
קידום הלולאה
בדיקת תנאי עצירה.

לולאות מונה

```
for (קידום ; תנאי ; אתחול)
{ ... }
```

קביעת הלולאה.

הלולאה

בלולאה הזו, יש שלושה חלקים: תנאי, אתחול, קידום, בהם קובעים את קיום הלולאה. אפשר להשמיט את אחד החלקים המשמשים אם האתחול מתבצע לפני הלולאה) אבל חייבים ;.

הערה בתוך השורה/בתוך הקטע

כאשר אנחנו רוצים לשים הערה בחלק מהתוכנית, ישנן כמה אפשרויות לסמן:

1) // - הערה לשורה שלמה (מסמן בתוך את כל השורה)

2) /* - הערה לכל מה שמתחת עד סוף התוכנית.

3) /* */ - הערה מקטע עם קטע. (/*-פתח, */-סגר)

האינקר והדקרמנט

הקומפיילר תפקידו לקמפל את התוכנית. המחשב יש שם פרוטוקול מבני מסמה תוכניות. ברגע שבשפת התוכנית יש חוסר, הבנה של התוכנית מתבצע ע"י הקומפיילר, אבל האינקר שפקידו לחבר בין ~~הפקודות~~ התוכניות לא יחלט כי יש לו שתי תוכניות עם נקודת התחלה ולכן צריך לעשות rebuild solution.

ההבדל בין ++i לבין i++

בפקודה העומדת בפני עצמה אין הבדל. ההבדל הוא בפקודה הבאה ב-i. ++i אומר תסתכל מה יש ב-i ואז תוסף אחד לעומת זאת, i++ אומר קודם לחסל אחד ואז להסתכל ב-i.

דוגמא

I int i=5; int j=i++; במקרה הזה מבקשים לבדוק מהו i ולהכניס ב-j ואחר להוסיף אחד, ולכן הפלט יהיה j=5

II int i=5; int j=++i; כאן מבקשים קודם להוסיף 1 ואז לראות את ה-i ולהכניס ב-j, ולכן הפלט יהיה j=6.

מספרים רנדומליים

פותחו טבלת מספרים בגדלים שונים (מ 1-1000, או מ-10000 עד 1000000 וכדו') שבה כל המספרים הם בסדר רנדומלי, שאין קשר כלשהו בין אחד לשני ושמתקיים מתמטיקה לבחור מספר, הוא פשוט לוקח מהטבלה.

- הפקודה נקראת: rand()

כאשר נכתוב rand - המחשב יבחר מספר כלשהו בין 0 ל- RANDMAX (בדרך כלל 32,767).

- אם נרצה לבחור תחום מסוים ($0 \le rand \le x$), ניתן להשתמש במודולו %. אם נרצה x מספרים - נבחר x%rand. ~~והמספר~~ והמספרים יהיו בין $0 \le x-1$.

- אם רוצים טווח מסוים בין: $a \to b$, נשתמש בנוסחה:

$$\boxed{rand() \% (b-a+1) + a}$$

מאיפה להתחיל, כמות המספרים הרצויה

- בדרך לכלל שכל פעם שמריצים, המחשב יתן את אותו רצף מספרים (אפי' הטבלה, מהתחלה). זה נעשה כדי שתהיה אפשרות לתקן שגיאות (אם זה לא היה חוצר - לא ניתן היה לדעת האם הטעות הוא בגלל המספר הספציפי או טעות תכנותית)

- אם רוצים מספרים רנדומליים באמת?

ישנם פונקציות הנקראות: srand()

וכדי שתהיה רנדומלית: srand(time(NULL)) → מאוסס טבלה לפי פונקציית זמן.

הקלדה Re-use

כשיש קוד בתוכנית ואנחנו משתמשים בו כמה פעמים בתוכנית, נוצרת בעיה. אם יש בעיה בקוד, יהיה קשה לצאת אותו ולתקן אותו. לכן, כדי שהפסר יהיה לשלוט בקלדה ולצמצם את הבעיה, מחלקים את הקוד לחלקים ובונים כל חלק בנפרד. החלקים הללו הם קומפוננטות.

כדי שהתוכנית תדע איך להשתמש בהם - יש פונקציות הפונקציות מוגדרות כמולה ואחריה סוגריים. דוגמאות לפונקציה: srand( ), time( ), for( )

איך קוראים פונקציות:

הפרמטרים שפונקציה חייבת להכיל:

1) שם ייחודי - לכל פונקציה צריך שם משלה.
2) קוד - אל הפונקציה.
3) פרמטר - שיהיו לה משתנים להשתמש בהם.
4) מה היא מחזירה - אולה סוג

קודם כותבים מה היא מחזירה, אחר כך את שם הפונקציה ואז את הפרמטרים (ולבסוף את הקוד). דוגמא:

```
double sqrt(double x)
```

מה היא מחזירה — שם פונקציה — פרמטר

את הפונקציה מגדירים לפני תחילת התוכנה. פונקציה חייבת לחזור מוגדרת לפני השימוש בו.

prototype של פונקציה

כפי שהקומפיילר ישתמש בפונקציה, הוא צריך לדעת שהיא קיימת. במקרים והפונקציה קיימת בקובץ אחר הקומפיילר לא יראה אותה, ותהיה טעות. כפי לתקן זאת צריך פשוט לכתוב למעלה את הפרוטוטייפ (שזו שורת ההצהרה של הפונקציה, כמו שכתבנו לעיל). ואז הקומפיילר יודע שקיימת פונקציה כזו ושולח את הלינקר לחפש בקבצים אחרים.

function overload

הפונקציה מוגדרת ע"י 3 פרמטרים: שם, מספר פרמטרים, סוג פרמטרים - לפי הסדר.

ולכן ניתן להגדיר כמה פונקציות עם אותו שם, רק עם תוכן שונה (משנה את פונקציות עם אותו שם רק עם מספר פרמטרים שונים, או סוג פרמטרים שונים). אם יהיו שתי פונקציות עם אותה חתימה - תהיה בעיה בתוכנית.

למשל, האופרטור +, הוא מחבר בין שני מספרים. אם נכניס שני מספרים שלמים הוא יחבר בין שני שלמים, אולם אם נגדיר אותם כ-float, אז הוא יחבר בין שני ממשיים.

כל פונקציה התלויה במשתנים ומסוגם. וכן לכל אותה פונקציה (int 3.5+4.3=7, לעומת כאן float 3.5+4.3=7.8)

חוקי חתימה בהתנגשות

~~הקומפיילר ~~ כשיש שתי פונקציות המוגדרות לפרמטרים

מסווים (כמו האופרטור + ל int ול-float). במקרה
והכנסנו שני פרמטרים שונים (למשל אחד int ואחד float)
הקומפיילר לא יוכל באיזו מהפונקציה לבחור. חוק ההמרה
קובע שההמרה תעשה אל הפונקציה שתשמור על רמת
הדיוק הכי גבוהה ולא תאבד נתונים. לכן, במקרה של ה+
הפונקציה הבחרת תהיה float והמספר השלם יומר ל-float.

דוגמא:

```
~~cout<<~~ int a;        הצהרת שלם
float b;                 הצהרת משתנה
cout<< a+b;              הצהרת פונקציות חיבור
```

כיוון שה-b הוא float, התשובה תתקבל כ- float,
כדי לא לאבד את המספרים שאחרי הנקודה העשרונית.