ההבדלים בין שפות תכנות C ו- C++

ספרות

- The C++ Programming Language / Bjarne Stroustrup (Third edition).
- Thinking in C++ / Bruce Erkel (Second edition) - available online.
- Effective C++ / Scott Meyers, Addison-Wesley, 1997 (Second edition).

כל ספר טוב על שפת C++ יספיק לצרכי קורס זה.

Difference between C and C++

- שפת ++C, היא שפה בעלת תפיסת תכנות שונה לחלוטין מ-C, אך משיקולים עסקיים עדיין תומכת בשפת C.
  כל דבר שניתן לבצע בשפת C –ניתן לביצוע בשפת ++C, אך לא נהוג (ולעניין קורס זה – **אסור**) להשתמש בספריות של שפת C עבור דברים שיש להם תחליף ב-++C.
- סיומות לקבצים - .h , .cpp.
- ישנם מס' הבדלים סינטקטיים בין השפות וכן מספר גדול של חידושים.

Difference between C and C++

הצהרה על משתנים

C++

- ניתן להצהיר על משתנים בכל מקום בבלוק.

```
for (int i=0;i<10;i++){
    int a=5;
    // Can access i and a
    a=3;
    int b=3;
    // Can access i, a and b
}
// i,a and b doesn't exist
```

- ע"מ לחסוך בזיכרון, נעדיף להצהיר על משתנה כמה שיותר קרוב לשימוש בו.

C

- ניתן להצהיר על משתנים רק בתחילת בלוק

```
{
    int i;
    // Can access i
    for (i=0;i<10;i++){
        // Can access i
    }
    // Can access i
}

// i doesn't exist
```

- המשתנה i מוכר החל מרגע ההצהרה עליו ועד לסיום הבלוק החיצוני.

Difference between C and C++

הצהרה על משתנים - דוגמא

- איפה בקוד נגדיר משתני לולאת for?
- ואיפה דגלי לולאת while?

- כיתבו קטע קוד הרץ מ-1 עד 100 וסוכם את החזקה החמישית של כל אינדקס ומחזיר כמה זה:

$1^5 + 2^5 + 3^5 + \ldots 100^5 = ?$

לחישוב החזקה מותר להשתמש רק בפעולות כפל!

חישבו היכן נגדיר כל משתנה!

Difference between C and C++

Function overloading - העמסת פונקציות

- תיאור הבעיה:
  - אנו רוצים פונקציה שתבצע לנו חיבור בין 2 משתנים.

```
int add(int a, int b){
    return a+b;
}
```

  - מה נעשה כשנרצה פונקציה נוספת שתחבר בין 3 משתנים?

```
int add3(int a, int b, int c){
    return a+b+c;
}
```

  - וכשנרצה פונקציה נוספת שתחבר בין int ו- float?

```
int addIF(int a, float b){
    return a+b;
}
```

Difference between C and C++

Function overloading - העמסת פונקציות

- הבעיה: אותה פעולה בסיסית (חיבור משתנים) דורשת מספר פונקציות עם **שמות שונים לגמרי!** (דוג' נוספת swap).
  - השקעת מאמץ בהמצאת שמות חדשים לפונקציות השונות, וזכירתם.
  - קריאות הקוד נפגעת.
  - יש חשש להתנגשויות שמות בחיבור קודים – פגיעה ב-reuseability!
  - כל שם חדש מזהם את מרחב השמות העתידיים!
- הפתרון – העמסת פונקציות!

Difference between C and C++

Function overloading - העמסת פונקציות

- העמסת פונקציות = היכולת לתת למספר פונקציות שונות שם זהה (עם פרמטרים שונים).
  - הקומפיילר מזהה את הפונקציה לפי **החתימה** שלה.
  - בשפת C, החתימה של הפונקציה היא השם שלה בלבד.
  - בשפת C++, החתימה היא השם ורשימת הפרמטרים שלה.
    - רשימה שונה (באורך או בסוג הפרמטרים) = פונקציה שונה.
    - תוספת רשימת הפרמטרים היא זו שמאפשרת לנו את היכולת לבצע העמסת פונקציות.

Difference between C and C++

Function overloading - העמסת פונקציות

C++

- פונקציה מזוהה לפי שם ורשימת פרמטרים!

```
void swap(int*,int*);
void swap(float*, float*);
void swap(char*,char*);
```

- סוגי פרמטרים שונים יבדילו בין פונקציות שעובדות על טיפוסים שונים.

C

- פונקציה מזוהה לפי שם בלבד.

```
void swap_int(int*, int *);
void swap_float(float*, float*);
void swap_char(char*,char*);
```

- שמות שונים יבדילו בין פונקציות שעובדות על טיפוסים שונים.

שימוש בהעמסת פונקציות - דוגמא

- כיתבו את הפונקציה abs (מקבלת ערך ומחזירה את הערך המוחלט שלו) לטיפוסים int ו-double, לפי:
  - שפת C
  - שפת C++

- מהם ההבדלים העיקריים? מי לדעתכם קריא יותר?

Difference between C and C++

העמסת פונקציות - Function overloading

- בשפת C++ הפונקציה מזוהה ע"י שם הפונקציה + מספר וסוג הפרמטרים שהיא מקבלת.

- כאשר הקומפיילר מזהה קריאה לפונקציה הוא יחפש את הפונקציות בעלי אותו השם ובין ה"תוצאות" שלו – הוא יחפש פונקציה המתאימה מבחינת סוגי הפרמטרים.
  - אם יש יותר מפונקציה אחת מתאימה – תתואם הפונקציה ה"קרובה" (פרמטרים שאפשר לעשות implicit conversion ביניהם).
  - אם אין פונקציה מתאימה כלל או שיש יותר מפונקציה אחת מתאימה **באותה רמת "קירבה"** – נקבל שגיאת קומפילציה.

Difference between C and C++

Function overloading - העמסת פונקציות

- דוגמא:

```
int add(int a, int b){
    return a+b;
}
int add(int a, int b, int c){
    return a+b+c;
}
int add(int a, float b){
    return a+b;
}
```

❖ערך החזרה אינו חלק מחתימת הפונקציה, ולא ניתן להבדיל לפיו.

```
int add(int a, int b){
    return a+b;
}
float add(int a, int b){
    return a+b;
}
```

Difference between C and C++

Function overloading - העמסת פונקציות

- Overload Resolution - שלבים בחיפוש הפונקציה המתאימה
  - דוגמא:

```
int add(float a, int b);
int add(int a, float b);

int x,y;
add(x,y);
//error - more than one best viable function
        /* error message:
        call of overloaded `add(int&, int&)' is ambiguous
        candidates are: int add(float, int)
                        int add(int, float) */
add(x,(float)y);
//OK - calls the 2nd function

add((float)x,y);
//OK - calls the 1st function
```

Difference between C and C++

ערכי ברירת מחדל לפרמטרים

- אנו רוצים ליצור פונקציה שתחבר בין 2 int-ים.

```
int add(int a, int b);
```

- כעת אנו רוצים פונקציה נוספת שתחבר בין 3 int-ים.
  ראינו שניתן לעשות העמסה:

```
int add(int a, int b);
int add(int a, int b, int c);
```

- אבל עכשיו, אנחנו נאלצים לתחזק 2 פונקציות שבעיקרן מבצעות את אותו הדבר.
- הפתרון – ערכי ברירת מחדל!

Difference between C and C++

ערכי ברירת מחדל לפרמטרים

- ערכי ברירת מחדל = האפשרות לתת ערך דיפולטי במידה והמשתמש לא מכניס ערך משלו.
- נגדיר פונקציה אחת כאשר הערך השלישי יקבל ערך ברירת מחדל.

```
int add(int a, int b, int c=0){
    return a+b+c;
}
```

- כעת הפונקציה יכולה לקבל 2 ערכים וגם 3 ערכים.
- בפתרון זה יצרנו 2 פונקציות. אחת המקבלת 3 פרמטרים ואחת המקבלת 2 פרמטרים ומכניסה למשתנה c את הערך 0.

נשים לב לכך שנוצר לנו function overloading.

Difference between C and C++

ערכי ברירת מחדל לפרמטרים

- בשפת C++ ניתן להגדיר ערכי ברירת מחדל לפרמטרים של פונקציה.
- ערכי ברירת מחדל יוגדרו במקום אחד בלבד! בהצהרת הפונקציה (קובץ h.) ולא במימוש (קובץ cpp.). (אלו המוסכמות בכתיבת קוד נכון בעולם).
- הפרמטרים המקבלים ערך ברירת מחדל חייבים להופיע בסוף רשימת הפרמטרים (יכול להיות שכל הפרמטרים יקבלו ערכי ברירת מחדל).
  במילים אחרות: אם נתנו ערך ברירת מחדל לפרמטר כלשהו, אנחנו חייבים לתת ערך ברירת מחדל לכל הפרמטרים המופיעים אחריו (מימינו).
- הדבר מאפשר השמטת הפרמטר בזמן הקריאה לפונקציה.
- אם משמיטים פרמטר אחד – יש להשמיט את כל הפרמטרים שאחריו (מימינו).

Difference between C and C++

ערכי ברירת מחדל לפרמטרים

- דוגמא:

file.h

```
int func (int a=10, int b=20, int c=30);
```

file.cpp

```
int func (int a, int b, int c)
{
    return a+b+c;
}
```

- כמה פונקציות נוצרו לנו?

```
int func (int a, int b, int c);
int func (int a, int b)
int func (int a)
int func ()
```

Difference between C and C++

ערכי ברירת מחדל לפרמטרים

- דוגמא:

file.h

```
int func (int a=10, int b=20, int c=30);
```

file.cpp

```
int func (int a, int b, int c)
{
    return a+b+c;
}
```

- מה יהיה הערך של x אחרי כל אחת מהקריאות הבאות:
  - x=func(1,2,3) 6
  - x=func(1,2) 33
  - x=func(1) 51
  - x=func() 60
  - ~~x=func(1, ,3)~~ Error!

Difference between C and C++

ערכי ברירת מחדל לפרמטרים

- כיתבו פונקציה אחת, תוך שימוש בערכי ברירת מחדל כך שניתן יהיה להשתמש בה גם כפונקציית מכפלה של 3 ערכים וגם של 2.

Difference between C and C++

משתנה בוליאני

- בשפת C, השתמשנו במשתנים מסוג int (enum) או char כדי לייצג ערכים בוליאניים (0 = שקר, כל דבר אחר = אמת).
- בשפת C++ נוסף סוג משתנה חדש - משתנה בוליאני (bool). משתנה זה מכיל 2 ערכים אפשריים: אמת \ שקר (true \ false).
  - דוגמא:

```
bool isRich (int money){
    bool rich = money > 200000;
    if (rich) //or if (rich==true)
        return true;
    else //if (!rich) //or if (rich==false)
        return false;
}
```

Difference between C and C++

משתנה בוליאני

- ישנן 2 סיבות עיקריות לשימוש במשתנה בוליאני.
  - קריאות הקוד.
    שימוש במשתנה בוליאני מאפשר לנו, למשל, לשמור תוצאות של בדיקות שביצענו בתוך משתנה בעל שם המייצג את אותה הבדיקה.
  - חיסכון במקום.
    בעוד שמשתנה int תופס 4 בתים – משתנה bool תופס בית (byte) אחד בלבד.

Difference between C and C++

הקצאת זיכרון דינאמי

- בשפת C השתמשנו בפונקצית malloc לצורך הקצאת מקום עבור משתנים כלשהם בצורה דינאמית (בזמן הריצה) ובפונקציה free ע"מ לשחרר את המקום שהקצאנו.
- כאשר הקצאנו זיכרון בצורה דינאמית – היינו נדרשים לקבוע בדיוק מה כמות המקום שנצטרך (בד"כ, בעזרת שימוש נרחב בפונקציה sizeof()).
  - היינו צריכים לבצע casting לסוג המשתנה שרצינו.
- בשפת C++ יש לנו את האופרטורים new ו-delete למטרה זו.

Difference between C and C++

הקצאת זיכרון דינאמי

C

```
#include <malloc.h>

int *pi = (int*)malloc(sizeof(int));

if(pi == NULL){
          ... /* out of memory */
}
*pi = 6;
...
free(pi);
```

C++

```
int main() {
     int *pi = new int;
     *pi=6;
     delete pi;
     …
     return 0;
}
```

- אין צורך להגיד כמה מקום בדיוק להקצות (הקומפיילר לוקח את המידע הזה מסוג המשתנה).
- לא צריך לבצע casting לסוג המשתנה שעבורו הוקצה הזיכרון.

Difference between C and C++

הקצאת זיכרון דינאמי - מערכים

- הקצאת מערך בצורה דינאמית בשפת C (בשימוש בפונקצית malloc) התבצעה בצורה דומה להקצאת משתנה יחיד. ההבדל היה בכך שכאשר הקצנו מקום – הכפלנו את המקום שהקצנו במספר התאים שרצינו.

```
int arrSize = 10;
int *pi = (int*)malloc(sizeof(int) * arrSize);
if(pi == NULL){ ... /* out of memory */ }
pi[3] = 6;
...
free(pi);
```

- גם כאן...
  - היינו צריכים לבצע casting לסוג המשתנה שרצינו.

Difference between C and C++

הקצאת זיכרון דינאמי - מערכים

- לעומת זאת, בשפת C++, החיים הרבה יותר קלים.

```
int main() {
        int arraySize = 10;
        int *pi = new int[arraySize];
        pi[7]=4;
        …
        delete[] pi;
        return 0;
}
```

- גם כאן...
  - אין צורך להגיד כמה מקום בדיוק להקצות (הקומפיילר לוקח את המידע הזה מסוג המשתנה).
  - לא צריך לבצע casting לסוג המשתנה שעבורו הוקצה הזיכרון.

Difference between C and C++

הקצאת זיכרון דינאמי - מערכים

- מספר דגשים:
  - טכנית, ניתן (אך מאוד לא רצוי, ואף **אסור** בקורס זה) להקצות זיכרון דינאמי בשפת C++ גם בעזרת פונקציית malloc וכן לשחרר את הזיכרון בעזרת free.
  - בכל מקרה, אסור לערבב בין אופרטורים new\delete לבין הפונקציות malloc\free.
  - יש לשים לב לשחרר הקצאה של new בעזרת delete והקצאה של new[] בעזרת delete[].
- מותר להפעיל delete על מצביע ל-NULL. ע"פ תקן השפה מדובר בפעולה חוקית שפשוט לא מבצעת כלום.

Difference between C and C++

Reference

- בשפת C יש שתי דרכים להעברת משתנה – by value (מעתיק ערך) ו- by address שיוצר משתנה חדש (מצביע) המכיל את הכתובת ומכריח אותנו להתמודד עם אופרטור הכתובת (&) ואופרטור ה-indirection (*).

- בשפת C++ קיים סוג משתנה מעניין: ה- reference.
  ה- reference הוא בעצם מתן שם נוסף למשתנה.

- דוגמא:

```
int i=1;
int& j=i;
```

  - בדוגמא זו, j הוא פשוט שם נוסף למשתנה i. (שימו לב לא להתבלבל עם &j שהוא בעצם הכתובת של המשתנה j).
  - כל פעולה שנבצע על i תשפיע גם על j.

Difference between C and C++

Reference

- דוגמא נוספת:
  - נשים לב לכך ש-i ו-j תמיד יהיו עקביים אחד עם השני. שינוי של אחד משפיע על השני.
  - k לעומת זאת, מקבל פעם אחת את הערך שהופיע ב-i ואינו מושפע משינויים עתידיים של i.

```
int i = 10 ;
int& j = i;
int k = i;
i = 7; // i=j=7, k=10
j++; // i=j=8, k=10
j = k; // i=j=10, k=10
j = 4; // i=j=4, k=10
```

- כאשר אנחנו מגדירים משתנה reference – אנחנו חייבים גם לאתחל אותו.
  לאחר האיתחול – לא ניתן לשנות את ההפניה למשתנה אחר!
  - עם זאת, ניתן לשנות את הערך של המשתנה שאליו אנחנו מפנים.

Difference between C and C++

Reference parameters

- דרך נוספת (ומאוד שימושית) לנצל את יכולות ה-reference היא בהעברת פרמטרים: by reference.

- הרעיון זהה למשתני reference, כאשר ההבדל הוא שאנחנו משתמשים במשתנים המוגדרים עבור אותה הפונקציה .
  - פרמטרים שנשלחים לפונקציה הם בעצם משתנים מקומיים לפונקציה שנוצרים בתחילת הפונקציה (עם הערכים שנשלחו) ומתים בסופה.
  - נוכל לשלב עיקרון זה עם reference parameters.

Difference between C and C++

Reference parameters

- דוגמא:
  - ניזכר בפונקציה isRich שראינו.

```
bool isRich (int money){
        bool rich = money > 20000;
        if (rich) //or if (rich==true)
                return true;
        else //if (!rich) //of if (rich==false)
                return false;
}
```

  - כאשר נקרא לפונקציה ע"י הפקודה isRich(15000);, הדבר הראשון שהפונקציה מבצעת היא פעולת ההשמה של הפרמטר, כלומר – int money=15000;.
לאחר מכן הפונקציה ממשיכה הלאה לביצוע שאר הפקודות בפונקציה.
  - (העברה by value).

Difference between C and C++

Reference parameters

- דוגמא:
  - נרצה לבצע החלפה של 2 ערכים של משתנים.

```
void swap(int& a, int& b){
    int temp=a;
    a=b;
    b=temp;
}
```

  - מה בעצם יקרה כאן?
  - כאשר נקרא לפונקציה ע"י הפקודה swap (x,y); הדבר הראשון שהפונקציה מבצעת היא את פעולות ההשמה של הפרמטרים, כלומר – int& a=x; ו- int& b=y;.
לאחר מכן הפונקציה ממשיכה הלאה לביצוע שאר הפקודות בפונקציה.
  - יצרנו בפונקציה reference ל-x ול-y ולכן השינויים בתוך הפונקציה יישמרו גם מחוץ לה.

Difference between C and C++
הבדלים בהעברת reference למצביע פרמטר

by reference

```
void swap(int& a, int& b){
    int temp=a;
    a=b;
    b=temp;
}
```

- נוצר לנו רק שם חדש למשתנה שהועבר כפרמטר.
- תחביר פשוט כמו by value.

by pointer

```
void swap(int* a, int* b){
    int temp=*a;
    *a=*b;
    *b=temp;
}
```

- נוצר משתנה חדש מסוג int* המצביע על אותה כתובת כמו המשתנה שהועבר כפרמטר.
- תחביר מסובך של מצביעים.

Difference between C and C++

Reference parameters

- שימו לב:
  - אחת השגיאות הנפוצות בנושא זה היא החזרת reference למשתנה מקומי (לוקאלי).
  - גרוע בדיוק כמו החזרה by address של משתנה מקומי!
  - בדוגמא זו אנחנו מחזירים reference למשתנה x, אבל משתנה זה ימות ביציאה מהפונקציה!

```
int& add(int a, int b){
        int x = a+b;
        return x;
}
```

Difference between C and C++

הבדלים בין reference למצביע

מצביע	reference
לא חייב להיות מאותחל.	חייב להיות מאותחל.
יכול להצביע לערך NULL.	לא יכול להפנות לערך NULL.
לאחר ההשמה הראשונית – ניתן לשנות את משתנה אליו אנחנו מצביעים.	לאחר האתחול – לא נוכל לשנות את המשתנה שאיתו אנחנו מתואמים.
המצביע תופס מקום בזיכרון.	לעיתים, אין צורך במקום נוסף בזיכרון.
	סינטקס ידידותי יותר.

Difference between C and C++

const -ל reference

- נניח שיש לנו בקוד משתנה מסוג int בשם a, אז:
  - **const int & pc = a;** (או **int const & pc = a;**) –
    pc הוא reference לאיבר שנתייחס אליו כאל const.
    a לא באמת חייב להיות const! פשוט לא ניתן לשנות את ערכו דרך pc.
  - **int & const cp = a;** – שגיאת קומפילציה!
    reference חייב להיות מאותחל למשתנה ספציפי ולא ניתן לשנות את ההפניה במהלך הקוד ולכן ההצהרה הזו היא חסרת משמעות.

Difference between C and C++

reference

- כתבו את הפונקציה abs_change של int כך שהיא אינה מחזירה ערך אלא משנה את הפרמטר שלה להיות הערך המחלט שלו עצמו. השתמשו ברפרנס.

Difference between C and C++

קלט ופלט

- אחת התוספות העיקריות לשפת C++ היא היכולת לעבוד בשיטת OOP, כלומר – שימוש במחלקות ואובייקטים. נרחיב על כך בהמשך.
- שפת C++ מגיעה עם מס' אובייקטים בנויים כבר בתוכה, ביניהם:
  - אובייקט של המחלקה istream בשם cin - אובייקט קלט.
  - אובייקט של המחלקה ostream בשם cout – אובייקט פלט.
- ע"מ להשתמש בהם יש להוסיף לקוד את:

```
#include <iostream>
using namespace std;
```

(בקשה לצרף את הספרייה המכילה את האובייקטים המתאימים)

Difference between C and C++

קלט ופלט

- עבור המחלקה istream קיים האופרטור >>

```
cin >>x; //x var. Like an arrow showing data flow direction.
```

- עבור המחלקה ostream קיים האופרטור <<

```
cout<<x; //same, only the direction is reverse!
```

- אופרטורים אלו משמשים אותנו לקלוט מידע מתוך המקלדת ולכתוב מידע את המסך.
- השימוש באופרטורים אלה מאפשר כתיבה למסך וקריאה מהמקלדת בצורה נוחה יותר ואינטואיטיבית יותר מאשר בשפת C.

Difference between C and C++

קלט ופלט

- דוגמא:

```
char c;
cin>>c;
```

- במקרה זה, כאשר הקוד יגיע לשורה השנייה (לפני הביצוע שלה), התוכנית תיעצר ותמתין לקלט מהמקלדת.
- בשיטה זו נוכל לקלוט תו אחד. אך מה נעשה כשנרצה לקלוט מספר?

```
int n;
cin>>n;
```

- נשים לב שיש לנו העמסת פונקציות!
  - לפי סוג הפרמטר הוא יודע איזה פונקציית קליטה להפעיל!

Difference between C and C++

קלט ופלט

```
char c;     int n;
cin>>c;     cin>>n;
```

- האופרטור >> מפעיל את פונקציית הקריאה מהמקלדת.
  כאשר הפונקצייה מקבלת פרמטר שהוא char תופעל הפונקציה המתייחסת לפרמטר מסוג char – והיא תכניס אליו את התו הראשון שהיא קיבלה.
  - במידה והיו תווים נוספים – הם יישמרו ב-buffer וישמו לתוך משתנים בפעמים הבאות שנקרא מהמקלדת (ולכן יש צורך לנקות את האנטר!).
- כאשר הפונקצייה מקבלת פרמטר שהוא int תופעל הפונקציה המתייחסת לפרמטר מסוג int– והיא תכניס אליו את מספר הראשון שהיא קיבלה.
  - במקרה זה, אנטר או רווח יפרידו בין מספר למספר, ואם הוכנסו 2 מספרים עם רווחים ביניהם – המספר הראשון יוכנס למשתנה והמספר השני יישמר ב-buffer (ושוב האנטר בסיום נשמר ב-buffer!).

Difference between C and C++

קלט ופלט

```
char c='a';
cout<<c;
```

```
int c=7;
cout<<c;
```

- דבר דומה קורה עבור cout.
- האופרטור >> מפעיל את פונקציית ההדפסה למסך.

כאשר הפונקצייה מקבלת פרמטר שהוא char תופעל הפונקציה המתייחסת לפרמטר מסוג char – והיא תדפיס למסך את התו המופיע במשתנה.

- כאשר הפונקצייה מקבלת פרמטר שהוא int תופעל הפונקציה המתייחסת לפרמטר מסוג int– והיא תדפיס למסך את המספר המופיע במשתנה.

Difference between C and C++

קלט ופלט

- ניתן גם לשרשר כתיבה למסך וקריאה מהמקלדת.
- לדוגמא:

```
int x,y;
cin>>x>>y;
cout<<"x="<<x<<" y="<<y<<endl;
```

  - תוכנית זו תקלוט 2 מספרים מהמקלדת ותדפיס אותם.
  - ההפרדה היא על ידי רווח, אנטר או טאב.
  - מה יהיה הפלט אם המספר הראשון שנכניס יהיה 123 והמספר השני שנכניס יהיה 321?
  - x=123 y=321

- לאובייקטים cin, cout יש פונקציות נוספות. הנכם מוזמנים, בזמנכם החופשי, לשחק מעט עם אובייקטים אלה ולהכיר את היכולות השנות שלהם.

Difference between C and C++

קלט ופלט

C

```
int x,y;
scanf("%d %d", &x, &y);
printf("x=%d y=%d\n",x ,y);
```

C++

```
int x,y;
cin>>x>>y;
cout<<"x="<<x<<" y="<<y<<endl;
```

Difference between C and C++

דוגמאות כלליות

- ניזכר לרגע בפונקציה swap(x,y) שהייתה לנו:

```
void swap(int& a, int& b){
    int x=a;
    a=b;
    b=x;
}
```

- עבור הקלט 123 456
מה ייתן קטע הקוד הבא?

```
int x,y;
cin>>x>>y;
swap(x,y);
cout<<"x="<<x<<" y="<<y<<endl;
```

- x=123 y=456

Difference between C and C++

דוגמאות כלליות

- ואם נשנה את swap(x,y) לצורה הבאה:

```
void swap(int a, int& b){
    int x=a;
    a=b;
    b=x;
}
```

- עבור הקלט 123 456
מה ייתן קטע הקוד הבא?

```
int x,y;
cin>>x>>y;
swap(x,y);
cout<<"x="<<x<<" y="<<y<<endl;
```

- x=456 y=456

Difference between C and C++

דוגמאות כלליות

- אם נשנה את הפונקציה swap(x,y) לצורה הבאה:

```
void swap(const int& a, const int& b){
    int x=a;
    a=b;
    b=x;
}
```

- מה יקרה?

- שגיאות קומפילציה בשורות השנייה והשלישית– אנו מנסים לשנות משתנים קבועים!

Difference between C and C++

דוגמאות כלליות

- כיתבו מערכת המבצעת את הדברים הבאים:
  - מבקשת מהמשתמש את גודל שמו (מספר האותיות)
  - מקצה מקום גדול מספיק לשם (אך לא גדול מידי).
  - קולטת למקום שהכינה את השם.
  - מדפיסה למסך את השם.
  - ולבסוף יוצאת בצורה מסודרת (תוך שחרור כל המשאבים).

Difference between C and C++

Hello World!

C

```
#include <stdio.h>


int main() {
   printf("Hello World! \n");
   return 0;
}
```

C++

```
#include <iostream>
using namespace std;

int main() {
   cout << "Hello World!"
        << endl;
   return 0;
}
```