תכנות מכוון עצמים ו- C++
יחידה 01
מ- C ל- C++

קרן כליף

ביחידה זו נלמד:

- הגדרת תכנות מכוון עצמים
- השוני הסינטקטי בין C ל- C++:
  - printf cout
  - scanf cin
  - gets cin.getline
  - malloc new
  - free delete
- טיפוס התייחסות
- מרחבי שמות (namespace)

מה אנחנו יודעים? תכנות פרוצדורלי

- בקורס C למדנו לתכנת במתודלוגיה הנקראת תכנות פרוצדורלי
  - התכנות היה בסגנון Top-Down
  - הדגש בתוכניות היה על פונקציות והמידע המועבר בינהן
- דוגמא:
  - אם רצינו לטפל במטריצה, היינו רושמים את הפונקציות הבאות:
    - פונקציה שיודעת להקצות מטריצה
    - פונקציה שיודעת לקלוט נתונים למטריצה
    - פונקציה שיודעת להדפיס את המטריצה
    - פונקציה שיודעת לשחרר את נתוני המטריצה
  - כל אחת מהפונקציות הנ"ל הייתה צריכה לקבל את המטריצה ומימדיה כפרמטרים

מה נלמד? תכנות מכוון עצמים

- בקורס זה נלמד מתודולוגית תכנות הנקרא "תכנות מכוון עצמים" (Object Oriented, או בקיצור OO)
  - הדגש יהיה על האובייקטים שיש במערכת, מה המידע שיש לכל אובייקט, ומה הפעולות שכל אובייקט יודע לבצע
  - כמו struct של C, אבל בנוסף לשדות, יהיו גם פעולות
- מתודולוגיית תכנות זו מייצגת כיצד העולם שלו בנוי
- דוגמא:
  - לסטודנט יהיו את השדות: ת.ז, שם, תאריך לידה, מגמת לימוד וממוצע
  - בנוסף, הוא ידע לבצע את הפעולות הבאות: להירשם לקורס, להדפיס את נתוניו, ללמוד למבחן, להכין מטלות וללכת לים

לתכנות מכוון עצמים 3 עקרונות מרכזיים

1. **הסתרה** (encapsulation): כל הנתונים והפעולות הקשורות לישות מסוימת מרוכזות יחדיו. המשתמש עובד עם "קופסא שחורה".

   קל לתחזק את הקוד ולהתמצא בו, תחזוקה פשוטה

2. **הורשה** (inheritance): הרחבה של ישות קיימת כדי למנוע שכפול קוד, או לחילופין כדי לתת מימוש אלטרנטיבי לקוד קיים.

   למשל: ל- person יש אוסף נתונים, ול- student יש בדיוק אותם נתונים ועוד כמה נוספים. לא נרצה לשכפל את כל הקוד שיש ב- person..

3. **רב-תצורתיות** (פולימורפיזם, polymorphism): מאפשר להתייחס לישויות שונות בעלי בסיס זהה באותו אופן.

   למשל: החזקת מערך של צורות, כאשר חלק מהצורות הן ריבוע, חלקן עיגול או משולש, ולהיות מסוגלים להדפיס את כולן.

תכנות מכוון עצמים ו- C++

בנוסף ל- 3 העקרונות שראינו קודם, בשפת C++ יש 2 עקרונות נוספים:

1. **תבניות** (templates): כלי המאפשר לכתוב קוד כללי לטיפוסים שונים.

   <u>דוגמא</u>: האלגוריתם למיון קבוע לכל טיפוס, אבל המערך המתקבל ופעולות ההשוואה מבוצעות על טיפוסים שונים.

   במקום לשכפל את הקוד עבור טיפוס שונה כל פעם, ניתן לכתוב פונקציה כללית אחת שתדע לטפל בכל טיפוס.

2. **חריגות (exceptions):** מנגנון לטיפול שגיאות בזמן ריצה.

מ- C ל- C++

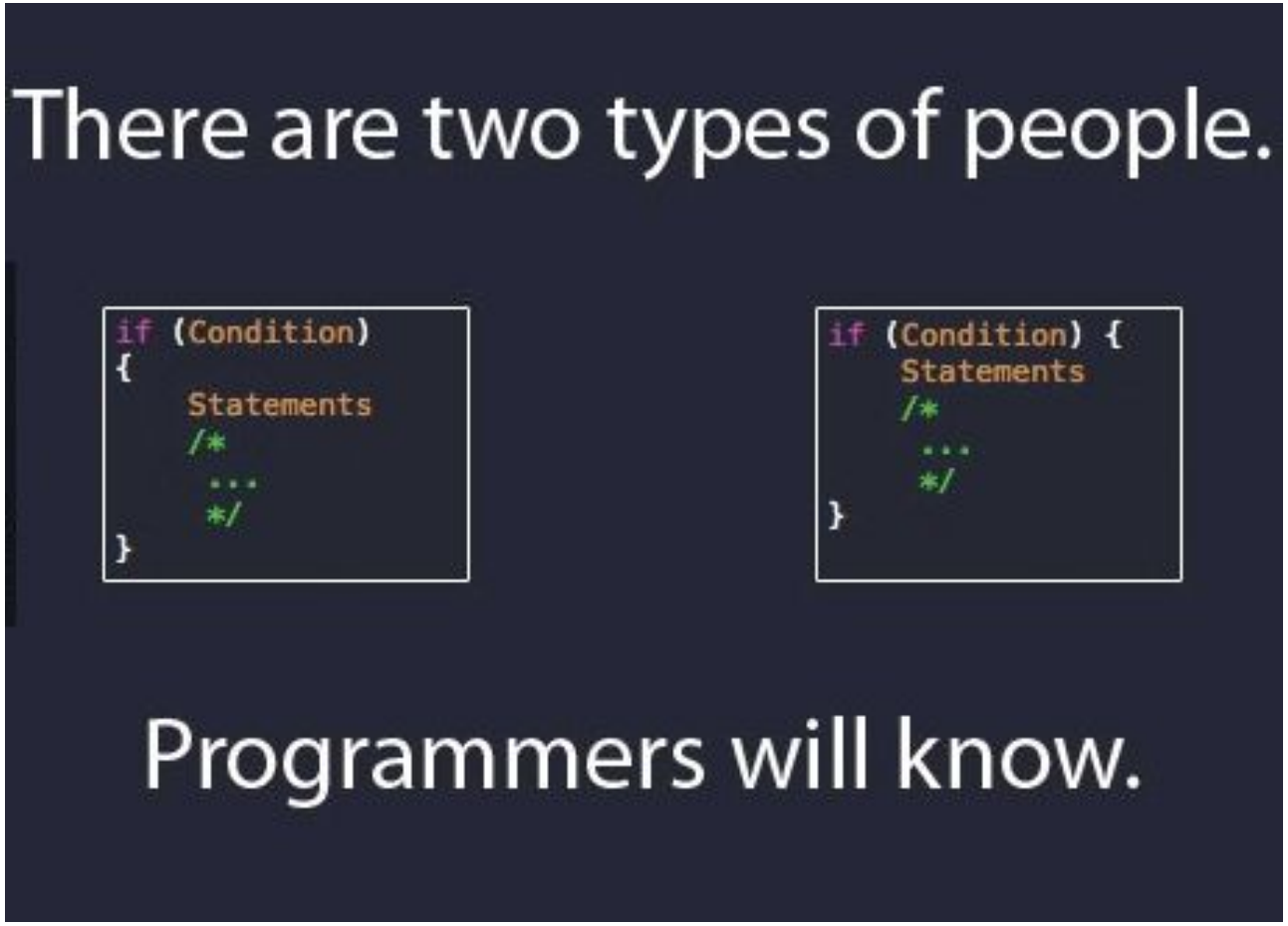
- שפת C++ מאוד דומה סינטקטית לשפת C, אך יחד עם זאת יש כמה שינויים:
  - נפתח פרויקט באותו אופן כמו בקורס C, אבל נייצר קובץ עם סיומת cpp (ברירת המחדל), ולא עם סיומת c
    - הקומפיילר שונה, ולכן יתכנו שגיאות קומפילציה טיפה שונות
  - ניתן להגדיר משתנים בכל חלק בתוכנית, ולא רק בתחילת בלוק
  - במקום הכללת הספריה *stdio.h* שהכילה פקודות קלט-פלט, נכליל את הספריה *iostream* ונוסיף *using namespace std;* (הסבר בהמשך)
  - קיים הטיפוס **bool** שמחזיק את הערכים **true** או **false**

מ- C ל- C++ (2)

5. פקודות שונות לטיפול בקלט ופלט:
   - במקום הפקודה printf נשתמש בפקודה cout
   - במקום הפקודה scanf נשתמש בפקודה cin

6. פקודות שונות לטיפול בהקצאות ושחרור זכרון:
   - במקום הפקודות malloc/calloc נשתמש בפקודה new
   - במקום הפקודה free נשתמש בפקודה delete

אבל ראשית, שאני לא אתעצבן..



http://qph.is.quoracdn.net/main-qimg-e0c9dafb319150b6c6d9816047ed9eae?convert_to_webp=true

הפקודה cout

- יודעת להדפיס נתונים למסך (Console OUT)
- הסינטקס הרבה יותר פשוט מ- printf: אין צורך להבדיל בהדפסה בין הטיפוסים השונים, משרשרים את חלקי המחרוזת להדפסה באמצעות <<

```
#include <stdio.h>


void main()
{
    char str[] = "hi";
    int num = 5;
    printf("string: %s number: %d\n",
       str, num);
{
```

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

void main()
{
    char str[] = "hi";
    int num = 5;
    cout << "string: " << str << " number: "
      << num << "\n";
}
```

הפקודה cin

- יודעת לקרוא נתונים מהמקלדת (Console IN)
- הסינטקס הרבה יותר פשוט מ- scanf: אין צורך להבדיל בקליטה בין הטיפוסים השונים, משרשרים את הנתונים השונים שרוצים לקרוא באמצעות <<

```
#include <stdio.h>


void main()
{
    char str[10]
    int num;
    printf("Enter string and number: ");
    scanf("%s%d", str, &num);
{
```

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

void main()
{
    char str[10];
    int num;
    cout << "Enter string and number: ";
    cin >> str >> num;
}
```

הפקודה cin.getline

- cin יודעת לקרוא מחרוזת עד רווח
- ע"י שימוש ב- cin.getline ניתן לקלוט תווים עד ENTER או עד למקסימום תווים

```
#include <stdio.h>


void main()
{
    char str[10];
    printf("Enter string: ");
    gets(str);
{
```

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

void main()
{
    char str[10];
    cout << "Enter string: ";
    cin.getline(str, 5);
}
```

הפונקיה תקרא עד 4 תווים
(אחד עבור ה- '\0')
או עד ENTER

הפקודות new ו- delete

- הפקודה new להקצאה דינאמית, מקצה מערך עם ערכי זבל (מקבילה ל- malloc)
  - אין מקבילה ל- calloc
  - אין מקבילה ל- realloc
- הפקודה delete לשחרור זכרון. אם משחררים מערך יש לשחרר עם []

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int *arr, numOfElements;

    printf("How many elements? ");
    scanf("%d", &numOfElements);
    arr = (int*)malloc(numOfElements*sizeof(int));
    free(arr);
}
```

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

void main()
{
    int *arr, numOfElements;

    cout << "How many elements? ";
    cin >> numOfElements;
    arr = new int[numOfElements];
    delete []arr;
}
```

דוגמא

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int i, numOfNumbers;
    int* arr;

    printf("How many numbers? ");
    scanf("%d", &numOfNumbers);

    arr = (int*)calloc(numOfNumbers, sizeof(int));
    for (i=0 ; i < numOfNumbers ; i++)
        printf("Value #%d: %d\n", i+1, arr[i]);

   free(arr);
{
```

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

void main()
{
    int  numOfNumbers;


    cout << "How many numbers? ";
    cin >> numOfNumbers;

    int* arr = new int[numOfNumbers];
    for (int i=0 ; i < numOfNumbers ; i++)
      cout << "Value #" << i+1 << ": " << arr[i] << "\n";

    delete []arr;
}
```

נשים לב להגדרת המשתנים באמצע התוכנית

דוגמא: תוכנית המטפלת במטריצה

```
1.  #include <iostream>
2.  using  namespace  std;
3.
4.  const  int N = 3;
5.  const  int M = 4;
6.
7.  // prototypes
8.  int**  allocateMatrix  (int rows, int cols);
9.  void  enterInput       (int** mat, int rows, int cols);
10. void  printMatrix      (int** mat, int rows, int cols);
11. void  freeMatrix       (int** mat, int rows);
12.
13. void main()
14. {
15.   int** mat = NULL;
16.
17.   mat = allocateMatrix(N, M);
18.   enterInput(mat, N, M);
19.   printMatrix(mat, N, M);
20.   freeMatrix(mat, N);
21. {
```

ה- main כתוב בראשי פרקים וניתן להבין בקלות מה קורה בתוכנית

דוגמא: תוכנית המטפלת במטריצה (2)

```
22.  int**  allocateMatrix(int rows, int cols)
23.  }
24.    int** mat = new int*[rows];  // allocating the rows
25.    for (int i=0 ; i < rows ; i++)
26.          mat[i] = new int[cols];  // allocating the columns
27.
28.    return mat;
29.  {
30.
31.  void  enterInput(int** mat, int rows, int cols)
32.  }
33.    for (int i=0 ; i < rows ; i++)
34.          for (int j=0 ; j < cols ; j++)
35.                mat[i][j] = i*cols + j;
36.  {
```

בתחילה מקצים מערך של int*, עבור השורות

נשים לב לאפשרות הגדרת המשתנים בכל חלק בקוד

דוגמא: תוכנית המטפלת במטריצה (3)

```
37.  void printMatrix(int** mat, int rows, int cols)
38.  {
39.    for (int i=0 ; i < rows ; i++)
40.    }
41.              for (int j=0 ; j < cols ; j++)
42.             cout << mat[i][j] << ", ";
43.              cout << "\b\b \n";
44.    {
45.  {
46.
47.  void freeMatrix(int** mat, int rows)
48.  }
49.    for (int i=0 ; i < rows ; i++)
50.          delete []mat[i];
51.
52.    delete []mat;
53.  {
```

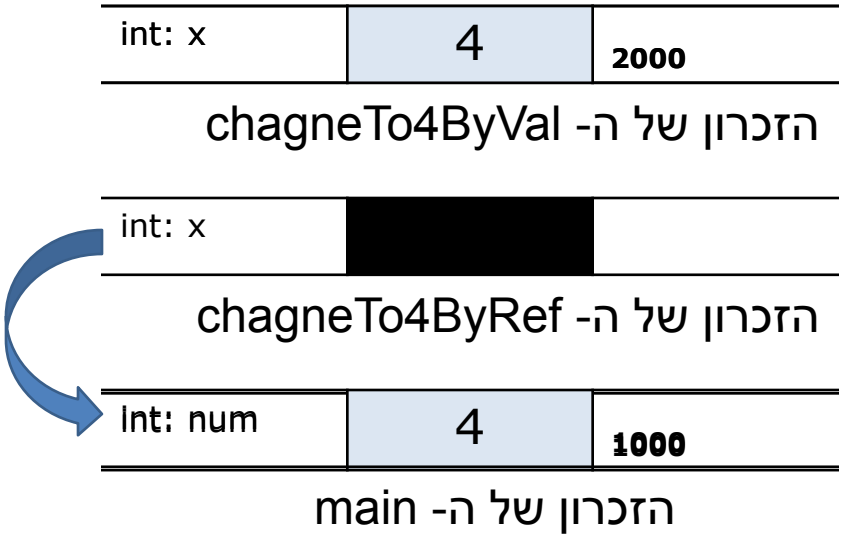
נשים לב: שחרור מערך עם ציון [ ]

טיפוס התייחסות

- בשפת C כאשר רצינו שפונקציה תשנה את ערכו של ארגומנט מסוים, העברנו את הכתובת שלו (העברה by pointer, לעומת העברת העתק, by value)
- בשפת C++ עדיין ניתן להעביר מצביעים כדי לשנות ארוגמנטים בפונקציות, אך יש דרך חדשה הנקראית העברת פרמטרים by reference
- בהגדרת הפרמטרים שהשיטה מקבלת מציינים ליד הפרמטר &
  - זהו למעשה מתן שם נוסף לארגומנט המקורי שנשלח הנגיש מתוך הפונקציה
- אין הבדל סינטקטי בשליחת המשתנה בעבור משתנה שעובר by value או by reference

שליחת פרמטר by ref לעומת by val

```
1.  void changeTo4byVal(int  x)
2.  {
3.    x = 4;
4.  {
5.
6.  void changeTo4byRef(int&  x)
7.  }
8.    x = 4;
9.  }
10.
11. void main()
12. }
13.   int num = 10;
14.
15.   cout << "orig num = " << num << endl;
16.
17.   changeTo4byVal(num);
18.   cout << "after changeTo4byVal: num = " << num << endl;
19.
20.   changeTo4byRef(num);
21.   cout << "after changeTo4byRef: num = " << num << endl;
22.  {
```



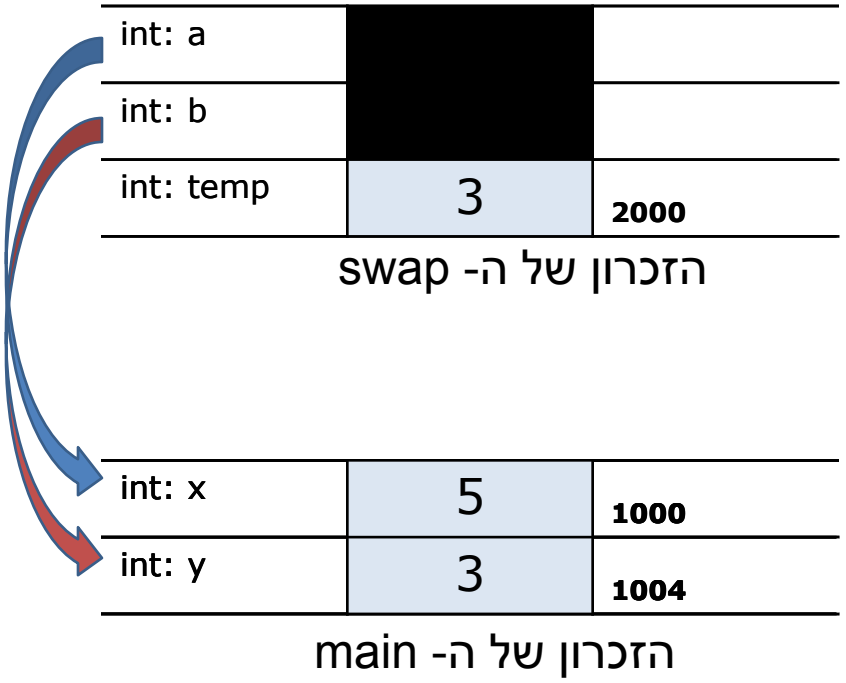
הדוגמא swap

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

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

void main()
}
    int x=3, y=5;

    cout << "Before swap: x = " << x << ", y = " << y << "\n";
    swap(x, y);
    cout << "After swap:  x = " << x << ", y = " << y << "\n";
{
```



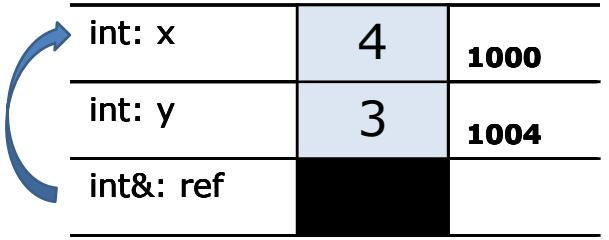
משתנה מטיפוס reference

```
0012FF60 0012FF60
x=5, y=3, ref=5
x=6, y=3, ref=6
x=3, y=3, ref=3
x=4, y=3, ref=4
```

- מתן שם נוסף למשתנה כלשהו
- חייב להיות מאותחל
- לא ניתן לייצר מערך של הפניות
- אינו תופס מקום נוסף, ולכן כתובתו כמו כתובת המשתנה אליו הוא מפנה

```
void main()
{
    int x = 5, y = 3;
    int& ref = x;

    cout << &ref << " " << &x << endl;
    cout << "x=" << x << ", y=" << y << ", ref=" << ref << endl;
    x++;
    cout << "x=" << x << ", y=" << y << ", ref=" << ref << endl;
    ref = y;
    cout << "x=" << x << ", y=" << y << ", ref=" << ref << endl;
    ref++;
    cout << "x=" << x << ", y=" << y << ", ref=" << ref << endl;
}
```



הזכרון של ה- main

החזרת טיפוס התייחסות מפונקציה

- בשפת C יכולנו להחזיר כתובת של משתנה מתוך פונקציה, וכך למעשה החזרנו את המשתנה המקורי, ולא העתק שלו
- בשפת C++ עדיין ניתן להחזיר משתנה by pointer, אך ניתן גם להחזיר משתנה by reference
- יש לשים לב שכשמחזירים משתנה by reference שהוא עדיין יהיה קיים ביציאה מהפונקציה (בדיוק כמו בשפת C)

דוגמא להחזרת משתנה by ref מפונקציה

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

void printArr(int arr[], int size)
{
    for (int i=0 ; i < size ; i++)
       cout << arr[i] << " ";
    cout << endl;
{
```

```
arr after changing max:
 6 80 2
```

הפונקציה מחזירה הפניה לאיבר המקסימלי

```
int& getMax(int arr[], int size)
}
    int maxIndex = 0;
    for (int i=1 ; i < size ; i++)
       if (arr[i] > arr[maxIndex])
              maxIndex = i;
    return arr[maxIndex];
{
```

```
void main()
}
    int arr[] = {6,8,2};
    int size = sizeof(arr)/sizeof(arr[0]);

    getMax(arr, size) *= 10;
    cout << "arr after changing max:\n ";
    printArr(arr, size);
```

int[]: arr	6	1000
	80	1004
	2	1008
int: size	3	1012

הזכרון של ה- main

int[]: arr	1000	1000
int: size	3	1004
int: maxIndex	1	1012

הזכרון של ה- getMax

מרחבי שמות (namespace)

- כאשר עובדים על פרויקט גדול, לרוב משתמשים בקוד מוכן, וכל מתכנת כותב את חלקו
- יתכן מצב שיהיו 2 פונקציות בעלות שם זהה המקבלות את אותם נתונים
- תיוצר בעיה של התנגשות בשמות, הקומפיילר לא ידע לאיזה פונקציה לפנות
- <u>הפתרון</u>: שימוש ב- namespace
  - השימוש ב- namespace מאפשר קישור של פונקציה מסוימת לחבילת קוד מסוימת

דוגמא:
שימוש ב- namespace

```
This is the first foo
This is the second foo
This is just foo
```

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  namespace first
5.  {
6.      void foo()     }cout << "This is the first foo\n“; {
7.  {
8.
9.  namespace second
10. }
11.      void foo()    }cout << "This is the second foo\n“; {
12. {
13.
14. void foo()    }cout << "This is just foo\n“; {
15.
16. void main()
17. {
18.   first::foo();
19.   second::foo();
20.         foo();
21. {
```

- להלן קטע קוד עם 3 פונקציות זהות
- 2 מימושים נמצאים בתוך namespace שונה
- פניה לפונקציה הנמצאת בתוך namespace מחייבת ציון שם ה- namespace שבתוכו היא נמצאת
- פונקציה שלא בתוך napespace נמצאת במרחב השמות הגלובלי

קיצור אופן השימוש ב- namespace

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  namespace first
5.  {
6.    void foo()    {cout << "This is the first foo\n";}
7.  }
8.
9.  namespace second
10. }
11.   void foo()    {cout << "This is the second foo\n";}
12. {
13.
14. using  namespace  second;
15.
16. void main()
17. }
18.   first::foo();
19.   second::foo();
20.   foo();
21. {
```

פקודה זו מאפשרת לנו לפנות לפונקציות שתחת namespace זה בלי הקידומת

קיצור אופן השימוש ב- namespace

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  namespace first
5.  {
6.    void foo()    {cout << "This is the first foo\n";}
7.  }
8.
9.  namespace second
10. }
11.   void foo()    {cout << "This is the second foo\n";}
12. {
13.
14. void foo()    }cout << "This is just foo\n“; {
15.
16. using namespace first;
17. using namespace second;
18. void main()
19. }
20.   first::foo();
21.   second::foo();
22.   foo();  // ERROR!
23.         ::foo();
24. {
```

במקרה זה נהייה חייבים תמיד לפנות בשם המלא של הפונקציה, אחרת נקבל את השגיאה: ambiguous call to overloaded function **מאחר והקומפיילר אינו יודע לאיזו פונקציה לפנות**

פניה בשם המלא לפונקציה הנמצאת במרחב השמות הגלובלי

מדוע שמים את using namespace std?

- בתוך namespace זה יש את כל הפקודות הבסיסיות
- בלעדיו נצטרך להוסיף את הקידומת std:: לכל הפונקציות הבסיסיות שבהן נשתמש, אחרת נקבל למשל את השגיאה:

error C2065: 'cout' : undeclared identifier

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

void main()
{
    int x;

    cout << "Enter a number: ";
    cin >> x;
{
```

```
#include <iostream>


void main()
{
    int x;

    std::cout << "Enter a number: ";
    std::cin >> x;
{
```

ביחידה זו למדנו:

- הגדרת תכנות מכוון עצמים
- השוני הסינטקטי בין C ל- C++:
  - printf cout
  - scanf cin
  - gets cin.getline
  - malloc new
  - free delete
- טיפוס התייחסות
- מרחבי שמות (namespace)

תרגול

Microsoft Office
d 97 - 2003 Docun