

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`
 - `nullptr`
- טיפוס התייחסות
- העמסת פונקציות
- ערכי ברירת מחדל לפונקציות

מה אנחנו יודעים? תכנות פרוצדורלי

- בקורס C למדנו לתכנת במתודולוגיה הנקראת תכנות פרוצדורלי
 - התכנות היה בסגנון Top-Down
 - הדגש בתוכניות היה על פונקציות והמידע המועבר ביניהן
- למשל, 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

לתכנות מכוון עצמים 3 עקרונות מרכזיים

כל הנתונים והפעולות הקשורות לישות מסוימת מרוכזות יחדיו.
המשתמש עובד עם "קופסא שחורה"
יתרונות: קל ללמוד את הקוד ולהתמצא בו, תחזוקה פשוטה

הכמסה
(encapsulation)

הרחבה של ישות קיימת כדי למנוע שכפול קוד, או לחילופין כדי לתת מימוש אלטרנטיבי לקוד קיים.

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

הורשה
(inheritance)

מאפשר להתייחס לישויות שונות בעלי בסיס זהה (מכנה משותף) באותו אופן

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

רב-תצורתיות
(פולימורפיזם,
polymorphism)

2 תכונות נוספות ב- ++C

כלי המאפשר לכתוב קוד כללי לטיפוסים שונים

- דוגמה: 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.

מנגנון לטיפול שגיאות בזמן ריצה

תבניות
(templates)

חריגות
(exceptions)

7 שינויים מ-C ל-C++

- שפת C++ מאוד דומה סינטקטית לשפת C, אך יחד עם זאת יש כמה שינויים:
 - (1) 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
 - (2) ניתן להגדיר משתנים בכל חלק בתוכנית, ולא רק בתחילת בלוק
 - (3) במקום הכללת הספרייה `stdio.h` שהכילה פקודות קלט-פלט, נכליל את הספרייה `iostream` ונוסיף `using namespace std;` (הסבר בהמשך)
 - (4) קיים הטיפוס `bool` שמחזיק את הערכים `true` או `false`
 - (5) קיים הטיפוס `nullptr_t` עם הערך `nullptr` ונשתמש בו במקום ב- `NULL`
 - (6) פקודות שונות לטיפול בקלט ופלט:
 - במקום הפקודה `printf` נשתמש בפקודה `cout`
 - במקום הפקודה `scanf` נשתמש בפקודה `cin`
 - (7) פקודות שונות לטיפול בהקצאות ושחרור זיכרון:
 - במקום הפקודות `malloc/calloc` נשתמש בפקודה `new`
 - במקום הפקודה `free` נשתמש בפקודה `delete`

אבל ראשית, שאני לא אתעצבן..

There are two types of people.

```
if (Condition)
{
    Statements
    /*
     *
     */
}
```

```
if (Condition) {
    Statements
    /*
     *
     */
}
```

Programmers will know.

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>
```

C

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

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

C++

```
int 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>
```

C

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

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

C++

```
int 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

C

```
#include <stdio.h>

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

C++

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

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

הפונקציה תקרא עד 9
תווים (אחד עבור ה- '\0')
או עד ENTER

הפקודה cin.getline ניקוי ה- buffer

```
void cleanBuffer()
{
    int c;
    do
    {
        c = getchar();
    } while (c != EOF && c != '\n');
}
```

```
int main()
{
    int x;
    char str[20];

    cout << "Enter num: ";
    cin >> x;
    cout << "Enter str: ";
    cleanBuffer();
    cin.getline(str, 20);

    cout << "x=" << x << endl;
    cout << "str=" << str << endl;
}
```

```
Enter num: 5
Enter str: x=5
str=
```

```
Enter num: 5
Enter str: hi
x=5
str=hi
```

או קריאה לפונקציה
flushall, שגרסאות חדשות
של VS מתעלמות מקיומה

הפקודה קראה את האנטר שהיה
ב- buffer ולכן נדמה כאילו
התוכנית דילגה על הפעולה

בעיה בגרסאות קומפיילר מתקדמות

- הקומפיילר של visual studio לא תמיד אוהב את הפונקציות שיש ב- string.h ולכן נותן warning (ובחלק מהגרסאות אפילו error) שמספרו 4996, והקומפיילר מציע לכם להשתמש בפונקציה חליפית
 - לא להשתמש בה!
- הפתרון הוא הוספת השורה הבאה כשורה הראשונה בתוכנית:
`#pragma warning (disable: 4996)`
- משמעות שורה זו היא לומר לקומפיילר להתעלם מהערה/שגיאה עם מספר זה

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 (שחרור מערך עם [])

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

C

```
int main()
{
    int *arr, size;

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

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

C++

```
int main()
{
    int *arr, size;

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

C

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

```
int main()
{
    int i, size;
    int* arr;

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

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

    free(arr);
}
```

C++

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

```
int main()
{
    int size;

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

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

    delete []arr;
}
```

נשים לב להגדרת המשתנים
באמצע התוכנית ובלולאה בפרט

דוגמה תוכנית המטפלת במטריצה

```
1. #include <iostream>
2. using namespace std;

3. const int N = 3;
4. const int M = 4;

5. // prototypes
6. int** allocateMatrix (int rows, int cols);
7. void enterInput      (int** mat, int rows, int cols);
8. void printMatrix     (int** mat, int rows, int cols);
9. void freeMatrix      (int** mat, int rows);

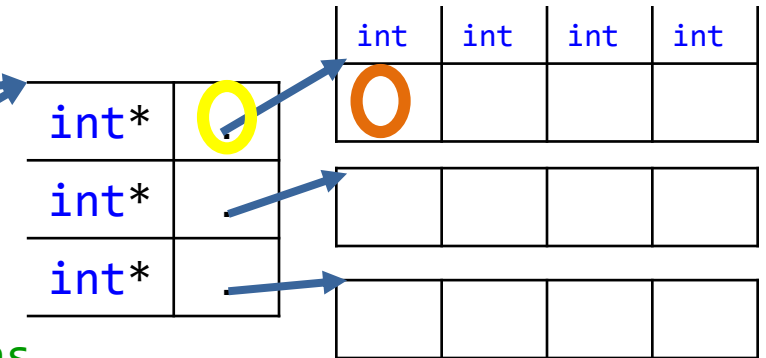
10. int main()
11. {
12.     int** mat;

13.     mat = allocateMatrix(N, M);
14.     enterInput(mat, N, M);
15.     printMatrix(mat, N, M);
16.     freeMatrix(mat, N);
17. }
```

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

דוגמה 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.     return mat;
28. }
```

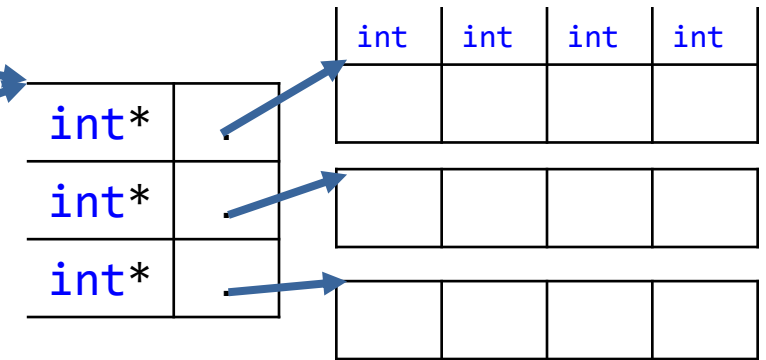


בתחילה מקצים מערך של int*, עבור השורות

```
29. void enterInput(int** mat, int rows, int cols)
30. {
31.     for (int i=0 ; i < rows ; i++)
32.         for (int j=0 ; j < cols ; j++) i=0, j=0
33.             cin >> mat[i][j];
34. }
```


דוגמה תוכנית המטפלת במטריצה (3)

```
37. void printMatrix(int** mat, int3 rows, int4 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. void freeMatrix(int** mat, int3 rows)
47. {
48.     for (int i=0 ; i < rows ; i++)
49.         delete []mat[i];
```

```
37.     delete []mat;
38. }
```

נשים לב: שחרור מערך עם ציון []

nullptr

- החל מגרסת C++11 הוסיפו את הטיפוס `nullptr_t` שערכו הוא `nullptr`
- טיפוס זה בא להחליף את השימוש ב- `NULL` מאחר ו- `NULL` הוא בסה"כ `define` לערך 0
- רצו להבדיל מבחינה תחבירית בין מספר לבין מצביע

```
int main()
{
    int* p = nullptr;
    p = 0;
    int x = NULL;
    //int y = nullptr;

    cout << p << endl;
    cout << x << endl;

    cout << typeid(NULL).name() << endl;
    cout << typeid(nullptr).name() << endl;
}
```

```
00000000
0
int
std::nullptr_t
```

דוגמה שעדיף להשתמש ב- nullptr

```
void foo(int) { cout << "In foo(int)\n"; }  
void foo(int*) { cout << "In foo(int*)\n"; }
```

```
int main()  
{  
    foo(0);  
    foo(NULL);  
    foo(nullptr);  
}
```

In foo(int)
In foo(int)
In foo(int*)

כנראה שהתכוונו
שילך ל- foo(int*)

טיפוס התייחסות (ref)

- 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 ארגומנטים בפונקציות, אך יש דרך חדשה הנקראת העברת פרמטרים by reference

שליחת פרמטר by ref לעומת by val

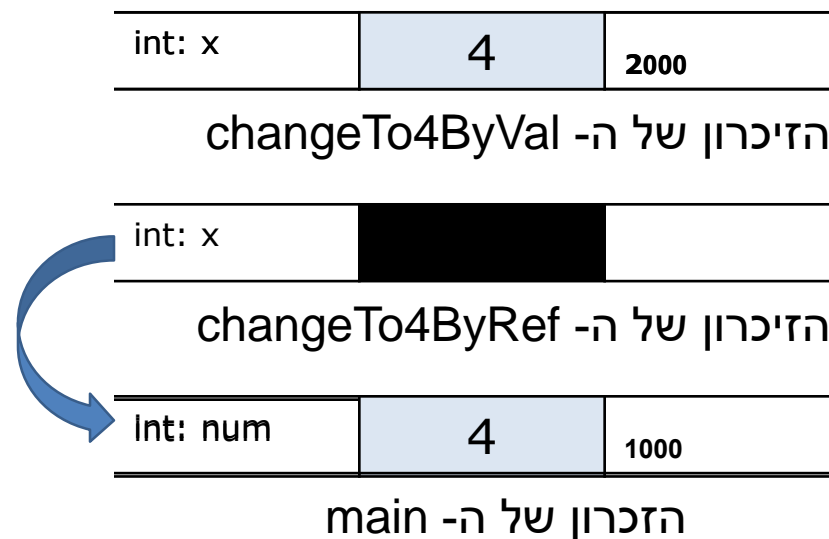
```
void changeTo4ByVal(int x)
{
    x = 4;
}
```

```
void changeTo4ByRef(int& x)
{
    x = 4;
}
```

```
int main()
{
    int num = 10;

    changeTo4ByVal(num);
    changeTo4ByRef(num);
}
```

- בפונקציה המקבלת פרמטר by ref, מציינים לידו & • זהו למעשה מתן שם נוסף לארגומנט המקורי שנשלח הנגיש מתוך הפונקציה
- אין הבדל סינטקטי בשליחת המשתנה בעבור משתנה שעובר by value או by reference

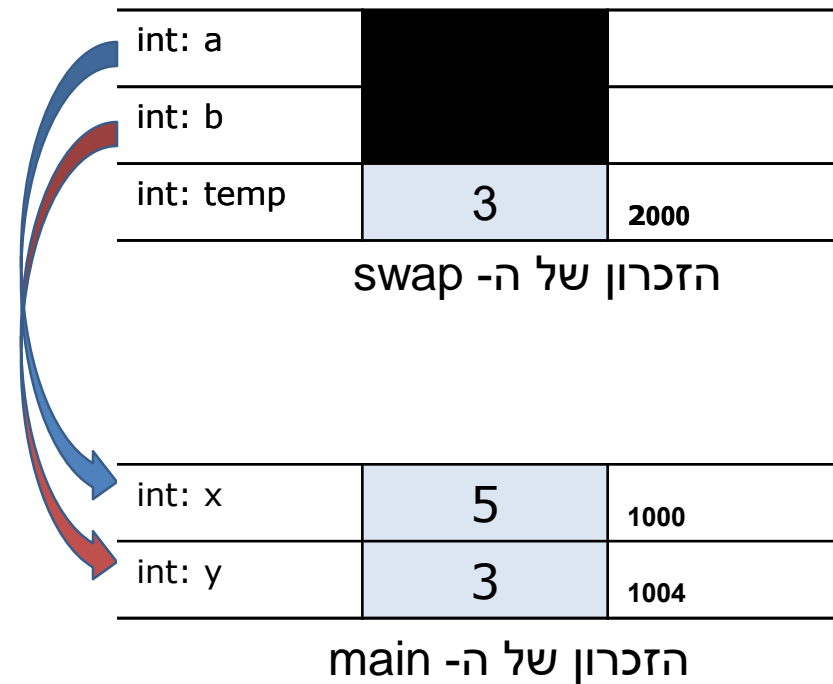


פרמטר ref הדוגמה swap

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

int 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";
}
```



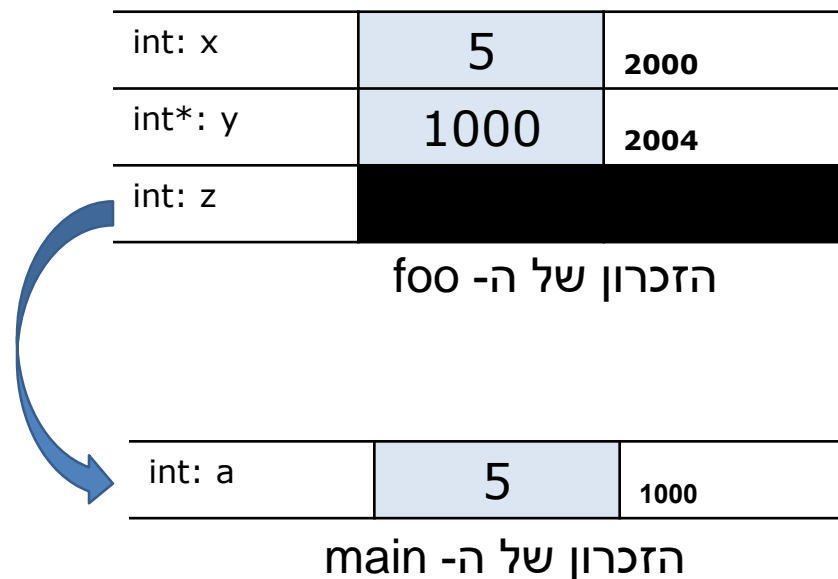
פרמטר ref לעומת פרמטר פוינטר

```
void foo(int x, int* y, int& z)
{
    cout << x << " " << &x << endl;
    cout << y << " " << &y << " " << *y << endl;
    cout << z << " " << &z << endl;
}

int main()
{
    int a = 5;

    cout << a << " " << &a << endl;
    foo(a, &a, a);
}
```

```
5 1000
5 2000
1000 2004 5
5 1000
```



משתנה מטיפוס reference

```
1000 1000
```

```
5 3 5
```

```
6 3 6
```

```
3 3 3
```

```
4 3 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

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

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

| | | |
|-----------|---|------|
| int: x | 4 | 1000 |
| int: y | 3 | 1004 |
| int&: ref | | |

הזכרון של ה-main

טיפוס ref החזרתו 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)

טיפוס ref החזרתו מפונקציה דוגמה

```
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];
}
```

```
int 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 | 2000 |
| int: size | 3 | 2004 |
| int: maxIndex | 1 | 2008 |

הזכרון של ה-getMax

העמסת פונקציות (Functions Overloading)

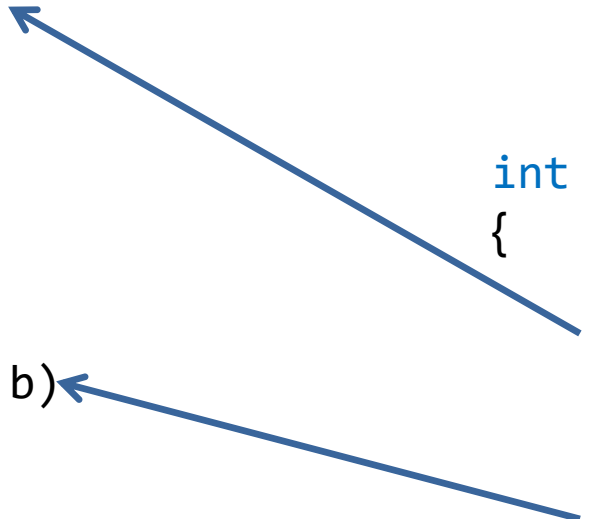
- בשפת ++C ניתן להגדיר כמה פונקציות עם שם זהה, בתנאי שהקומפילר ידע לאיזו גרסה לפנות
- כלומר, ניתן להגדיר פונקציות דומות עם שם זהה, המקבלות טיפוסים שונים
- דוגמה:

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

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

int main()
{
    int n1=3, n2=6;
    swap(n1, n2);

    char ch1='a', ch2='z';
    swap(ch1, ch2);
}
```



בעית דו-משמעות (ambiguity)

- הקומפיילר אינו יכול להבדיל בין 2 פונקציות הנבדלות אך ורק בערך המוחזר
- הסיבה: לא בהכרח יהיה שימוש בערך המוחזר
- דוגמה:

```
int foo() {return 0;}  
char foo() {return 'a';}
```

```
int main()  
{  
    foo();  
}
```

מבחינת הקומפיילר זו הגדרה כפולה של הפונקציה, כי אינו מבחין
כגרסה שוני כאשר ההבדל הוא רק בערך המחזר:
error C2371: 'foo' : redefinition; different basic types

בעיית דו-משמעות (ambiguity) (2)

- גם במקרה זה הקומפיילר לא ידע לאיזו גרסה לפנות:

```
void foo(double d) {}  
void foo(float f) {}  
  
int main()  
{  
    int x=3;  
    foo(x);  
}
```

error C2668: 'foo' : ambiguous
call to overloaded function
הקומפיילר לא יודע אם להמיר את x
ל- double או ל- float

דוגמא תקינה

```
void foo(double d) {}  
void foo(float f) {}  
  
int main()  
{  
    double d = 5.2;  
    float f = 7.3f;  
    foo(d);  
    foo(f);  
}
```

בעיית ה- ambiguity אינה בעיה של הפונקציות, אלא של אופן
הקריאה, שאינו מורה לקומפיילר באופן חד-משמעי לאיזו גרסה לפנות

ערכי ברירת מחדל (default values)

- בשפת ++C ניתן לתת ערך ברירת מחדל לפרמטרים האחרונים שהפונקציה מקבלת, ובכך יהיה ניתן בשימוש להחסיר ארגומנטים אלו
- התוצאה היא העמסת הפונקציה

```
void printChar(char ch='*', int times=5);
```

```
int main()
```

```
{
```

```
    printChar('#', 3);
```

```
    printChar('@');
```

```
    printChar();
```

```
}
```

במקרה של הפרדה בין ההצהרה למימוש,
ערכי ברירת המחדל יכתבו בהצהרה

העמסת 3 פונקציות
במחיר מימוש של אחת ☺

```
###  
@@@@@  
*****
```

```
void printChar(char ch, int times)
```

```
{
```

```
    for (int i=0 ; i < times ; i++)
```

```
        cout << ch;
```

```
    cout << endl;
```

```
}
```

ערכי ברירת מחדל (2)

1. `void printChar(char ch='*', int times=5);`

2. `void printChar(int times, char ch='*');`

3.

4. `int main()`

5. {

6. `printChar('#', 3);`

7. `printChar('@');`

8. `printChar();`

9. `printChar(8);`

10. }

11.

12. `void printChar(char ch, int times)`

13. {

14. `for (int i=0 ; i < times`

15. `cout << ch;`

16. `cout << endl;`

17. }

18. `void printChar(int times, char ch)`

19. {

20. `printChar(ch, times);`

21. }

נשים לב כי לא ניתן לתת ערך ב"מ לפרמטר `times`, משום שנקבל שגיאת `ambiguity` במידה ויקראו לפונקציה ללא פרמטרים

קריאה לפונקציה הראשונה

קריאה לפונקציה השנייה

```
###
@@@@@
*****
*****
```

לא ניתן היה להגדיר את את הפונקציה הבאה:
`void printChar(int times=5, char ch);`
משום שערכי ב"מ ניתן לתת רק לפרמטרים האחרונים ברשימה. אחרת, הקריאה לפונקציה תצטרך להיות: `printChar(, '*')` וזה לא תקין

ביחידה זו למדנו:

- הגדרת תכנות מכוון עצמים
- השוני הסינטקטי בין C ל- C++:
 - `printf` → `cout`
 - `scanf` → `cin`
 - `gets` → `cin.getline`
 - `malloc` → `new`
 - `free` → `delete`
 - `nullptr`
- טיפוס התייחסות
- העמסת פונקציות
- ערכי ברירת מחדל לפונקציות

תרגיל – טיפוס התייחסות – מה הפלט?

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

int globalVar = 8;

int& getGlobalVarByRef()
{
    return globalVar;
}

// this is wrong !!!
// can't return local variable ByRef !
// Should be compilation error, but the VS
// takes it and gives a warning
// (anyway it's WRONG)
int& getLocalVar()
{
    int i = 10;
    return i;
}

// here i is a reference to globalVar
// and this is fine
int& getVarByRef()
{
    int& i = globalVar;
    return i;
}
```

```
const int& getConstValByRef()
{
    // warning
    return 5; // temp var is created
}

int main()
{
    getGlobalVarByRef() = 3;
    cout << globalVar << endl;

    getLocalVar() = 4;
    cout << getLocalVar() << endl;

    getVarByRef() = 5;
    cout << getVarByRef() << endl;
    cout << globalVar << endl;

    //getConstValByRef() = 6;
    // because const can not be changed....
    cout << getConstValByRef() << endl;
}
```

תרגיל – החזרת טיפוס התייחסות

- כתוב את הפונקציה הבאה אשר מקבלת מערך, גודלו וערך לחיפוש המתקבל by ref:

```
int& find(int arr[], int size, int& seek)
```

- הפונקציה תחזיר by ref את האיבר הראשון במערך שערכו שווה ל- seek, במידה ואינו קיים תחזיר את seek.

- שאלה: מדוע הפרמטר seek צריך לעבור by ref?

- כתבו main:

- הגדירו מערך

- שלחו ערכיו לפונקציה הנ"ל עם ערך לחיפוש. במידה והערך נמצא במערך, יש להחליף ערכו ב-1.

- הדפיסו את המערך לאחר הקריאה לפונקציה

תרגיל – חזרה על הקצאות דינאמיות (1/4)

1. כתבו את המבנה Survivor:

- נתוני המבנה:

- א. שם השורד: מחרוזת סטטית בגודל 20

- ב. גיל

- כתבו את הפונקציות הגלובליות הבאות, בחלק מהמקרים עליכם להחליט מהם הפרמטרים שיתקבלו:

- א. אתחול: הפונקציה תקבל ref ל-Survivor ותקרא אליו נתונים מהמקלדת

- ב. פונקציה המדפיסה את נתוני השורד

תרגול – חזרה על הקצאות דינאמיות (2/4)

2. כתבו main:

- I. שאלו את המשתמש כמה שורדים יש בכל שבט
- II. הגדירו 2 מערכים של מצביעים ל-Survivor בגודל המבוקש
- III. קלטו נתונים בלולאה לתוך 2 השבטים עד אשר המשתמש יבחר להפסיק או עד אשר לא יהיה מקום לשורדים נוספים
- IV. הדפיסו את נתוני 2 השבטים
- V. שחררו זיכרון

תרגול – חזרה על הקצאות דינאמיות (3/4)

3. כתוב את המבנה Tribe:

• נתוני המבנה:

- מספר השורדים המקסימלית בשבט
- מערך של מצביעים ל-Survivor
- מספר השורדים שנשארו בשבט

4. כתבו את הפונקציות הגלובליות הבאות, בחלק מהמקרים עליכם להחליט מהם הפרמטרים שיתקבלו:

- אתחול: הפונקציה תקבל ref ל-Tribe ותאתחל את נתוניו (ללא שורדים עדיין, רק איתחול)
- הוספת שורד לשבט ועדכון הנתונים הרלוונטיים
- הדפסה שמות כל השורדים בשבט

תרגול – חזרה על הקצאות דינאמיות (4/4)

5. עדכנו את ה-main כך שישתמש עבור כל שבט במבנה Tribe שכתבתם במקום שני מערכים של Survivor
- אל תשכחו לשחרר זיכרון!