

הספרייה

- הנתונים שנרצה לשמור על כל ספר:



- שם
- מחבר
- מספר ISBN
- שפה
- מספר עמודים
- הוצאה
- תאריך פרסום
- ...

1

2

קורס תכנות

שיעור אחד עשר: מבנים

נתונים של ספר

```
char title[TITLE_LEN],
      author[AUTHOR_LEN],
      isbn[ISBN_LEN],
      language[LANG_LEN],
      publisher[PUBLISHER_LEN];
int pages, year, month, day;
```

איפה הספר?

3

פונקציה שמדפיסה פרטים של ספר

```
void print(const char *title,
          const char *author,
          const char *isbn,
          const char *language,
          const char *publisher,
          int pages,
          int year,
          int month,
          int day)
```

```
{
  ...
}
```

איפה הספר?

4

מה היינו רוצים?

- לקבץ הגדרות של משתנים שונים תחת שם יחיד
- בדומה לפונקציות שמקבצות פעולות תחת שם

- שימוש בטיפוס החדש כאילו היה טיפוס בסיסי
- הגדרת משתנים, איתחול, פעולות על הטיפוס וכדומה

```
void print(const book *b)
{
  ...
}
```

5

מבנים - Structures

- טיפוס חדש

- "חבילה" של משתנה אחד או יותר תחת שם יחיד

- שונה ממערך, יכול להכיל משתנים מטיפוסי שונים
- טיפוסים בסיסיים, מערכים, מצביעים ואפילו מבנים אחרים

6

הצהרה (בעזרת דוגמא)

שם הטיפוס (רשות) מאפשר לנו להתייחס לטיפוס בהמשך

המילה השמורה struct

```
struct book
{
    char title[TITLE_LEN],
      author[AUTHOR_LEN],
      isbn[ISBN_LEN],
      language[LANG_LEN],
      publisher[PUBLISHER_LEN];
    int pages;
    Date publication_date;
};
```

המשתנים (שדות) הטיפוס החדש המרכיבים את הטיפוס

?Date mm

7

הצהרה - כללי

שם הטיפוס (רשות) מאפשר לנו להתייחס לטיפוס בהמשך

המילה השמורה struct

```
struct <Tag>
{
    <field-type> <field-name>;
    <field-type> <field-name>;
    ...
    <field-type> <field-name>;
};
```

המשתנים (שדות) הטיפוס החדש המרכיבים את הטיפוס

8

הצהרה

- תופיע בתחילת הקובץ
- לאחר הגדרות ה define
- מגדירה את התבנית הכללית של הטיפוס
- שימו לב, זו הצהרה על טיפוס ולא על משתנה!

9

נקודה במישור

- זוג סדור, (x, y) , של מספרים המייצגים את מיקום הנקודה במישור

- נרצה למפות את הטיפוס המורכב לטיפוסים פשוטים יותר

```
struct point
{
    double x, y;
};
```

- מכיל שני שדות x, y מהטיפוס הבסיסי double המייצגים קואורדינטאות במישור

10

עבודה עם מבנים

- המבנה הוא טיפוס
- כמו char^* , double, int

- נגדיר משתנים מהטיפוס החדש

שם הטיפוס

```
int main()
{
    struct point p;
    int i;
    double d;
};
```

שם המשתנה

11

כתיבה מקוצרת - typedef

- ניתן לתת שם נרדף לטיפוסים

- לדוגמא: קרא לטיפוס char^* גם בשם string

```
typedef char* string;
```

- ניתן להשתמש בשם הנרדף בכל מקום שבו השתמשנו בשם המקורי

```
string strlen(const string str) {...}
```

```
typedef <name> <alias>
```

- באופן כללי:

```
typedef struct book Book;
typedef struct point Point;
```

- גם למבנים

12

קצר יותר

- ניתן לחבר את הגדרת המבנה עם השם הנרדף

```
typedef struct
{
    double x, y;
} point;
```

לא חייבים את השם

- וכאשר נשתמש

```
double distance(point p, point q)
{
    ...
}

int main() {
    point p1, p2;
    ...
}
```

13

משתנים מטיפוס מורכב

אם משתמים ב typedef

- משתנים ככל המשתנים האחרים (כמעט)

```
point p, q;
point p = {1, 0.2};
q = p;
point *p_ptr;

double distance(point p, point q) {
    ...
}
```

קריאה לפונקציות / ערך מוחזר

14

איתחול מבנים

- ניתן לאתחל משתנה בשורת ההגדרה על ידי:

```
int main()
{
    point p1 = {1.5, 2};
    ...
    return 0;
}
```

- האיתחול מתבצע לפי סדר השדות בהגדרה
- איתחול חלקי יגרום לשאר השדות להיות מאותחלים ל-0

15

איתחול מבנים (המשך)

```
book b = {"Alice in Wonderland", // title
         "Lewis Carroll",       // author
         "193659420X",          // isbn
         "English",             // language
         "Tribeca Books",       // publisher
         100,                   // pages
         {2010, 12, 12}};      // date
```

- שימו לב לאיתחול של השדה `publication_date` שהוא בעצמו מהטיפוס המורכב `Date`

16

פעולות על מבנים

- הפעולות היחידות המוגדרות על מבנה הן:

- גישה לשדה** – האופרטור `.` (dot)
- כתובת המשתנה** – אופרטור `&` (כמו עבור כל משתנה אחר)
- מחזיר את הכתובת של השדה הראשון במבנה
- השמה** – אופרטור `=`
- ההשמה מתבצעת ע"י העתקה שדה-שדה ממשתנה אחד לשני

- לא מוגדרות: השוואה, פעולות אריתמטיות ולוגיות
- נדרש להגדיר פונקציות

17

גישה לשדות של מבנה

- מתבצעת בעזרת האופרטור `.` (dot)
- מופעל על משתנה ולא על הטיפוס

- גישה לשדה `x` במשתנה `p` מטיפוס `point`

```
p.x
```

- באופן כללי

```
<variable-name>.<field-name>
```

18

גישה לשדות של מבנה

- מתבצעת בעזרת האופרטור . (dot)
- מופעל על משתנה ולא על הטיפוס

```
int main()
{
    point p1, p2;
    scanf("%lf%lf", &p1.x, &p1.y);
    p2.x = 2 * p1.y;
    p2.y = 4 * p1.x;
    printf("p2 = <%g, %g>\n", p2.x, p2.y);
    return 0;
}
```

שני משתנים מטיפוס point

קלט של ערכי השדות x ו-y עבור המשתנה p1

חישוב ערכי השדות x ו-y של המשתנה p2 בעזרת ערכי השדות x ו-y של המשתנה p1

פלט של ערכי השדות x ו-y של המשתנה p2

19

כתובת של מבנה

- בדומה לטיפוסים אחרים
- נשתמש באופרטור & כדי לקבל את הכתובת של המשתנה

```
point p;
point *p_ptr = &p;
```

הכתובת של p

מצביע ל-point בשם p_ptr

20

השמה

- ניתן לבצע השמה בין שני משתנים מאותו טיפוס מורכב

```
point p1, p2;
...
p1 = p2;
```

העתקה של הערך במשתנה p1 למשתנה p2

- השמה מבוצעת עבור כל השדות
- שקול ל:

```
point p1, p2;
...
p1.x = p2.x;
p1.y = p2.y;
```

21

פעולות אחרות

- לא מוגדרות פעולות אחרות על מבנים
- תלויות בטיפוס הספציפי
- מה המשמעות של שוויון בין נקודות? ספרים?
- נצטרך לממש אותן בעצמנו
- בעזרת פונקציות

22

השוואה

- האופרטור == לא מוגדר עבור מבנים
- נממש שוויון עבור נקודות בעזרת פונקציה
- נקודות שוות עם ערכי ה-x וה-y שלהם שווים

```
int equals(point p1, point p2)
{
    return p1.x == p2.x && p1.y == p2.y;
}
```

האופרטור == מוגדר על הטיפוס ה-bסיסי double

23

כיצד נחשב מרחק בין שתי נקודות?

- מרחק בין שתי נקודות p_1 ו- p_2 מוגדר על ידי
- $$p_1 = \{x_1, y_1\}$$
- $$p_2 = \{x_2, y_2\}$$
- $$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
- הפונקציה distance

```
double distance(point p1, point p2)
{
    return sqrt( (p1.x - p2.x) * (p1.x - p2.x) +
                (p1.y - p2.y) * (p1.y - p2.y) );
}
```

24

כיצד נחשב מרחק בין שתי נקודות?

```
int main()
{
    point p, q;

    printf("Enter x and y coord. of the first point\n");
    scanf("%lf%lf", &p.x, &p.y);

    printf("Enter x and y coord. of the second point\n");
    scanf("%lf%lf", &q.x, &q.y);

    printf("The distance is %g\n", distance(p, q));

    return 0;
}
```

25

מבנים ופונקציות

- בדיוק כמו משתנים רגילים
- call by value
- העתקת הערך של המשתנה למשתנה המקומי בפונקציה
- בשביל זה צריך השמה
- שינוי המשתנה בתוך הפונקציה לא משפיע על המשתנה בפונקציה הקוראת
- ניתן גם להחזיר משתנה

26

מצביעים למבנים

- נוכל להגדיר מצביע למבנה בעזרת *

```
point *p_ptr;
```

- כפי שראינו האופרטור & פועל גם על מבנים

היכן נשתמש

- בכל מקום שבו נרצה שהפונקציה תשנה את המבנה המקורי
- העברת מבנה גדול כפרמטר לפונקציה. העברת הכתובת (גודל קבוע) חוסכת שיפול המבנה כולו

27

דוגמה

```
double distance(const point* p, const point* q)
{
    ...
}

int main()
{
    point p, q;

    printf("Enter x and y coord. of the first point\n");
    scanf("%lf%lf", &p.x, &p.y);

    printf("Enter x and y coord. of the second point\n");
    scanf("%lf%lf", &q.x, &q.y);

    printf("The distance is %g\n", distance(&p, &q));

    return 0;
}
```

הפונקציה מקבלת מצביעים לשני משתנים מטיפוס point

נעבר לפונקציה את הכתובת של המשתנים p ו-q

28

מבנים ומצביעים – גישה לשדות

- נשתמש באופרטור *
- ל . קדימות על * ולכן צריך סוגריים
- שימוש ב * על מצביע למבנה יחזיר את המבנה, כדי לגשת לשדה נשתמש כעת באופרטור dot

```
(<struct-pointer>).<field-name>
```

מבנה

```
double distance(const point* p1, const point* p2)
{
    return sqrt( ((*p1).x - (*p2).x) * (*p1).x - (*p2).x) +
                ((*p1).y - (*p2).y) * ((*p1).y - (*p2).y) );
}
```

29

אופרטור החץ -> (סוכר תחבירי)

- במקום להשתמש בצירוף שדה. (מצביע*) כדי לגשת לשדה נוכל להשתמש באופרטור ->
- שינוי בתחביר בלבד
- זהו התחביר המועדף

```
int distance(const point* p1, const point* p2)
{
    return sqrt( (p1->x - p2->x) * (p1->x - p2->x) +
                (p1->y - p2->y) * (p1->y - p2->y) );
}
```

30

מערכים של מבנים

- כמו מערכים של טיפוסים בסיסיים
- הגדרת מערך
- פונקציה המאתחלת את המערך

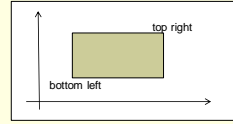
```
point points[10];
```

```
void init_points(point points[], int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        points[i].x = i;
        points[i].y = i;
    }
}
```

31

מלבן

- כיצד נגדיר מלבן שצלעותיו מקבילות לצירים?



- נשתמש במבנה הנקודה שכבר הגדרנו

```
typedef struct
{
    point bottom_left, top_right;
} rectangle;
```

המבנה rectangle מכיל שני שדות מטיפוס point

32

מבנה כשדה של מבנה אחר - איתחול

- בשורת ההגדרה

```
int main()
{
    rectangle rec = { {5, 6}, {7, 8} };
}
```

bottom_left

top_right

33

מערך כשדה של מבנה

- מבנים יכולים להכיל מערך כשדה
- מימוש חלופי של המבנה point

```
typedef struct
{
    double coordinates[2];
} point;
```

- העברת מבנה המכיל מערך לפונקציה תגרום להעתקת כל אברי המערך
- שינוי המערך בתוך הפונקציה לא יגרום לשינוי המשתנה המקורי

34

דוגמא

```
void init_point(point p)
{
    p.coordinates[0] = 1;
    p.coordinates[1] = 1;
}

int main()
{
    point p = { 0, 0 };
    init_point(p);
    printf("[%g, %g]", p.coordinates[0], p.coordinates[1]);
    return 0;
}
```

- הפלט יהיה [0,0]

35

דוגמא (המשך)

```
void init_point(point p)
{
    p.coordinates[0] = 1;
    p.coordinates[1] = 1;
}
```

- הפונקציה לא תשפיע על המשתנה המועבר אליה
- הפונקציה פועלת על העתק של המשתנה הכולל העתק של המערך
- כיצד נשנה את המערך המקורי?
- נעביר מצביע ל-point

36

תרגיל

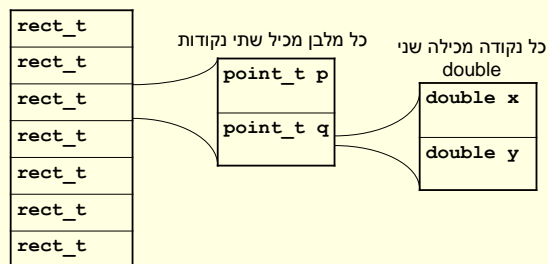
- נממש פונקציה אשר בהינתן רשימה של מלבנים מוצאת את המלבן בעל האלכסון הגדול ביותר

- משתנים:
 - רשימה – מערך של מלבנים (וגודלי)
- ערך מוחזר:
 - כתובת המלבן המבוקש

37

מערך של מלבנים בזיכרון

המלבנים נשמרים ברצף



38

מימוש – max_rect

```
rectangle* max_rect(const rectangle rectangles[], int size)
{
    double max_diagonal = 0, length;
    rectangle_t *result = NULL;
    int i;
    for (i = 0; i < size; i++)
    {
        length = diagonal(&rectangles[i]);
        if (length > max_diagonal)
        {
            max_diagonal = length;
            result = &rectangles[i];
        }
    }
    return result;
}
```

עבור כל מלבן

נחשב את אורך האלכסון

נזכור את האלכסון המקסימאלי שראינו עד עכשיו ואת המלבן שזוהו האלכסון שלו

39

חישוב אורך האלכסון

- אורך האלכסון שווה למרחק בין הנקודות המגדירות את המלבן

```
double diagonal(const rectangle* rec)
{
    return distance(&rec->bottom_left, &rec->top_right);
}
```

נשתמש בפונקציה distance שראינו קודם

40

מבנים - סיכום

- הגדרת טיפוסים משתנים חדשים שקרובים לעולם-הבעיה

- שימוש כמו בטיפוסים רגילים
 - מערך של מבנים, מצביעים על מבנה, העברה לפונקציה וכו'

- מוגדרת פעולת ההשמה
 - יש להגדיר פונקציות עבור פעולות אחרות (השוואה, אריתמטיות ועוד)

41