# מבנה

באנגלית struct :הוא טיפוס חדש המוגדר על ידי המתכנת. בשונה ממערך ,מבנה יכול להכיל טיפוסים שונים של משתנים, והגישה אליהם אינה מספרית אלא שמית.

## הצורך במבנים

נחשוב על תוכנית לניהול מלאי של חנות. בחנות יש סוגי פריטים שונים, שלכל אחד מהם מספר קטלוגי משלו, שם, מחיר, וכמות (כלומר, כמה יש בחנות כעת). אנו רוצים ליצור בסיס נתונים לטיפול במידע הזה, למשל לעדכון מחיר של פריט, לעדכון כמות פריט במקרה של מכירה, וכדומה. בפרט, היינו רוצים לשמור מערך שכל אחד מאיבריו יתאר פריט.

שפת C אינה כוללת טיפוס מתאים לתיאור פריט במלאי החנות - כל אחד מטיפוסי השפה יכול לתאר דברים פשוטים מאד בלבד (כמו מספרים). בפרק זה נראה כיצד להגדיר טיפוסים חדשים ולהשתמש בהם.

## מהו מבנה?

מבנה הוא טיפוס חדש, שאנו מגדירים את שמו ואת מרכיביו כרצוננו. למבנה יש *שדות* ,שהם איברים בעלי טיפוסים ושמות שאנו בוחרים.

לדוגמה, בדוגמה שראינו בצורך במבנים ,נרצה ליצור טיפוס חדש שייקרא item מבנה זה יכיל את השדות הבאים:

- מספר קטלוגי, מסוג שלם
- שם, מסוג מחרוזת בעלת 20 תווים לכל היותר
- מחיר, מסוג נקודה צפה
- כמות (כמה יש במלאי), מסוג שלם לא-שלילי

## הגדרת מבנה

```
struct <name>
{
  [fields]
};
```

כאשר name הוא שם המבנה fields היא רשימה של שדות. כל שדה הוא הצהרה על משתנה שהוא איבר של המבנה.

לדוגמה, בדוגמה שראינו בצורך במבנים ,נגדיר את המבנה כך:

```
struct item
{
  int catalog_number;
  char name[20];
  float price;
  unsigned int count ;
};
```

אין חשיבות מיוחדת לסדר השדות בתוך המבנה - נוכל לקבוע אותו כרצוננו.

הגדרת המבנה הנ"ל מודיעה למהדר שיש כעת טיפוס חדש, ששמו struct item בדיוק כפי שישנם int, char, ו float, לדוגמה, כך גם יש כעת טיפוס בשם struct item

# גישה לשדות המבנה

## גישה לשדות משתנה

ניגשים לשדות משתנה בצורה:

```
<name>.<field_name>
```

כאשר name הוא שם המשתנה - field_name הוא שם השדה.

לדוגמה, נניח ש shoko - הוא משתנה מסוג struct item כדי לקבוע את מחירו ל12.90, נכתוב:

```
shoko.price = 12.90;
```

כדי לקבוע את שמו כ "shoko", - נכתוב:

```
strcpy(shoko.name, "shoko");
```

כדי להדפיס את שמו ואת מחירו, נכתוב:

```
printf("The price of %s is %f", shoko.name, shoko.price);
```

## גישה לשדות מצביע

נניח ש p הוא מצביע למבנה. נוכל לגשת לאיבר שלו בצורה:

```
(*p).<field_name>
```

לדוגמה, אם p הוא מצביע ל struct item - אז את שלוש הדוגמאות הקודמות אפשר לכתוב כך:

```
(*p).price = 12.90;
strcpy((*p).name, "shoko");
printf("The price of %s is %f", *(p.name), *(p.price));
```

גישה למבנה על ידי מצביע היא מהפעולות השכיחות בשפת C השפה לכן כוללת את הצורה המקוצרת:

```
p-><field_name>
```

שמשמעותה זהה. נוכל, לכן, לכתוב את שלוש הדוגמאות בצורה קצרה יותר:

```
p->price = 12.90;
strcpy(p->name, "shoko");
printf("The price of %s is %f", p->name, p->price);
```

## גישה לשדות איבר מערך

ניגשים לשדות איבר במערך בדיוק באופן שבו ניגשים לשדות משתנה לדוגמה:

```
/* An array of 30 items. */
struct item items[30];

/* Access the 2nd item. */

items[1].price = 12.90;
strcpy(items[1].name, "shoko");
printf("The price of %s is %f", items[1].name, items[1].price);
```

# דוגמאות ביניים

נראה כעת מספר דוגמאות של פונקציות המקבלות מצביעים למבנים ופועלות עליהן.

הפונקציה הראשונה מקבלת מצביע לפריט ותוספת מחיר, ומעלה את מחיר הפריט בתוספת:

```
void raise_price(struct item *p, float amount)
{
  p->price += amount;
}
```

הפונקציה הבאה שנכתוב מקבלת מצביע לפריט, ומספר הפריטים שרוצים לקנות מתוכו. היא מעדכנת את מספר הפריטים, ומחזירה את המחיר הכולל שיש לשלם על הקניה:

```
float update_num(struct item *p, unsigned int how_many)
{
  how_many = p->num < how_many? p->num : how_many;
  p->num -= how_many;

  return p->price * how_many;
}
```

הפונקציה האחרונה שנכתוב כאן מקבלת מצביע לפריט, ומדפיסה למסך את נתוניו. להלן הצעה להצהרה לה:

```
void print_item(struct item *p);
```

נזכר במה שראינו במצביעים קבועים .הגיוני שפונקציה שמדפיסה פריט - לא תשנה את פרטיו. כדאי לשנות את ההצהרה כדי להדגיש שמדובר במצביע לפריט שאסור לשנותו, כך:

נשנה כעת את ההצהרה, ונכתוב את הפונקציה:

```
void print_item(const struct item *p)
{
  printf("name: %s, catalog number: %d, price: %f, in stock: %d\n", p-
>name, p->catalog_number, p->price, p->num);
}
```

# אתחול מבנה

לעתים, כאשר מייצרים משתנה מסוג מבנה, יש לתת ערך התחלתי לאיבריו. לדוגמה, נניח שמייצרים משתנה מסוג struct item שמתאר שוקו. ייתכן שנרצה לתת ערך התחלתי לאיבריו כך שמספרו הקטלוגי הוא 23, שמו הוא "shoko", מחירו הוא 12.90, ויש 100 יחידות שלו במלאי.

כפי שראינו מקודם בגישה לשדות המבנה ,אפשר לגשת לכל אחד משדות המשתנה. נוכל, לכן, להשתמש בהשמה לכל אחד מאיבריו:

```
struct item shoko;

shoko.catalog_number = 23;
strcpy(shoko.name, "shoko");
shoko.price = 12.90;
shoko.num = 100;
```

נראה בנושא זה דרכים קצרות יותר לעשות כן.

## אתחול פרטני

לצורך אתחול המבנה בלבד, השפה מאפשרת צורה מקוצרת יותר (בדומה מאד למה שראינו באתחול מערך ) מאשר השמה איבר אחר איבר:

```
struct item shoko = {23, "shoko", 12.90, 100};
```

כאן יש לשים לב לסדר האיברים: הוא צריך להיות זהה לסדר שלפיו הוצהרו השדות במבנה.

אפשר גם לאתחל מערך של מבנים. לדוגמה:

```
struct item items[3] = {{23, "shoko", 12.90, 100}, {109, "roll", 5, 100},
{22, "kartiv", 2.3, 100}};
```

## אתחול ממבנה אחר

אפשר לאתחל מבנה ישירות ממבנה אחר.

נתבונן לדוגמה בקוד הבא:

```
struct item shoko = {23, "shoko", 12.90, 100};

struct item temp = shoko;
```

השורה:

```
struct item temp = shoko;
```

שקולה לאתחול:

```
struct item temp = {shoko.catalog_number, shoko.name, shoko.price,
shoko.num};
```

בס"ד

# שימוש ב־typedef

השימוש ב־ typedef יכול לקצר הצהרות על משתני מבנים. היות שכל מבנה שאנו מגדירים הוא טיפוס, אפשר להשתמש ב typedefכדי לתת לו שם נרדף נוח יותר.

## הבעיה

נתבונן שוב בדוגמאות למשתנים מהטיפוס החדש:

```
struct item shoko;
struct item *p;
struct item items[300];
```

שמו של הטיפוס הוא struct item (שתי מילים), ולכן כל אחת מההצהרות ארוכה יחסית. שפת C ידועה בקצרנותה הרבה. אם הקוד מכיל הצהרות רבות כאלה, עלול הדבר להחשב כאריכות יתר.

## פתרון א

נוכל להשתמש ב־ typedef בלי לשנות את הגדרת המבנה שכבר ראינו:

```
struct item
{
  int catalog_number;
  char name[20];
  float price;
  unsigned int num;
};
```

לאחר הגדרה זו, פשוט נרשום:

```
typedef struct item store_item;
```

וכך ייצרנו שם נרדף store_item ל struct item

## פתרון ב

הפתרון השני מתבסס על כך שאנו יכולים להכניס את הגדרת המבנה לתוך הפקודה typedef, כך:

```
typedef struct
{
  int catalog_number;
  char name[20];
  float price;
  unsigned int num;
} store_item;
```

גם כך ייצרנו "שם נרדף store_item, "למבנה שכרגע הגדרנו.

## התוצאה

בין אם בחרנו בדרך א' והן אם בחרנו בדרך ב', קבלנו שם נרדף בעל מילה אחת. נוכל לכתוב כעת הצהרות כאלו:

```
store_item shoko;
store_item *p;
store_item items[300];
```

## דוגמת המשך

נמשיך בדוגמה פשוטה, שתסכם את רוב מה שלמדנו על מבנים.

התוכנית הבאה היא תוכנת ניהול פשוטה מאד לחנות מכולת:

```
#include <stdio.h>


struct item
{
  int catalog_number;
  char name[20];
  float price;
  unsigned int num;
};


float update_num(struct item *p, unsigned int how_many)
{
  how_many = p->num < how_many? p->num : how_many;
  p->num -= how_many;

  return p->price * how_many;
}


void print_item(const struct item *p)
{
  printf("name: %s, catalog number: %d, price: %f, in stock: %d\n", p-
>name, p->catalog_number, p->price, p->num);
}


int main()
{
  struct item items[6] = {
    {23, "shoko", 12.90, 100},
    {109, "roll", 5, 100},
    {22, "kartiv", 2.3, 5},
    {33, "mastik", 1.0, 10},
    {1000, "pita", 5, 1000},
    {2233, "humus", 23, 20},
    };
  char reply;

  do
  {
    unsigned int i;

    printf("The items in the store are:\n");
    for(i = 0; i < 6; ++i)
      print_item(&items[i]);
```

```
    printf("Which item would you like to purchase? ");
    scanf("%ld", &i);

    if(i > 6)
      printf("This is not a valid item!\n");
    else
      update_num(&items[i], 1);

    printf("Please type 'q' to quit, or anything else to continue: ");
    scanf("%c", &reply);
    printf("\n");
  }
  while(reply != 'q');

  return 0;
}
```

ראשית, הנה הגדרת מבנה הפריט ומספר פונקציות העזר שכבר ראינו:

```
struct item
{
  int catalog_number;
  char name[20];
  float price;
  unsigned int num;
};


float update_num(struct item *p, unsigned int how_many)
{
  how_many = p->num < how_many? p->num : how_many;
  p->num -= how_many;

  return p->price * how_many;
}


void print_item(const struct item *p)
{
  printf("name: %s, catalog number: %d, price: %f, in stock: %d\n", p-
>name, p->catalog_number, p->price, p->num);
}
```

כעת לפונקציה main, המנהלת את הפריטים:

```
int main()
{
  struct item items[6] = {
    {23, "shoko", 12.90, 100},
    {109, "roll", 5, 100},
    {22, "kartiv", 2.3, 5},
    {33, "mastik", 1.0, 10},
    {1000, "pita", 5, 1000},
    {2233, "humus", 23, 20},
    };
  char reply;
```

בס"ד

```
  do
  {
    unsigned int i;

    printf("The items in the store are:\n");
    for(i = 0; i < 6; ++i)
      print_item(&items[i]);

    printf("Which item would you like to purchase? ");
    scanf("%ld", &i);

    if(i > 6)
      printf("This is not a valid item!\n");
    else
      update_num(&items[i], 1);

    printf("Please type 'q' to quit, or anything else to continue: ");
    scanf("%c", &reply);
    printf("\n");
  }
  while(reply != 'q');

  return 0;
}
```

ראשית מגדירים את תכולת המלאי, המכיל 6 סוגי פריטים:

```
struct item items[6] = {
    {23, "shoko", 12.90, 100},
    {109, "roll", 5, 100},
    {22, "kartiv", 2.3, 5},
    {33, "mastik", 1.0, 10},
    {1000, "pita", 5, 1000},
    {2233, "humus", 23, 20},
  };
```

הלולאה:

```
char reply;

do
{
  ..

  printf("Please type 'q' to quit, or anything else to continue: ");
  scanf("%c", &reply);
  printf("\n");
}
while(reply != 'q');
```

פועלת כל עוד לא הקליד המשתמש.'q'

בתוך הלולאה, ראשית מדפיסים את הפריטים:

```
printf("The items in the store are:\n");
for(i = 0; i < 6; ++i)
```

```
  print_item(&items[i]);
```

לאחר מכן מבקשים מהמשתמשת את הפריט שברצונה לרכוש:

```
printf("Which item would you like to purchase? ");
scanf("%ld", &i);
```

כל שנותר הוא (לבדוק אם הפריט חוקי ו) לטפל בבקשה:

```
if(i > 6)
  printf("This is not a valid item!\n");
else
  update_num(&items[i], 1);
```

# מבנים ומצביעים

## כתובת שדה

אפשר למצוא כתובת שדה בצורה:

```
&<s>.<f>
```

כאשר s הוא שם המשתנה, ו f הוא שם השדה.

לדוגמה:

```
struct foo
{
  short int c;

  int m;
};
struct foo s;

int *p = &s.m;
```

## שדות מצביעים

שדות יכולים להיות מכל טיפוס שהוא, כולל, בין היתר, מצביעים. בקטע הקוד הבא, לדוגמה:

```
struct foo
{
  ...
  char *p;
  ...
};
```

p הוא שדה של struct foo וטיפוסו הוא מצביע לתו. קטע הקוד הבא מראה כיצד להשתמש בו:

```c
struct foo f;

char a = 1;

f.p = &a;

/* This makes the value of a be 1. */
*f.p = 1;
```