תכנות בסיסי בשפת C

תוכנית ראשונה

תוכנית ב - C מורכבת מאוסף פונקציות והגדרות טיפוסים. C איננה שפה object oriented כך שאין בה מושגים של מחלקה ואובייקט.

נתחיל בתוכנית הראשונה המסורתית, זו המדפיסה הודעה יחידה על המסך:

```
#include <stdio.h>

int main () {
   printf("hello lord!\n");
   return 0;
}
```

הסבר:
הפקודה הראשונה היא בקשה להכללת קובץ הצהרות. הקובץ הספציפי ששמו stdio.h הוא קובץ המכיל הצהרות על פונקציות קלט\פלט של הספרייה הסטנדרטית (standard input/output). ללא הכללת קובץ זה לא היינו יכולים להשתמש בפקודת ההדפסה שבה השתמשנו בהמשך.
הפונקציה main היא הפונקציה הנקראת ראשונה באופן אוטומטי כאשר מריצים את התוכנית. הערך שהיא מחזירה מועבר למערכת ההפעלה ומציין את האופן בו הסתיימה התוכנית. אם main מחזירה 0, פירושו של דבר שהתוכנית הסתיימה באופן תקין. כל מספר אחר המוחזר מ - main, מציין בעיה כל שהיא שגרמה לתוכנית להסתיים באופן לא תקין.
בדוגמה הנוכחית main לא מקבלת פרמטרים. ניתן להוסיף ל main פרמטרים שייצגו את המחרוזות הנשלחות ממערכת ההפעלה (ניראה בהמשך).
הפקודה printf היא פקודת הדפסה. במקרה זה בחרנו להדפיס מחרוזת פשוטה. הסימון "\n" מייצג ירידת שורה. הפקודה האחרונה גורמת ל main להסתיים ולהעביר למערכת ההפעלה את המסר שהסיום היה תקין.

קומפילציה

נניח שכתבנו את התוכנית הזאת בקובץ הנקרא check.c. בשביל לקמפל את התוכנית, יש לכתוב:

```
$ gcc check.c
```

(במהלך כל הקורס נניח שהסימון $ מייצג פקודת shell של מערכת ההפעלה)
פקודת shell זאת תפעיל את הקומפיילר של C ותיתן לו את הקובץ check.c לתרגום. הקומפיילר יתרגם את התוכנית שלכם לתוכנית שקולה בשפת מכונה שניתן להריץ. לקובץ ההרצה שיצור הקומפיילר יינתן השם a.out.
בשביל להריץ את התוכנית יש לכתוב את שם קובץ ההרצה כפקודת shell :

```
$ a.out
```

הפלט שנקבל:

```
hello lord!
```

אם נרצה ששם קובץ ההרצה יהיה שם אחר, לדוגמה check, נכתוב:

```
$ gcc check.c -o check
```

ואז נריץ אותו כך:

```
$ check
```

ניתן לבקש מהקומפיילר להדפיס הזהרות של "חשד לבעייתיות הקוד" באופן הבא:

```
$ gcc -Wall check.c -o check
```

בקורס זה, על כל הקומפילציות לכלול את ה "-Wall".

הקומפיילר שבו אנו משתמשים הוא קומפיילר שיוצר כחלק מפרוייקט גדול של תוכנה חופשית הנקרא GNU (ראשי תיבות רקורסיביים של : GNU's Not Unix) לקומפיילר זה יש הרבה אפשרויות הרצה ש '-o' ו - '-Wall' , בהן השתמשנו היא רק אחדות. אם תנסו $ man gcc או $ info gcc , תקבלו הסבר מפורט של שאר האפשרויות. בכדי לקבל אינפורמציה על פרוייקט GNU, על משמעות המושג תוכנה חופשית וכלים חופשיים רבים נוספים, אפשר להיכנס לאתר המרכזי של הפרוייקט: www.gnu.org.

משתנים, לולאות, פונקציות ותחביר בסיסי

ניתן דוגמה לתוכנית נוספת, מורכבת מעט יותר:

```
#include <stdio.h>
#include <math.h> /* for pow(_,_) */

int foo1(int n) {
   int i,s=0;
   for(i=1; i<=n; i++)
      s+=i;
   return s;
}

int foo2(int n) {
   return (int)((pow(n,2)+n)/2);
}

void foo3(int d) {
   int a=1,i=2,res1,res2;
   while(i<=d)
      a*=i++;

   res1=a; i=2; a=1;
   do {
      a*=i;
      i++;
   } while (i<=d);

   res2=a; i=2; a=1;
   while(1) {
      a*=i;
      if(i>=d)
         break;
      i++;
   }

   printf("while = %d, do = %d, while&break = %d\n",res1,res2,a);
}

int main () {
   int n;
   printf("Please insert a number:\n");
   scanf("%d",&n);
   printf("foo1(%d)=%d, foo2(%d)=%d\n",n,foo1(n),n,foo2(n));
   foo3(3);
   foo3(1);
   return 0;
}
```

בהרצת התוכנית, אם המספר שנכניס יהיה 3, נקבל את הפלט:

```
foo1(3)=6, foo2(3)=6
```

```
while = 6, do = 6, while&break = 6
while = 1, do = 2, while&break = 2
```

הפקודה #include<math.h> נועדה לאפשר שימוש בפונקציות מתמטיות של הספרייה הסטנדרטית של C. במקרה זה נזקקנו רק לפונקצית החזקה.

כפי שניתן לראות, תחביר הלולאות, המשתנים והאריתמטיקה, הוא התחביר המוכר מ - Java. כאמור, תוכנית ב - C היא אוסף של פונקציות. כאן קיימות שלוש פונקציות רגילות ופונקצית main - ה. בכל תוכנית המיועדת להרצה חייבת להיות פונקצית main אחת ויחידה.

כאשר הקומפיילר רואה קריאה לפונקציה הוא צריך לדעת שהפונקציה אכן קיימת. אם פונקציה קוראת לפונקציה הנמצאת לפניה בקוד לא תתעורר בעיה אך אם נרצה להפעיל פונקציה הנמצאת המקום מאוחר יותר בקוד, תיווצר בעיית קומפילציה. לדוגמה:

```
void foo1(int n) {
   foo2(5); /* compilation error */
}

void foo2(int n) {
   foo1(5); /* no problem */
}
```

בכדי לפתור את הבעיה יש להצהיר על הפונקציה המאוחרת, מראש,בצורה הבאה:

```
void foo2(int n); /* this is a declaration that the function exists */

void foo1(int n) {
   foo2(5); /* no problem */
}

void foo2(int n) {
   foo1(5); /* no problem */
}
```

ההצהרה המתבצעת ע"י כתיבת הכותרת של הפונקציה ללא המימוש שלה.

מובן שבדוגמה הזאת יש רקורסיה הדדית אין סופית, אך זוהי בעיית זמן ריצה ולא בעיית קומפילציה.

שימוש בסיסי ב - printf וב- scanf

הפונקציה scanf היא פונקציה המאפשרת קבלת קלט מהמשתמש. בכדי שאפשר יהיה לקרוא לה, חייבת הפקודה #include<stdio.h> להופיע.
הפונקציה scanf מקבלת מחרוזת ומצביע למשתנה המקבל לתוכו את הערך הנקלט מהמשתמש. על משמעות המצביע נעמוד בהמשך. המחרוזת מתארת את סוג האינפורמציה שאנו מצפים לקבל.
כאשר כתבנו:

```
scanf("%d",&n);
```

המשמעות הייתה שקראנו קלט של מספר שלם בייצוג דצימלי (ה - "d" הוא ל - decimal) לתוך המשתנה n. את משמעות הסימון &, נבין בהמשך.

בהדפסות, כל פעם שבמחרוזת הופיע "%d" הוא הוחלף בערך של מספר שלם בייצוג דצימלי. התוכנית הפשוטה הבאה, מדגימה שימושים נוספים בפונקציות קלט\פלט אלו:

```
#include <stdio.h>
int main() {
   int n;
   float q;
   double w;
   printf("Please enter an int, a float and a double\n");
   scanf("%d %f %lf",&n,&q,&w);
   printf("ok, I got: n=%d, q=%f, w=%f",n,q,w);
   return 0;
}
```

עבור הקלט:

```
7 2.5 3.004
```

נקבל:

```
ok, I got: n=7, q=2.500000, w=3.004000
```

ישנן עד אפשרויות רבות לשימוש ב scanf ובו- printf. לקריאה נוספת אפשר לפנות ל man:

```
$ man scanf
$ man 3 printf
```

(ה "3" נחוץ מכיוון ש printf הוא גם שם לפקודת shell. אם היינו משמיטים את ה - 3 היינו מקבלים הסבר על printf כפקודת shell).

מקור נוסף לאינפורמציה: K&R עמודים 153, 154, 155, 157, 158 ו- 159.

טיפוסים פרימיטיביים

ב C קיימים מספר טיפוסים פרימיטיביים, טיפוסי משתנים המהווים חלק מהשפה ולא הוגדרו ע"י מתכנת. char, int, float ו - double הם דוגמאות לטיפוסים פרימיטיביים. unsinged int הוא דוגמה לטיפוס פרימיטיבי נוסף. ההבדל בין int רגיל ל - unsigned int הוא ש int יכל להכיל ערכים של שלמים שליליים וחיוביים ואילו unsinged int יכל להכיל רק ערכים אי שליליים. טווח הערכים של משתנים מטיפוסים אלה, תלוי בגודל הזיכרון המוקצה להם, גודל הזהה לשניהם.

בכל פלטפורמה, מספר הבתים (bytes) הדרושים ל int עלול להיות שונה. בכדי לדעת כמה בתים דרושים לטיפוס או למשתנה בזיכרון, ניתן להשתמש באופרטור sizeof. לדוגמה:

```
int main() {
   printf("sizeof(char) = %ld, sizeof(int) = %ld, sizeof(float) = %ld, sizeof(double) = %ld\n",sizeof
(char), sizeof(int),sizeof(float),sizeof(double));
   return 0;
}
```

sizeof(char) מוגדר להיות שווה ל - 1 בכל פלטפורמה, אבל הגדלים של הפרימיטיביים האחרים, תלויים בפלטפורמה.

טווח הערכים שיכל לקבל int הוא: [2^(sizeof(int)*8-1)-1) , 2^(sizeof(int)*8-1)-2]. הסיבה לכך היא שמספר הערכים ניתנים לשמירה ב - sizeof(int) בתים בזיכרון היא 2^(sizeof(int)*8) ובחירת מתכנני השפה הייתה להקצות חצי ממספר זה לייצוג ערכים שליליים ואת החצי השני להקצות ל - 0 ולמספרים החיוביים.

טווח הערכים שיכל לקבל unsigned int הוא [0,2^(sizeof(int)*8)-1].

קיימים גם הטיפוסים הפרימיטיביים long int ו - short int (ניתן לכתוב בקיצור long ו - short) המייצגים int הדורש מקום רב יותר בזיכרון ומקום מועט יותר בזיכרון (בהתאמה).

ביטויים בוליאניים

ב - C לא קיים טיפוס עבור ביטויים בוליאניים אך לכל ערך מספרי אפשר להתייחס גם כביטוי בוליאני. המוסכמה היא שהערך אפס נחשב false וכל ערך אחר נחשב true. לדוגמה:

```
int main() {
   int a = 5;
   while(1) {
      if(!(a-3)) {
         printf("** a=3 **\n");
         break;
      }
      printf("a=%d\n",a--);
   }
   return 0;
}
```

פלט:

```
a=5
a=4
** a=3 **
```

מערכים

הקוד הבא מדגים שימוש במערך (חד ממדי) בעל גודל ידוע מראש:

```
int main() {
   int myArray[27],i;
   myArray[0] = 0;
   for(i=1; i<27; i++)
      myArray[i] = myArray[i-1]+i;
   for(i=0; i<27; i++)
      printf("%d |",myArray[i]);
   return 0;
}
```

שימו לב שהתחביר להצהרת מערכים הוא: <type> <identifier>[<size>] ולא <type>[]> <identifier>>> כנהוג ב - Java. כך שלדוגמה :

```
double array[100];
```

היא הצהרה חוקית בשפה, בעוד ש:

```
double[] array;
```

איננה חוקית.

הקוד הבא מדגים מערכים רב ממדיים בעלי ממדים ידועים:

```
int main() {
   float table[10][20],cube[4][7][5];
   table[4][19] = 1.5;
   cube[3][0][4] = 2.5;
   printf("%f %f\n",table[4][19],cube[3][0][4]); // will output: 1.500000 2.500000
   return 0;
}
```

למערכים יש קשר הדוק עם מצביעים ב C. שימוש במערכים בגודל שאיננו ידוע מראש יילמד לאחר הצגת המושגים הבסיסיים של המצביעים. לפני שנוכל להציג את מושג המצביע, נדרש לדעת מספר עובדות בסיסיות על זיכרון המחשב ועל האופן בו C מתייחסת עליו.