שאלה 1 –

כמה תהליכים חדשים (כלומר לא כולל התהליך הראשי ממנו הפונקציה נקראה) נוצרים במהלך הריצה של התוכנית הבאה? **הסבירו תשובתכם.**

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
void main(){
        1.int a = 0;
        2.a += (fork() != 0) ? 2 : 3;
        3.if (a == 2) fork();
        4.a++;
        5.if (a == 3) fork();
}
```

בשורה 2 מתבצע fork ראשון שיוצר את הבן הראשון (לפי הוראות התרגיל ניתן להניח שהוא מצליח), ולכן:

בשביל תהליך האב, התנאי 0 =! (fork() מתקיים כיוון שבפקודת fork האב מקבל מספר גדול מ0 שהוא המזהה של הבן שנוצר, ולכן, עבורו a=2.

לעומת זאת, בשביל תהליך הבן, התנאי 0 =! (fork() לא מתקיים כיוון שבפקודת fork הבן מקבל 0, ולכן, עבורו a=3.

נמשיך מנקודת המבט של **האב** – כרגע a=2. האב נכנס לתנאי שבשורה 3 ומבצע שוב fork שיוצר בן שני. לאחר מכן, בשורה 4 מקבל a=3 ולכן נכנס לתנאי שבשורה 5 ומבצע שוב fork שיוצר בן שלישי. **הבן השני** מקבל גם הוא את a=2 שבשורה 4 הופך לa=3 ולכן נכנס לתנאי שבשורה 5 ומבצע שוב fork שיוצר בן רביעי. **הבן השלישי** ו**הבן הרביעי** מסתיימים מיד אחרי שהם נוצרים.

נעבור לנקודת המבט של **הבן הראשון** - כרגע a=3. הבן לא נכנס לתנאי שבשורה 3. בשורה 4 מקבל a=4 ולכן גם לא נכנס לתנאי שבשורה 5. כלומר, הבן הראשון לא יוצר שום תהליך נוסף.

סה״כ מספר התהליכים החדשים שנוצרים הוא 4.

שאלה 2 –

מה תהיה התוצאה של הרצת קטע הקוד הבא? **הסבירו תשובתכם.**

הניחו שהPID של האבא הוא 168, ואילו של הבנים שלו 768,769,770 וכו׳.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
        1.int x = 150;
        2.printf("PARENT: x is %d\n", x);
        3.printf("PARENT: forking...\n");
        4.pid_t pid = fork();
        5.printf("PARENT: forked...\n");
        6.if (pid == 0){
                7.printf("CHILD: happy birthday\n");
                8.x *= 2;
                9.printf("CHILD: %d\n", x);
        }
```

```
        10.else{
                11.wait(NULL);
                12.printf("PARENT: child completed\n");
                13.x *= 3;
                14.printf("PARENT: %d\n", x);
        }
        15.return EXIT_SUCCESS;
}
```

בשורה 1 -> x=150.

בשורה 2 מודפס "PARENT: x is 150".

בשורה 3 מודפס "PARENT: forking...".

בשורה 4 מתבצע fork ראשון שיוצר את הבן הראשון עם PID 768.

נמשיך מנקודת המבט של **האב** – בשורה 5 מודפס "PARENT: forked...", האב לא נכנס לתנאי שבשורה 6 כיוון שהpid שווה אצלו למזהה של הבן שהוא 768 (כלומר, לא שווה 0), אלא נכנס לelse שבשורה 10.

בשורה 11 מתבצע wait(NULL) שאומר לאב לחכות לכל ילד שיש לו שיסיים לרוץ, ולכן כרגע האב ממתין לילד.

נעבור לנקודת המבט של **הבן הראשון** – בשורה 5 מודפס "PARENT: forked...", הבן נכנס לתנאי שבשורה 6 כיוון שהpid שווה אצלו ל0, בשורה 7 מודפס "CHILD: happy birthday".לאחר מכן, בשורה 8 מבצעים x *= 2 וכיוון שx=150 כעת הוא 300. בשורה 9 מודפס "CHILD: 300". הבן לא נכנס לelse שבשורה 10 ומוחזר EXIT_SUCCESS בשורה 15.

כעת, כשהבן סיים ניתן להמשיך מנקודת המבט של **האב** – בשורה 12 מודפס "PARENT: child completed". בשורה 13 מבצעים x *= 3 וכיוון שבשביל האב x עדיין שווה 150 כעת הוא 450. בשורה 14 מודפס "PARENT: 450" ושוב מוחזר (הפעם בשביל האב) EXIT_SUCCESS בשורה 15.

אז לסיכום, תוצאת הרצת קטע הקוד:

```
PARENT: x is 150
PARENT: forking...
PARENT: forked...
PARENT: forked...
CHILD: happy birthday
CHILD: 300
PARENT: child completed
PARENT: 450
```

!!! נשים לב שכיוון שיש כאן 2 תהליכים שרצים במקביל (האב והבן) וכיוון שהמעבד מבצע אופטימיזציות למיניהן, מרגע היווצרותו של הבן ועד לרגע מותו עלולים להיות שיבושים בהדפסות המסומנות בצהוב (חלקים של האב יודפסו אולי בתוך של הבן או להפך). !!!

<u>שאלה 3 –</u>

מה תהיה התוצאה של הרצת קטע הקוד הבא? **הסבירו תשובתכם.**

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
        1.pid_t child;
```

```
        2.int status;
        3.child = fork();
        4.switch (child) {
                5.case -1:
                        6.perror("fork");
                        7.exit(1);
                8.case 0:
                        9.printf("quitting\n");
                        10._exit(2);
                11.default:
                        12.wait(&status);
                        13.printf("%d\n",WEXITSTATUS(status));
                        14.break;
        }
        15.return 0;
}
```

בשורה 3 מתבצע fork ראשון שיוצר את הבן הראשון.

כיוון שלפי הוראות התרגיל ניתן להניח שהfork מצליח, לעולם child לא יהיה שווה 1- ולכן אף אחד לא יכנס לcase שבשורה 5.

נמשיך מנקודת המבט של **האב** – האב לא יכנס לcase שבשורה 8 כיוון שבפקודת fork האב מקבל מספר גדול מ0 שהוא המזהה של הבן שנוצר, לכן יכנס לdefault שבשורה 11.

בשורה 12 מתבצע wait(&status) שאומר לאב להמתין עד שהמצב (state) של אחד מבניו ישתנה.

נעבור לנקודת המבט של **הבן הראשון** – הבן יכנס לcase שבשורה 8 כיוון שבפקודת fork הבן מקבל 0. בשורה 9 מודפס "quitting " ולאחריו תהליך הבן יסתיים עם סטטוס 2.

כעת, כשהשתנה הסטטוס של הבן (עבר לterminated), ניתן להמשיך מנקודת המבט של **האב** – בשורה 13 מודפס 2 (סטטוס היציאה של הבן - WEXITSTATUS מחזיר את זה). באופן כללי אמורים להשתמש בWEXITSTATUS רק לאחר שמשתמשים בWIFEXITE , אבל שוב, כיוון שאנו מניחים שכל הקריאות לsystem calls מצליחות – אין בעיה עם כך.

לבסוף, מוחזר 0.

אז לסיכום, תוצאת הרצת קטע הקוד :

```
quitting
2
```

<u>שאלה 4 –</u>

מה הם כל הפלטים האפשריים של התוכנית הבאה? **הסבירו תשובתכם.**

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
        1.int value = 3;
        2.if (fork() != 0){
                3.wait(&value);
        }
        4.else{
```

```
            5.exit(value);
        }
        6.value = WEXITSTATUS(value);
        7.value++;
        8.printf("%d", value);
        9.return value;
}
```

בשורה 1 -> value = 3.

בשורה 2 מתבצע fork ראשון שיוצר את הבן הראשון.

נמשיך מנקודת המבט של **האב** – האב יכנס לתנאי שבשורה 2 כיוון שבפקודת fork האב מקבל מספר גדול מ0 שהוא המזהה של הבן שנוצר.

בשורה 3 מתבצע wait(&value) שאומר לאב להמתין עד שהמצב (state) של אחד מבניו ישתנה.

נעבור לנקודת המבט של **הבן הראשון** – הבן לא יכנס לתנאי שבשורה 2 כיוון שבפקודת fork הבן מקבל 0. לכן, הוא יכנס לelse שבשורה 4 ובשורה 5 יסתיים עם סטטוס 3 (כיוון שזה הערך של value).

כעת, כשהשתנה הסטטוס של הבן (עבר לterminated), ניתן להמשיך מנקודת המבט של **האב** – הערך של value משתנה בהתאם לסטטוס היציאה של הבן, אבל כיוון שהוא סיים עם סטטוס 3 value נשאר עדיין 3. האב מדלג על הelse שבשורה 4 כיוון שנכנס כבר לif.

בשורה 6, מתבצעת השמה של WEXITSTATUS(value) לתוך value. WEXITSTATUS מחזיר את סטטוס היציאה של הבן ולכן שוב value מקבל את הערך 3.

באופן כללי אמורים להשתמש בWEXITSTATUS רק לאחר שמשתמשים בWIFEXITE , אבל שוב, כיוון שאנו מניחים שכל הקריאות לsystem calls מצליחות – אין בעיה עם כך.

בשורה 7 מוסיפים 1 לvalue ולכן ערכו כרגע הוא 4.

בשורה 8 מודפס 4 ובשורה 9 מוחזר 4.

סה״כ כל הפלטים האפשריים של התוכנית הם : 4.