

תוכניות דוגמא c_intr1d.c, c_intr2d, c_intr3d.c

התוכניות הללו הן תוכניות C, להמחשה של אספקטים של נושא הפסיקות.

התוכנית c_intr1d.c

התוכנית `c_intr1d.c` היא תוכנית C Turbo Techorah, להמחשה ראשונית של נושא הפסיקות. ההדגשה היא "תוכנית C Turbo" ולא סתם "תוכנית C" מסוימת. התוכנית הזו מתקמפל רק ע"י קומpileר של C עבור DOS (ביחוד C ו-C Microsoft) משומש שהוא מישת רעיון שככל מערכת הפעלה של המחשב האישי שבסאו אחרי DOS אינה נחמצת יורח: שתוכניות אפליקציה משתמשת על DOS הדבר היה אפשרי והרבה תוכניות אפליקציה עשו זאת, ותוכניות האמורלציה (הדמייה) של DOS בדרך כלל משתמשות לתוכן בהדמייה כזו או אחרת של הטכניקה הזו. בגלל זה ניתן להרייך את התוכניות הללו בחלונות ה-command cmd של WINDOWS אם כי ישנן תוכניות דומות (למשל כזו המסתמכת על מקש ה-Print-Screen (Prnt-Scr) שייעבדו אך ורק מתחת DOS האמיתי).

כפי שנאמר, תפקידה של `c_intr1d.c` לשמש הממחשה ראשונית של מושג הפסיקה. שימושו לב לבנה הבא של התוכנית:

```
volatile int Ctrl_Break_Flag;
```

```
....
```

```
void main()
{
```

```
....
```

```
Ctrl_Break_Flag = 0;
```

```
while(Ctrl_Break_Flag == 0)
    ;
```

```
....
```

```
} /* main */
```

על משמעות מילת המפתח volatile נרחיב בהמשך. בשלב זה נתעלם ממנו. על פי הכללים של שפת C, Ctrl_Break_Flag הינו משתנה גלובלי שלם. במהלך הריצה של התוכנית הראשית, אנחנו מציבים לתוכו אפס ומשם נוכנים Ctrl_Break_Flag לולאה while שאינה עורשה דבר למעט לבדוק אם Ctrl_Break_Flag לאפס ואם כן לחזור על הולואה. לכורה לולאת ה-while היא עדין שווה לאפס ואם כן לחזור על הולואה. במקרה לולאת ה-while היא בחזקת לולאה אינסופית ומובוי סתום, שכן היא תלויות בערכו של המשתנה שהוא אין היא משנה תוך כדי ביצוע הולואה. ואכן הרצת התוכנית תתקע את המחשב (במקרה של DOS) או את חלון האמורלה עד שהמשתמש ילחץ על מקש Ctrl-Break.

מה שצריך להיות ברור הוא שטיסום לולאת ה-while כרוך בהפרת העקרון שבו כל פקודת מכונה שבוצע על ידי המחשב נקבע ע"י פקודת המכונה שלפניה. אנחנו רואים לפי הריצה של התוכנית שזה אכן מה שקרה. הדרך שאנו נרשים זאת היא על ידי השתלטת על פסיקת מס' 27 (18h בהקסה), פסיקה שתרחשת בכל רגע שנלחץ המקש Ctrl-Break. בפקודה

```
setvect(27, Ctrl_Break_Handler);
```

קבענו את הרoutine המינוחדת Ctrl_Break_Handler בתור רoutine הטיפול בפסיקה של מס' 27. מרגע זה ואילך הרoutine הזה היא שתבצע עם כל לחיצה של המקש Ctrl-Break ולא זו הרגילה (הגורמת בתנאים מסוימים חוזרת ל-DOS). לאחר واحد הפקודות של הרוטינה הזו היא

```
Ctrl_Break_Flag = 1;
```

כפעם הראשונה שהרוטינה תבצע הערך של המשתנה הגלובלי Ctrl_Break_Flag ישנה ועם החזרה מפסיקה התוכנית אכן משחרר מלולאת ה-while. setvect היא רoutine פנימית של C Turbo המאפשרת השתלטת על פסיקה נתונה, תוך ציון הרוטינה החלופית לפסיקה.

מההდפסות הנראות בפלט

C: Ctrl-Break has been pressed.

C: Terminating ...

צורך להיות ברור שלחיצת Ctrl-Breakacon גרמה הסתעפה לROUTINE Ctrl_Break_Handler ועם סיוםו חזר לתוכנית הראשית ומשם לסיום.

נקודות נוספת שצורך שם לב עליהם:

- שינוי טיפול בפסקה היא פעולה שאינה מתחזרת או מתקבלת עם סיום התוכנית. לכן בכדי שהמערכת תמשיך ל愧ד עם סיום התוכנית הראשית, יש לשמור את הטיפול המקורי בפסקה ולהחזיר אותה לפני סיום. לשם כך קיימת הקריאה **ל-*getvect* בפקודה**

```
Int27Save = getvect(27);
```

getvect היא רoutine פנימית של C המחזיר את כתובות ROUTINE הנורכית שמתפלת בפסקה שמספרה היא מקבלת כפרמטר. היא כמובן אינה משנה אותו. Int27Save הינו משתנה 32 בית בזיכרון שעוד נתיחס אליו מגדרים אותו, אבל בסופו של חשבון הוא פשוט 32 בית המאחסנים את תוכן הכתישה. התוכנית הראשית פשוט משמרת את כתובות המטפל הנוכחי לפני הקריאה ל- **setvect ולפני הסיום משוחרת אותו ע"י הפקודה**

```
setvect(27, Int27Save);
```

- בשביל לכחוב ROUTINE שתתפל בפסקה, היא חייבת להיות מוגדרת כפונקציה void עם רשיית פרמטרים void. לפיכך ההגדלה של Ctrl_Break_Handler

```
void interrupt Ctrl_Break_Handler(void);
```

המילה interrupt היא מילה שמורה של C. משמעות ההגדלה הדו שהROUTINE משמרת את ערכי כל האוגרים ומסימת את ריצתה ע"י פקודת המcona .IRET

- ההגדלה של Int27Save

```
void interrupt (*Int27Save)(void);
```

.void interrupt Int27Save
setvect
מלבד הקצתה שטח מתאים (32 ביט) C יסכים ל�מפל קריאה ל-
ו-getvect רק למשתנים מהסוג הזה.

- המילה volatile המופיעה בהכרזה

```
volatile int Ctrl_Break_Flag;
```

מנחה את הקומpileר שהמשתנה הוזה עשוי לקבל ערכיים באופן לא צפוי ולכון
יש לגשת לזכרוון כל אימת שעריך לעמוד על ערכיו. הדבר נחוץ כי אחרת
הקומpileר עשוי לעשות אופטימיזציה נוספת קריית הערך פעם אחת לתוך ארגר
ולסמן על כך שכיוון שהתוכנית לא שינה את המשתנה, אפשר לסמן על
הערך שנמצא באוגר ולהחסוך בכך גישה לזכרוון. המילה volatile היא מילה
שמורה של שפט C הסטנדרטית והיא רלוונטי לא רק כאן.

- פקודות הקימפל של התוכנית

```
tcc -ml -r- c_intr1d.c
```

מנחה את הקומpileר לקמפל את התוכנית
א. במודל Large ע"י ההנחיה -l-
ב. ללא שימוש במשתני אוגר ע"י ההנחיה -z-.

עבור תוכניות המשתלטות על פסיקות בצורה זו הדבר הכרחי במספר
נסיבות, שלא ניכנס להם כאן.

התוכניות c_intr2d.c, c_intr3d.c

התוכניות הללו הן גירסאות של c_intr1d.c רק שהרטינות getvect ו-
setvect מוחלפים במימושים באסמלוי. הקוד באסמלוי ממומש כאן בשיטה של
inline assembly ועוד כתיבת קוד אסמלוי מתוך תוכנית C.
הדבר נעשה משיקולים של נוחות של תצוגה, השימוש בטכניקה זו היא לא
דווקא קשור לפסיקות. בשיטה זו ממשים קוד באסמלוי בבלוקים מהסוג

```
asm {
    קוד אסמלבי
}
```

או בשורדות קוד מהסוג

פקודת אסמלבי אחתasm

באסמלבי מהסוג זהה יש מגבלות (למשל אין תמייה בפקודות הטעפות מוחנות). ההערות לפי המוסכמויות של שפת C.

התוכנית c_intr2d.c

- בתוכנית c_intr2d.c הפעולות setvect ו-getvect נעשו ע"י גישה ישירה לזכרון לפי כתובת, כאשר התוכנית מזיבה 0 לתוך ה-ES על מנת שתצטבע על השטח שבו נמצאת טבלת ה-IV ובהיסטים קבועים יחסית אליו. למשל הפקודה

MOV AX,ES:[27*4]

הינו קריאה של שדה היחס של הכניסה מספר 27 ב-IV. הכפל הזה נעשה ע"י האסמלר בזמן אסמלி כלומר רק קבועים ניתנים לבצע פעולות אדיתמטיות (לא ניתן לכתוב AX*5 בתוך הסוגרים המרובעים, למשל). באופן שקול ניתן היה לכתוב

MOV AX,ES:[108]

- השימושים של setvect כרכבים ביוטר מפקודת מכונה אחת ולכון מקדים אוTEM הפקודה CLI ולאחריהם פקודת המכונה STI. הדבר נעשה על מנת למניע תקללה כתוצאה מתיקול בבדיקה הפסיקה זו לאחר שהחל השינוי של הכניסה ב-IV ולפני ההשלמה שלו. מבן השאכוי שתקללה צו תתרחש היא אפסית, אך תמיד חייבים להתייחס לזה. למשל במימוש (setvect(27,Ctrl_Break_Handler)

CLI

```
MOV WORD PTR ES:[27*4], OFFSET Ctrl_Break_Handler
MOV WORD PTR ES:[27*4+2], SEG Ctrl_Break_Handler
STI
```

```
asm {
    קוד אסמלוי
}
```

או בשורות קוד מהסוג

פקודת אסמלוי אחתasm

באסמלוי מהסוג זהה יש מגבלות (למשל אין תמייה בפקודות הסתעפות מותניות). ההדרות לפי המוסכמות של שפת C.

התוכנית c_intr2d.c

- בתוכנית c_intr2d.c הפעולות getvect ו-setvect נשות ע"י גישה ישירה לזכרו לפי כתובת, כאשר התוכנית מציבה 0 לתוך ה-ES על מנת שתציביע על השטח שבו נמצא טבלת ה-IV וביחסים קבועים יחסית אליו. למשל הפקודה

MOV AX,ES:[27*4]

הינו קדיחה של שדה היחס של הכניסה מספר 27 ב-IV. הכפל הזה נעשה ע"י האסמלר בזמן אסמלוי כלומר רק קבועים ניתן לבצע פעולות אריתמטיות (לא ניתן לכתוב AX*5 בתחום הסוגרים המרובעים, למשל). באופן שקול ניתן לכתוב

MOV AX,ES:[108]

- המימושים של setvect כרוכים ביותר מפקודת מכונה אחת וכן מקדים אותו הפקודה CLI ולאחריהם פקודת המכונה STI. הדבר נעשה על מנת למנוע תקלת כתזאה מתיפול בבדיקה הפסיקה זו לאחר שהחל השינוי של הכניסה ב-IV ולפניהם ההשלמה שלו. מבון שהסיכוי שתקלה בזו תתרחש היא אפסית, אך תמיד חייבים להתייחס לזה. למשל בימוש setvect(27,Ctrl_Break_Handler)

CLI

```
MOV WORD PTR ES:[27*4], OFFSET Ctrl_Break_Handler  
MOV WORD PTR ES:[27*4+2], SEG Ctrl_Break_Handler  
STI
```

אילו לא ביצענו את ה-CLI ואם יתרחש פסיקה מס' 27 בדיקת לאחר ה-MOV הראשון ולפני הביצוע ה-MOV השני ה-CPU יסתהעף לכתובת של מס' סגמנט לפי רוטינה הטיפול המקורי וαιלו הhispt לפי רוטינה הטיפול החדש. בKİצ'ור לא פה ולא שם. העקרון הכללי הוא פשוט למנוע טיפול בפסקיות בזמן שמבנה הנתונים של ה-IV התחליל לעבור שינויים והשינויים הללו עוד לא הושלמו. מובן שגם היינו שוכחים לכתוב את CLI ו-STI סביר להניח שהשgiaה הזה לא מורגשת או באה ידי ביתוי. אך באופן כללי, חיבבים להקפיד על עקרונות כאלו.

- האופרטור SEG נותנת אפקט דומה לאופרטור OFFSET רק שכאן זה קבוע של מס' סגמנט. הבדל אחר הוא שלא כל כך קל למשר אותו שכן מס' הסגמנט נקבע בזמן ריצה. יש לזה פתרון שלאណון בר כאן.
- המשתנה Int27Save מוגדר כאן כ-int long על מנת להציג את העבודה בכל התוכניות Int27Save הוא בסך הכל משתנה 32 ביט.

התוכנית c_intr3d.c

זוהי תוכנית כמעט זהה ל-c_intr2d.c רק ש-setvect ו-getvect ממומשים ע"י רוטיות DOS:

getvect ממומש ע"י

INT 21h, AH = 35h

"הזר ב-BX:ES את תוכן הכניתה ב-IV שמספרה מועבר אליו ב-AL".

setvect ממומש ע"י

INT 21h, AH = 25h

"הצב לכניתה ב-IV שמספרה מועבר אליו ב-AL את התוכן המועבר אליו ב-DS:DX".

```

/* c_intr1d.c - Change Ctrl-Break interrupt handler.
 Pure C version. */

#include <stdio.h>
#include <dos.h>

volatile int Ctrl_Break_Flag;

void interrupt (*Int27Save) (void); /* Pointer to function */

void interrupt Ctrl_Break_Handler(void)
{
    Ctrl_Break_Flag = 1;
    printf("C: Ctrl-Break has been pressed.\n");
} /* Ctrl_Break_Handler */

void main(void)
{
    Int27Save = getvect(27);           /* Preserve old pointer */
    setvect(27, Ctrl_Break_Handler);   /* Set entry to new handler */
    printf("C: Press Ctrl-Break to continue.\n");

    Ctrl_Break_Flag = 0;

    while (Ctrl_Break_Flag == 0)
        ; /* Do nothing */

    printf("C: Terminating ... \n");

    setvect(27,Int27Save);           /* Restore old pointer */
} /* main */

```

```

E:\users\eytan\asm>tcc -ml -r- c_intr1d.c
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
c_intr1d.c:
Turbo Link Version 5.0 Copyright (c) 1992 Borland International

```

Available memory 4112144

```

E:\>c_intr1d.exe
C: Press Ctrl-Break to continue.
C: Ctrl-Break has been pressed.
C: Terminating ...

```

E:\>

```

/* c_intr2d.c - Change Ctrl-Break interrupt handler.
C + ASM version. */

#include <stdio.h>
#include <dos.h>

volatile int Ctrl_Break_Flag;
long int Int27Save; /* 32bit used to hold old ISR address */

void interrupt Ctrl_Break_Handler(void)
{
    Ctrl_Break_Flag = 1;
    printf("C+InASM 1: Ctrl-Break has been pressed.\n");
}

void main(void)
{
    /* Int27Save = getvect(27);      Preserve old pointer */
    asm {
        PUSH ES          /* Preserve registers */
        PUSH AX
        MOV AX,0          /* Set ES to point to Interrupt vector segment */
        MOV ES,AX
        MOV AX,ES:[27*4]   /* Read offset entry of Ctrl-Break ... */
        MOV WORD PTR Int27Save,AX /* ... interrupt handler */
        MOV AX,ES:[27*4+2]  /* Read segment entry of Ctrl-Break ... */
        MOV WORD PTR Int27Save+2,AX /* ... interrupt handler */

        /* setvect(27,Ctrl_Break_Handler);      Set new interrupt handler */
        CLI             /* Disable interrupts while IV is being changed */
        /* Set offset of new interrupt handler */
        MOV WORD PTR ES:[27*4],OFFSET Ctrl_Break_Handler
        /* Set segment number of new interrupt handler */
        MOV WORD PTR ES:[27*4+2],SEG Ctrl_Break_Handler
        STI             /* Re-enable interrupts (modification finished) */
        POP AX          /* Restore registers */
        POP ES
    }

    printf("C+InASM 1: Press Ctrl-Break to continue.\n");

    Ctrl_Break_Flag = 0;

    while (!Ctrl_Break_Flag)
        ; /* Do nothing */

    /* setvect(27,Int27Save);      Restore old pointer */
    asm {
        PUSH ES          /* Preserve registers */
        PUSH AX
        MOV AX,0          /* Set ES to point to Interrupt vector segment */
        MOV ES,AX
        CLI             /* Disable interrupts while IV is being changed */
        MOV AX,WORD PTR Int27Save /* Restore the offset ... */
        MOV ES:[27*4],AX /* ... of old interrupt handler */
        MOV AX,WORD PTR Int27Save+2 /* Restore the segment number ... */
        MOV ES:[27*4+2],AX /* ... of old interrupt handler */
        STI             /* Re-enable interrupts (modification finished) */
        POP AX          /* Restore registers */
        POP ES
    }
} /* main */

```

E:\>c_intr2d.exe
C+InASM 1: Press Ctrl-Break to continue.
C+InASM 1: Ctrl-Break has been pressed.

E:\>

```

/* c_intr3d.c - Change Ctrl-Break interrupt handler.
   C + ASM version, use DOS INT 21h AH=35h, 25h. */

#include <stdio.h>
#include <dos.h>

int Ctrl_Break_Flag;
long int Int27Save; /* 32bit used to hold old ISR address */

void interrupt Ctrl_Break_Handler(void)
{
    Ctrl_Break_Flag = 1;
    printf("C+InASM 2: Ctrl-Break has been pressed.\n");
}

void main(void)
{
    /* Int27Save = getvect(27);      Preserve old pointer */
    asm {
        PUSH DS          /* Preserve registers */
        PUSH ES
        PUSH AX
        PUSH BX

        MOV AH,35h        /* Set DOS interrupt handler */
        MOV AL,27         /* Set interrupt vector entry number */
        INT 21h          /* Invoke DOS */
        /* Preserve offset of old interrupt handler */
        MOV WORD PTR Int27Save,BX
        MOV AX,ES          /* Preserve segment number of old ... */
        MOV WORD PTR Int27Save+2,AX /* ... interrupt handler */

        /* setvect(27, Ctrl_Break_Handler); Set new interrupt handler */
        /* Set offset of new interrupt handler */
        MOV DX,OFFSET Ctrl_Break_Handler
        MOV AX,SEG Ctrl_Break_Handler /* Set segment number of new ... */
        MOV DS,AX          /* ... interrupt handler */
        MOV AH,25h          /* Set DOS interrupt handler */
        MOV AL,27          /* Set interrupt vector entry number */
        INT 21h          /* Invoke DOS */

        POP BX          /* Restore registers */
        POP AX
        POP ES
        POP DS
    }

    printf("C+InASM 2: Press Ctrl-Break to continue.\n");

    Ctrl_Break_Flag = 0;
    while (!Ctrl_Break_Flag)
        ; /* Do nothing */

    /* setvect(27,Int27Save);      Restore old pointer */
    asm {
        PUSH DS          /* Preserve registers */
        PUSH AX
        /* Restore number old interrupt handler */
        MOV DX,WORD PTR Int27Save
        MOV AX,WORD PTR Int27Save+2 /* Restore the number ... */
        MOV DS,AX          /* ... of old interrupt handler */
        MOV AH,25h          /* Set DOS interrupt handler */
        MOV AL,27          /* Set interrupt vector entry number */
        INT 21h          /* Invoke DOS */
        POP AX          /* Restore registers */
        POP DS
    }
} /* main */

```

E:\>c_intr3d.exe
C+InASM 2: Press Ctrl-Break to continue.
C+InASM 2: Ctrl-Break has been pressed.

E:\>

תוכניות דוגמא ctrlbrk1.asm

התוכנית mainint3.c כמעט זהה ל-c_intrld.c, למעט הعبارة שהרוטינה Ctrl_Break_Handler אינה מומשת שם אלא מוגדרת כ-extern ומצויה שהיא תהיה מומשת בפועל בקובץ אחר. בפועל היא מומשת פרוצדורה בשפת אסמבלי טהורה בקובץ ctrbrh1.asm. לפיכך מה שנחנו רואים כאן להבדיל מ- c_intrld.c היא השורה

```
extern void interrupt Ctrl_Break_Handler(void);
```

שורת הקימפבול של התוכנית המשלובת היא

```
tcc -ml -r mainint3.c ctrbrh1.asm
```

והתוכן של הקובץ ctrbrh1.asm. אשר לו האחרון כדי לשים לב ליעובדה שהחזרה של הרוטינה מומשת ע"י פקודת המכוונה IRET, הרוטינה משמשת/משוחררת את כל האורדרים בפקודות

```
PUSH AX  
PUSH BX
```

...

```
PUSH BP
```

ומשוחררת אותם בפקודות

```
POP BP  
POP DI
```

...

```
POP AX
```

מכיוון שהתוכנית מתكمפלת במודל LARGE, הקריאה ל-printf מחייבת דחיפת למחסנית של פוינטר מלא (גם מספר סגמנט וגם היסט) המומושכים בפקודות

```
MOV AX,SEG printf_str
```

....
PUSH AX
PUSH OFFSET printf_str

בנוסף ישנה פקודה

MOV DS,AX

בכדי להבטיח שה-printf של מודל LARGE יהיה לו את הערך שהוא מצפה לו. זאת מאחר שמדובר בROUTINE טיפול בפסיקה איננו יכולים לסמוך על ערכי העוגרים, שכן איננו יודעים בוודאות מהכן הגיעו ה-CPU לאן.

```

/* mainint3.c - Main for pure assembler interrupt handler */

#include <stdio.h>
#include <dos.h>

volatile int Ctrl_Break_Flag;
void interrupt (*Int27Save) (void); /* Pointer to function */

extern void interrupt Ctrl_Break_Handler(void);

void main(void)
{
    Int27Save = getvect(27); /* Preserve old pointer */
    setvect(27, Ctrl_Break_Handler);
    printf("C: Press Ctrl-Break to continue.\n");

    Ctrl_Break_Flag = 0;

    while (Ctrl_Break_Flag == 0)
        ; /* Do nothing */

    printf("Terminating ... \n");

    setvect(27, Int27Save); /* Restore old pointer */

} /* main */

```

```

E:\>tcc -ml -r- mainint3.c ctrlbrh1.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
mainint3.c:
ctrlbrh1.asm:
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

```

```

Assembling file: ctrlbrh1.ASM
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 418k

```

```
Turbo Link Version 5.0 Copyright (c) 1992 Borland International
```

```
Available memory 4112112
```

```
E:\>mainint3.exe
C: Press Ctrl-Break to continue.
ASM: Ctrl-Break has been pressed
Terminating ...
```

```
E:\USERS\EYTAN\ASM>
```

```

; ctrlbrh1.asm - pure assembler Ctrl-Break handler.

;
; .MODEL LARGE
; PUBLIC _Ctrl_Break_Handler
; EXTRN _Ctrl_Break_Flag:WORD
; EXTRN _printf:FAR
;
; void interrupt Ctrl_Break_Handler(void)
;
; .DATA
printf_str DB 'ASM: Ctrl-Break has been pressed',10,0
.CODE
_Ctrl_Break_Handler PROC FAR
    PUSH AX          ; Preseve all registers ...
    PUSH BX          ; ... other than FLAGS and SP.
    PUSH CX
    PUSH DX
    PUSH ES
    PUSH DS
    PUSH SI
    PUSH DI
    PUSH BP
    MOV AX,SEG _Ctrl_Break_Flag      ; Set ES to segment ...
    MOV ES,AX           ; ... of Ctrl_Break_Flag
;
;
; {
;     Ctrl_Break_Flag = 1;
;
    MOV ES:[_Ctrl_Break_Flag],1      ; Set global flag to 1
;
;     printf("ASM: Ctrl-Break has been pressed\n");
;
    MOV AX,SEG printf_str          ; Set DS to segment ...
    MOV DS,AX           ; ... of printf_str
    PUSH DS            ; Store complete address
    PUSH OFFSET printf_str        ; ... of printf_str
    CALL _printf
    ADD SP,4           ; Free parameter space
;
;
; }
;
    POP BP            ; Restore all registers ...
    POP DI            ; ... other than FLAGS and SP.
    POP SI
    POP DS
    POP ES
    POP DX
    POP CX
    POP BX
    POP AX
    IRET             ; Return from interrupt
_Ctrl_Break_Handler ENDP
END

```