CP 111 Computer Programming (2/2554 Section B01-B02)

Handout 10

<u>หัวข้อ:</u>

File I/O (Chapter 18.5 textbook)

Dynamic memory Allocation (Chapter 19.4 textbook): Dynamically-Sized array, `malloc()`, `free()`

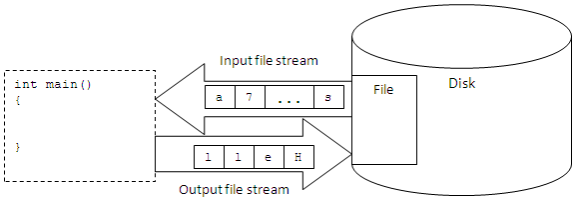
Miscellaneous C features:

- C preprocessor: Macro substitution (Appendix D.8 textbook),
- Conditional Expression operator: `expA ? expB : expC`
- `typedef`
- Conditional compilation using pre-processing directive

<u>File Input/Output</u>

1. ก่อนหน้านี้เราเขียนโปรแกรมภาษาซี ติดต่อกับ Input คือ ……. โดยใช้ standard library function ชื่อ ……และ ติดต่อกับ output คือ……..โดยใช้ฟังก์ชัน ……..
ในบทเรียนนี้เราจะศึกษาการเขียนโปรแกรมติดต่อกับ I/O ที่เก็บอยู่ในรูปของไฟล์ใน Harddisk

2. File Input/Output stream: ในเชิง concept เราจะมองการติดต่อระหว่างโปรแกรมกับ File หนึ่งๆ จะทำโดยผ่าน input/output file streams โดยที่โปรแกรมจะอ่านข้อมูลจากไฟล์หรือเขียนข้อมูลลงไฟล์โดยผ่าน file stream ซึ่งข้อมูลจะเรียงต่อกันตามลำดับ



3. การเขียนโปรแกรมติดต่อกับ File ในวิชานี้ เราจะแบ่ง file เป็น 2 ประเภท ได้แก่……....กับ…….

4. การเขียนโปรแกรมติดต่อกับ File สามารถ<u>สรุปได้เป็น 4 ขั้นตอนย่อยต่อไปนี้</u>

<u>(Step 1)</u> นิยามตัวแปรสำหรับระบุ File stream ให้มีชนิดแบบ `FILE *` เช่น `FILE *infile;`

Note: ชนิดตัวแปร FILE เป็น programmer-defined data type ที่ถูกนิยามไว้ใน <stdio.h>

<u>(Step 2)</u> เปิด file stream ซึ่งจะทำการเชื่อมโยงตัวแปรแบบ FILE * ที่ระบุ File stream เข้ากับไฟล์ที่อยู่บน harddisk โดยใช้ฟังก์ชัน `fopen`

```
FILE *fopen(char *filename_str, char *mode_str);
```

ซึ่งฟังก์ชัน fopen จะ map ตัวแปรแบบ FILE * ที่ระบุ File stream ให้เข้ากับไฟล์ที่ชื่อระบุโดยอาร์กิวเมนต์ `filename_str` บน harddisk ที่โปรแกรมต้องการจะติดต่อด้วย โดย mode ของการติดต่อจะระบุโดยอาร์กิวเมนต์ `mode_str`

<u>ตัวอย่าง</u>

```
char fname[50]="C:/Pradit/CP111/Exam/Midterm_Exam2.pdf"
FILE *infile;
       infile=fopen(fname,"r");
```

- ในกรณีที่การเปิด file stream สำเร็จฟังก์ชัน fopen จะคืนค่า pointer to FILE (FILE *)

- แต่ถ้ามีปัญหาและเปิดไฟล์ไม่สำเร็จ เช่น ไฟล์ที่ต้องการอ่านไม่มีอยู่บน hardisk หรือ program ไม่มี permission ที่จะเขียนไฟล์ลง Harddisk ฟังก์ชันจะคืนค่า NULL เพื่อแสดงว่าการเปิดไฟล์มีความผิดพลาด

ส่วนความหมายของสตริงที่ระบุ mode มีดังต่อไปนี้

mode	ความหมาย
"r"	read only (text file): เปิดไฟล์ที่มีอยู่แล้วเพื่อทำการอ่านอย่างเดียว
"w"	write only (text file): เปิดไฟล์เพื่อทำการเขียนอย่างเดียว (ถ้ามีไฟล์อยู่แล้วไฟล์เดิมจะถูกลบไป)
"a"	append only (text file) หรือเขียนข้อมูลเพิ่มโดยต่อท้ายไฟล์ที่มีอยู่แล้ว
"r+"	read and write เริ่มที่ต้นไฟล์ (text file) โดยไฟล์ที่จะ open ต้องมีอยู่แล้ว
"w+"	read and write เริ่มที่ต้นไฟล์ (text file) ถ้าเป็นไฟล์ที่มีข้อมูลอยู่แล้วข้อมูลจะถูกลบไป
"a+"	read and append ถ้าไฟล์ที่จะ open ไม่มีอยู่จะสร้างไฟล์ใหม่
"b"	ใช้ระบุพร้อมกับ mode ข้างบนเพื่อเป็นการระบุว่าต้องการติดต่อกับไฟล์แบบ binary file เช่น "rb" (read only กับไฟล์แบบ binary), "wb" (write only), "ab", "r+b"

<u>(Step 3)</u> อ่านข้อมูลจากไฟล์หรือเขียนข้อมูลลงไฟล์ผ่าน file stream ที่ได้เปิดไว้ก่อนหน้านี้โดยใช้ standard library function ต่างๆ ได้แก่

fscanf	***int fscanf(FILE* stream, char* format_str, argument)*** - อ่านข้อมูลจากไฟล์ (ระบุโดย stream) ในรูปแบบที่เหมือนกับ scanf - คืนค่า EOF ถ้าอ่านที่ตำแหน่งสุดท้ายของไฟล์ - เช่น `fscanf(infile,"%d %d",&a,&b);`
fprintf	***int fprintf(FILE* stream, char* format_str, argument)*** - เขียนข้อมูลลงไฟล์ (ระบุโดย stream) ในรูปแบบที่เหมือนกับ printf - เช่น `fprintf(outfile,"%d %.2f %s\n",x,y,str);`
fgets	***int fgets(char *str,int N, FILE *stream)*** - อ่านสตริงจากไฟล์จำนวน 1 บรรทัด (จนกว่าจะเจอ '\n') ในรูปแบบที่เหมือนกับ gets นั้นคืออ่านสตริงมาเก็บที่ str แต่ไม่เกิน N ตัวอักษร (มีตัวอักษร \n อยู่ที่ตัวสุดท้ายด้วย) - คืนค่า NULL ถ้าอ่านจนถึงท้ายไฟล์ - เช่น `char buff[501];` `fgets(buff,500,infile);`
fgetc	***int fgetc(FILE* stream)*** - อ่านตัวอักษรจากไฟล์จำนวน 1 ตัวอักษรและคืนค่ารหัสแอสกีของตัวอักษรนั้นกลับไป - คืนค่า EOF ถ้าอ่านที่ตำแหน่งสุดท้ายของไฟล์ - เช่น `ch=fgetc(infile);`
fputs	***int fputs(char *string, FILE *stream)*** - เขียนสตริงลงไฟล์ - เช่น `fputs(str,outfile);`

fputc	***int fputc(int ch, FILE* stream)*** - เขียนตัวอักษร 1 ตัวลงไฟล์ - เช่น `fputc(str[0],outfile);`

ฟังก์ชันข้างบนใช้เขียนหรืออ่านไฟล์แบบ text file ในกรณีที่ต้องการเขียนข้อมูลลง binary file ให้ใช้ฟังก์ชัน fread และ fwrite ซึ่งมีคำอธิบายได้ดังนี้

fread	`int fread( void* buffer, int size, int count, FILE* stream)`
	- อ่านข้อมูลที่อยู่ในตัวแปรที่แอดเดรสระบุโดย buffer จำนวน count ตัว แต่ละตัวขนาด size ไบต์ลงไฟล์
	- คืนค่าจำนวนไบต์ที่อ่านข้อมูลจริงๆได้จากไฟล์
	เช่น
	`int x,data[5];`
	`fread(&x,sizeof(int),1,infile);`
	`fread(data,sizeof(int),5,infile);`

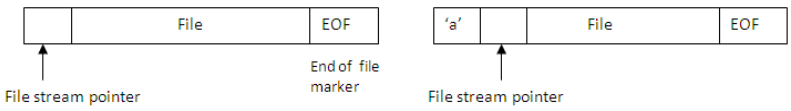
fwrite	`int fwrite(void* buffer,int size,int count,FILE* stream)`
	- เขียนข้อมูลที่อยู่ในตัวแปรที่แอดเดรสระบุโดย buffer จำนวน count ตัว แต่ละตัวขนาด size ไบต์ลงไฟล์
	- เขียนตัวอักษร 1 ตัวลงไฟล์
	เช่น
	`int x,data[5];`
	`fwrite(&x,sizeof(int),1,outfile);`
	`fwrite(data,sizeof(int),5,outfile);`

<u>หมายเหตุ</u> (1) การอ่านหรือเขียนข้อมูลกับไฟล์ จะกระทำที่ตำแหน่งปัจจุบันที่ FILE stream pointer กำลังระบุไว้ และหลังจากการอ่านหรือเขียนข้อมูล 1 ครั้งกับไฟล์ ตำแหน่งของ FILE stream pointer จะเลื่อนไปที่ตำแหน่งถัดไป

(2) ในไฟล์ทุกๆไฟล์จะมีข้อมูล Marker ค่าเท่ากับ 0 ไว้ที่ท้าย File stream หนึ่งๆ สามารถอ้างถึงในโปรแกรมโดยใช้สัญลักษณ์ EOF เพื่อใช้เป็นตัวระบุการสิ้นสุดของข้อมูลในไฟล์

(3) เราสามารถตรวจสอบว่าตำแหน่งปัจจุบันของ file stream pointer อยู่ที่ท้ายไฟล์หรือไม่ โดยใช้ฟังก์ชัน feof ซึ่งจะคืนค่าไม่เท่ากับ 0 ถ้าตำแหน่งปัจจุบันอยู่ที่ EOF

```
int feof(FILE *stream)
```

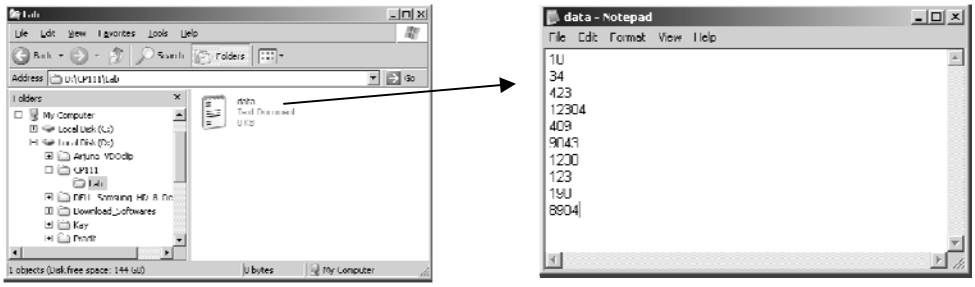


<u>(Step 4)</u> ปิด file stream โดยใช้ฟังก์ชัน `fclose`.

```
int fclose(FILE *stream)
```

เช่น `fclose(infile);`

5. โปรแกรมตัวอย่าง: จงเขียนโปรแกรมเพื่ออ่านข้อมูลที่อยู่ในไฟล์ (แบบ text file) ชื่อ data.txt ที่เก็บในไดเร็คทอรี D:\CP111\Lab\ ให้ลงมาเก็บไว้ในอาร์เรย์ A สมมติว่าข้อมูลมีไม่เกิน 500 ตัว ดังตัวอย่างข้างล่าง



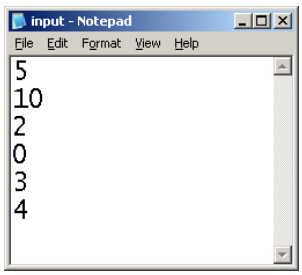
```
#include <stdio.h>
int main()
{
int data[500],x,retval,N;

……................................................

        ……..............................................................
        if(……..............................................................)
        {
                printf("Can't open file\n");
                return -1;
        }
        N=0;
        while(1)
        {
                retval=……..............................................................
                if(……..............................................................)
                        break;
                data[N]=x;
                N=N+1;
        }
        ……..............................................................
        return 0;
}
```

6. โจทย์ตัวอย่าง (ข้อสอบเก่าปี 2553) กำหนดให้ไฟล์ข้อมูลแบบ Text file มีรูปแบบดังนี้ บรรทัดแรกคือตัวเลขที่ระบุจำนวนข้อมูลที่ตามมา สมมติว่าคือ N หลังจากนั้นบรรทัดที่ 2 ถึง N+1 จะเป็นตัวเลขแบบจำนวนเต็มที่มากกว่าหรือเท่ากับศูนย์บรรทัดละตัว

จงเขียนโค้ดในฟังก์ชัน `Sum` ที่ทำการคำนวณและคืนค่าผลรวมของข้อมูลทั้ง N จำนวนที่อยู่ในไฟล์ที่ชื่อไฟล์ระบุอยู่ในอาร์กิวเมนต์ `filename` <u>โดยในกรณีที่การเปิดไฟล์ไม่สำเร็จให้ฟังก์ชันคืนค่า -1</u>

<u>ตัวอย่าง</u> เมื่อฟังก์ชันทำการประมวลผลไฟล์ input.dat แสดงดังข้างล่าง จะคืนค่า 19 (=10+2+0+3+4)

```
int Sum(char *filename)
{
……..................................................
int N,sumval,x;
        ……..................................................
        if(……..................................)
                return -1;
        ……..................................................
        sumval =0;
        for(i=0;i<N;i++)
        {
                ……..................................................
                sumval = sumval +x;
        }
        ……..................................................
        return (sumval);
}
```



7. โจทย์ตัวอย่าง (ดัดแปลงจากข้อสอบเก่าปี 2552 และ 2553) จงระบุผลลัพธ์ของโปรแกรมข้างล่าง

```
#include <stdio.h>
int main()
{
FILE *infile,*outfile;
char str1[20]="3423";
char str2[20];
int retval,x,y,N;
      outfile=fopen("data.txt","w");
      if(outfile==NULL)
      {
            printf("Can't open file\n");
            return -1;
      }
      fputc('8',outfile);
      x=563;
      fprintf(outfile,"%d\n",x);
      fputs(str1,outfile);
      fputc('\n',outfile);
      fclose(outfile);
            -- continue --
```

```
            -- continue --

      infile=fopen("data.txt","r");
      if(infile==NULL)
      {
            printf("Can't open file\n");
            return -1;
      }
      y=fgetc(infile);
      y=y-'0';
      fgets(str2,30,infile);
      fscanf(infile,"%d",&x);
      printf("%d %c %d\n",y,str2[1],x);
      fclose(infile);
      return 0;

}
```

8. Binary file:

ในกรณีของ binary file ข้อมูลที่จัดเก็บในไฟล์จะมีในลักษณะเดียวกับข้อมูล binary ที่เก็บในตัวแปร เช่นข้อมูลแบบ int ที่มีค่า 13890234 จะเก็บในไฟล์ binary โดยใช้ 4 bytes (ในรูปของ 32-bit 2's complement)

การประมวลผลกับไฟล์แบบ binary จะใช้คำสั่ง fread และ fwrite นอกจากนี้อาจจำเป็นต้องใช้ฟังก์ชัน fseek เพื่อเลื่อนตำแหน่งของ file stream pointer

(<u>หมายเหตุ</u> ในกรณีของ text file ข้อมูลที่เก็บในไฟล์จะเป็นรหัสแอสกีของตัวอักษร)

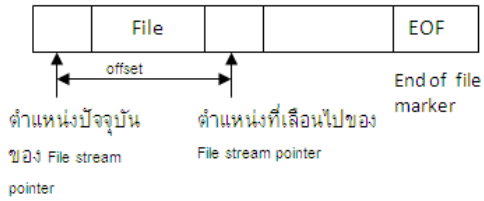
```
int fseek(FILE *stream, int offset, int origin)
```

ทำการเลื่อนตำแหน่ง file stream pointer ไปเป็นจำนวนตำแหน่งเท่ากับ offset ตำแหน่ง โดยการเลื่อนจะเทียบกับตำแหน่งที่ระบุโดย origin

`origin=SEEK_SET` เทียบกับต้นไฟล์

`origin=SEEK_CUR` เทียบกับตำแหน่งปัจจุบันของ file stream pointer

`origin=SEEK_END` เทียบกับท้ายไฟล์



9. โจทย์ตัวอย่าง (ข้อสอบเก่าปี 2553):

จงเขียนโค้ดในฟังก์ชัน `Save_IntArray_BinaryFile` เพื่อทำการบันทึกข้อมูลที่อยู่ใน array ที่ระบุโดยอาร์กิวเมนต์ `data` และขนาด `N` <u>ลงเก็บในไฟล์แบบ binary</u> ที่ชื่อไฟล์ระบุโดยอาร์กิวเมนต์ `filename`

โดยที่ file format คือ 4 ไบต์แรก คือจำนวน element ในอาร์เรย์ (กำหนดให้ `sizeof(int)` มีค่าเท่ากับ 4) และทุกๆ 4 ไบต์จะเก็บข้อมูลของอาร์เรย์ตั้งแต่ตัวที่ 0 จนถึงตัวที่ N-1 เรียงต่อกันไป

และใน<u>*กรณีที่การเปิดไฟล์ไม่สำเร็จฟังก์ชันจะคืนค่า -1 และถ้าฟังก์ชันทำงานสำเร็จจะคืนค่า 0*</u>

```
int Save_IntArray_BinaryFile(int *data, int N, char *filename)
{
int i;
……................................................
        ……..……..……..
```

void free(void *ptr)

ทำการคืนพื้นที่ในหน่วยความจำที่ได้จองไว้โดย malloc โดย memory block ที่จะ free จะอยู่ที่แอดเดรสระบุโดย ptr ตัวอย่างการใช้เช่น `free(data);`

<u>หมายเหตุ</u> เราจำเป็นที่จะต้อง free หน่วยความจำที่ได้จองไว้ มิฉะนั้นอาจจะทำให้เกิด memory leak จนทำให้โปรแกรมกินทรัพยากร memory ของคอมพิวเตอร์ไปจนทำให้คอมพิวเตอร์ไม่สามารถทำงานต่อได้

3. โจทย์ตัวอย่าง: จงเขียนโปรแกรมคำนวณ Mean และ Std ของอาร์เรย์ โดยที่บรรทัดแรกของข้อมูลนำเข้าจะระบุจำนวนข้อมูลในอาร์เรย์ หลังจากนั้นจะเป็นข้อมูลบรรทัดละตัว

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
int *data,N,i,sum;
double avg,std,sumsq;
    scanf("%d",&N);

    ……................................................................

   if(……..................................................)
   {
       printf("Can't allocate memory\n");
       return -1;
    }
    sum=0;
    for(i=0;i<N;i++)
    {
         scanf("%d",&data[i]);
         sum=sum+data[i];
    }
    avg=(double)sum/n;
```

```
  -- continue --


  sumsq=0;
  for(i=0;i<N;i++)
    sumsq=sumsq+(data[i]-avg)*(data[i]-avg);

  std=sqrt(sumsq/N);


  ……..................................................................
  return 0;
}
```

<u>Miscellaneous C features</u>

1. C preprocessor: Macro substitution

เราสามารถใช้ `#define` preprocessing directive เพื่อสั่งให้ C preprocessor เพื่อจะแทนสัญลักษณ์ที่เป็นชุดของตัวอักษรที่ปรากฏในไฟล์ <u>โดยทั่วไปเรียกสัญลักษณ์นี้ว่า Macro</u> ตัวอย่างเช่น สัญลักษณ์ N และ M ในส่วนย่อยของโปรแกรมข้างล่างด้านซ้ายจะถูกแทนด้วยตัวเลข 100 และ 20 ในขั้นตอนก่อนการ compile โดย preprocessor

```
#define N 100
#define M 20
int main()
{
int i,A[N];
           for(i=0;i<N;i++)
            A[i]=i+M;
}
```

→

```
int main()
{
 int i,A[100];
        for(i=0;i<100;i++)
         A[i]=i+20;
 }
```

ข้อดีของการใช้ #define คือทำให้เราสามารถรวมกลุ่มของค่าคงที่ที่อาจจะปรากฏหลายที่ในโปรแกรม โดยให้นิยามค่าไว้ที่เดียวที่ต้นไฟล์ เมื่อต้องการแก้ไขค่า ก็เพียงมาแก้ที่ #define จุดเดียว

นอกจากนี้เรายังสามารถใช้ #define มานิยามส่วนย่อยของโค้ดที่สั้นๆ แต่จะถูกใช้บ่อย ดังตัวอย่างโปรแกรมข้างล่างด้านซ้าย โดยหลังจาก C preprocessor ทำการประมวลผล มันจะทำการแทนโค้ดที่นิยาม SQUARE และ REMAINDER และ MULT ทำให้โค้ดด้านขวา

```
#define SQUARE(x) ((x)*(x))
#define REMAINDER(x,y) ((x)%(y))
#define MULT(x,y)  (x*y)
int main()
{
int a,b,c,d,e;
      c=SQUARE(a);
      d=REMAINDER(a,b);
      e=MULT(a+b,a+b);
}
```

→

```
 int main()
{
   int a,b,c,d;
      c=((a)*(a));
      d=((a)%(b));
      e=.................................
   }
```

จะเห็นได้ว่าการใช้ #define ในกรณี MULT(a+b,a+b) จะมีข้อผิดพลาด ซึ่งจะให้ได้โค้ด (a+b*a+b) ซึ่งไม่เท่ากับ (a+b)*(a+b) วิธีแก้คือใช้วงเล็บครอบอาร์กิวเมนต์ของ MACRO

……

2. Conditional Expression Operators

รูปแบบ: `{expressionA} ? {expressionB} : {expressionC}`

ความหมาย: ถ้า expressionA มีค่าเป็น TRUE ค่าของทั้ง expression จะเท่ากับค่าของ expressionB มิฉะนั้นค่าของทั้ง expression จะเท่ากับค่าของ expressionC

ตัวอย่าง:

```
int x,y,w;
w = (y>x ? y+x : y-x);
```

ถ้า x=12 y=20	`(y>x ? y+x : y-x)` มีค่าเท่ากับ	`w` มีค่าเท่ากับ .....
ถ้า x=12 y=10	`(y>x ? y+x : y-x)` มีค่าเท่ากับ	`w` มีค่าเท่ากับ .....

3. โจทย์ตัวอย่าง

(1) จงระบุผลลัพธ์การทำงานของโปรแกรมเมื่อผู้ใช้ป้อน 1 2 3

```
void main()
{
int x,y,z,a;
      scanf("%d %d %d",&x,&y,&z);
      a = ((z > x) ? (y+(2*x)) : (y-z));
     printf("%d\n",a);
}
```

(2) จงเขียน ชื่อmacro เพื่อนิยาม `MAX(x,y)`ซึ่งจะทำการหาค่ามากที่สุดระหว่างค่าของอาร์กิวเมนต์ x และ y

……

(3) สมมติมีการประกาศตัวแปร y, x1, x2, x3 เป็นแบบ int จงเขียน statement 1 บรรทัด ที่ใช้ macro `MAX` ในข้อ (2) เพื่อทำการกำหนดค่าของตัวแปร y ให้เท่ากับค่าที่มากที่สุดในบรรดาค่าของ x1, x2 และ x3

……

(4) จงเขียน ชื่อmacro เพื่อนิยาม `ABS(x)`ซึ่งจะทำการหาค่าสัมบูรณ์ของอาร์กิวเมนต์ x

……

4. `typedef`

ในภาษา C ได้อนุญาตให้ programmer นิยามชนิดของข้อมูล (data type) ในโปรแกรมเพื่อทำให้โค้ดอ่านง่ายในรูปแบบ

```
typedef type name;
```

โดยที่ `type` หมายถึง basic data type (`int, char, double`) หรือ derived data type (array, pointer, structure)

เช่น `typedef unsigned char BYTE;`

จะนิยามชนิดข้อมูลที่ชื่อ `BYTE` โดยให้เทียบเท่ากับชนิดข้อมูล `unsigned char`

`typedef unsigned long int UINT64;`

จะนิยามชนิดข้อมูลที่ชื่อ `UINT64` โดยให้เทียบเท่ากับชนิดข้อมูล `unsigned long int`

<u>ตัวอย่างโปรแกรม</u>

```
#include <stdio.h>

#define MAX(x,y) (x)>(y)?(x):(y)
typedef unsigned char BYTE;
typedef unsigned long int UINT64;

int main()
{ int x,y;
  UINT64 a,b,c;
  BYTE d;

      c=MAX(a,b);
      d=(BYTE)c;
      return 0;
}
```

<u>หมายเหตุ</u> `typedef` จะถูกใช้อย่างมากในบทถัดไปในการนิยามชนิดข้อมูลแบบ structure

5. Conditional compilation using pre-processor directive (`#ifdef, #ifndef, #else, #endif`)

```
#ifdef name
      Program text 1
#else
      Program text 2
#endif
```

- เราสามารถเลือกบางส่วนของโปรแกรมที่จะนำมา compile

- ถ้ามีการนิยาม name โดยใช้ #define โค้ดส่วน Program text 1 (ตั้งแต่ #ifdef ถึง #else) จะถูกนำไป compile โดยโค้ดส่วน Program text 2 จะเสมือนว่าไม่มีอยู่ในโปรแกรม

- ถ้าไม่มีการนิยาม name โค้ดส่วน Program text 2 (ตั้งแต่ #else ถึง #endif) จะถูกนำไป compile

<u>ตัวอย่าง</u>

```
#define _USE_CODE_A

int main()
{
 int x,y,z;

   x=10;
   y=20;
   #ifdef _USE_CODE_A
       z=y*x;
   #else
       z=(y+x);
   #endif
   printf("%d\n",z);
   return 0;
}
```

Q1: ผลลัพธ์ของโปรแกรมคือ

……..

Q2: ถ้าลบบรรทัด #define _USE_CODE_A ออกแล้ว compile โปรแกรมใหม่ ผลลัพธ์ของโปรแกรมคือ

……..

ตัวอย่างที่ใช้งานในทางปฏิบัติ

```
#define _DEBUG

double Avg(int *A,int n)
{
int i,sum;
   sum=0;
   for(i=0;i<n;i++)
   {
     #ifdef _DEBUG
       printf("%d %d\n",i,A[i]);
     #endif

     sum=sum+A[i];
   }

   #ifdef DEBUG
      printf("%d %d\n",sum,n);
   #endif

   return((double)sum/n);
}
```

```
#define _WINDOW

#ifdef _LINUX
  void func(char *filename,int mode)
#endif

#ifdef _WINDOW
 void func(char *filename,int mode, int par1)
#endif
{
int x,y;

    #ifdef _LINUX
       y=10;
    #endif

   #ifdef _WINDOW
      y=par1;
   #endif
    ...
}
```

6. โจทย์ตัวอย่าง (ข้อสอบปี 2553)

```
#include <stdio.h>
#define CP111_DEBUG 1
void main()
{
 int x,y,z;
        scanf("%d %d",&x,&y);
        #ifdef CP111_DEBUG
           if(x<y)
                 printf("LT\n");
           else
                 printf("GT\n");
        #endif
         #ifndef _CODE_A
              printf("%d\n",x+y);
         #else
              printf("%d\n",x-y);
         #endif
}
```

(1) เมื่อโปรแกรมข้างบนทำงานโดยผู้ใช้ป้อนค่า 1 2 ที่คีย์บอร์ด ผลลัพธ์ที่หน้าจอจะเป็นอย่างไร

(2) ถ้าเราลบบรรทัด `#define CP111_DEBUG 1` ออก แล้วเขียนโค้ด `#define _CODE_A 1` แทนลงไปหลังจากนั้นทำการคอมไพล์โปรแกรมใหม่ เมื่อโปรแกรมข้างบนทำงานโดยผู้ใช้ป้อนค่า 1 2 ที่คีย์บอร์ด ผลลัพธ์ที่หน้าจอจะเป็นอย่างไร

<u>Standard I/O file stream (เพิ่มเติมเรื่อง FILE)</u>

ในโปรแกรมภาษา C เราสามารถอ้างถึง standard file stream ที่ได้เชื่อมโยง (โดย default) ไว้กับ I/O โดยการอ้างถึง file stream pointer ต่อไปนี้: `stdin, stdout, stderr` โดยที่เราสามารถใช้ฟังก์ชันเขียนและอ่านไฟล์เพื่อทำการรับค่าจาก standard input (keyboard) แสดงผลออก standard output (display)

`stdin` (read-only): ซึ่งจะเชื่อมต่อกับ keyboard	ตัวอย่าง
	`fprintf(stdout,"%d\n",x);/* เทียบเท่ากับ printf("%d\n",x)); */`
	`fprintf(stderr,"%d\n",y);`
	`fscanf(stdin,"%d",&y);/* เทียบเท่ากับ scanf("%d",&y)); */`
	`ch=fgetc(stdin);`
	`fgets(str,100,stdin);`

`stdout` (write-only): ซึ่งจะเชื่อมต่อกับ display	
`stderr` (write -only): ซึ่งจะเชื่อมต่อกับ display	