ÔN TẬP CÁC KIẾN THỨC CB VỀ LẬP TRÌNH C

A.CÁC PHẦN TỬ CƠ BẢN:

- Các phép tính:
  - **Phép toán cơ bản:** + , - , * , / , %(lấy phần dư).
  - **Phép toán tăng giảm**: **x++ hay ++x** (**x--,--x):** tăng (giảm) x xuống 1 đơn vị.
  - **Phép toán trên Bit**: **&** (and), **|** (or), **^** (XOR), **<<** (dịch trái), **>>** (dịch phải).
  - **Biểu thức điều kiện**: cú pháp: **BT1? BT2: BT3** (*BT: Biểu thức*)**.** Nếu BT1 đúng(<>0) thì BT điều kiện trên = giá trị của BT2 ngược lại = giá trị của BT3 nếu BT1 sai (=0). **VD:** `printf("Min of a & b: %d",a<b? a:b);`
- Kiểu dữ liệu:
  - **int** (-32768 ... 32767) (2 hoặc 4 Byte); **unsigned int** (0 ... 65535).
  - **char** (-128 ... 127) (1 Byte); **unsigned char** (0 ... 255).
  - **long** (-2147483648 ... 2147483647) (4 Byte). **unsigned long** (0 ... 4294967297).
  - **float** (3.4 E-38 ... 3.4E+38) (4 Byte).
  - **double** (1.7E-308 ... 1.7E+308) (8 Byte). **long double** (3.4E-4932 ... 1.1E4932).
  - **Định nghĩa kiểu với typedef: typedef khaibáo.... VD:** tạo một kiểu ma trận kích thước [20][20] với tên là matrix : `typedef float matrix[20][20];`
- Biến, hằng số:
  - Khai báo biến: **kiểudữliệu tênbiến;** hay **kiểudữliệu tênbiến = giátrịkhởiđầu;**
  - Khai báo hằng số: **const type NAME=giátrị;** hay **#define** tênhằngsố **gíatrị;**
  - **VD:** `const float Pi=3.14;` `#define Pi 3.14;`

- Các hàm cơ bản : (***Các hàm kiểm tra được khai báo trong <ctype.h>).***
  - **fabs(x):** lấy giá trị tuyệt đối.
  - **sin(x), cos(x), tan(x):** lấy sin, cos, tang của x.
  - **floor(x):** Lấy phần nguyên của số x.
  - **exp(x):** tính e mũ x (e^x).
  - **log(x):** tính ln(x).
  - **log10(x)**: tính lg(x).
  - **pow(x,n)**: tính x mũ n (x^n).
  - **M_PI:** Hằng số Pi.
  - **flushall():** xoá bộ đệm bàn phím.
  - **void calloc(unsigned n, unsigned size): cấp phát vùng nhớ n***size byte, nếu thành công hàm trả về địa đầu vùng nhớ được cấp, ngược lại trả về NULL. Để giải phóng vùng nhớ được cấp phát bởi malloc hay calloc do pt trỏ tới ta dùng : void **free(void *pt)**; <alloc.h>
  - **int getchar(void)**: nhận một ký tự từ bàn phím (stdin), trả về ký tự nhận được.
  - **int random(int n)**: cho một giá trị ngẫu nhiên từ 0 ... n-1. <stdlib.h>
  - **int tolower(int c):** đổi ký tự chữ hoa sang chữ thường. <ctype.h>
  - **int toupper(int c):** đổi ký tự chữ thường sang chữ hoa.<ctype.h>
  - **int isalnum(int c)**: kiểm tra c có phải là ký tự alphanumeric? (chữ cái hay số).
  - **int isalpha(intc)**: kiểm tra c có phải là chữ cái không?
  - **int isdigit(int c):** kiểm tra c có phải là chữ số không?
  - Int ispunct(int c): kiểm tra c có phải là ký tự chấm câu không?
  - **int isxdigit(int c):** kiểm tra c có phải là chữ số hệ 16 không?
  - **int isupper(int c): k**iểm tra c có phải là chữ hoa (từ A đến Z) không.

  Chi chú: Các hàm kiểm tra nếu thoả thì trả về giá trị <>0, ngược lại trả về 0.
- Phép gán: **Tên biến = Biểuthức / biến;**
- Xuất nhập dữ liệu:
  - **Xuất:** printf("ký tự điều khiển",bt1,bt2,...);

    Trong đó danh sách biểu thức có thể bao gồm biểu thức, số, hay "văn bản", các đối tượng phải cách nhau bởi dấu phẩy.

    Vd: `printf( "Nam %d la the ky %d: ", 1999+2, 40/2);`

⇨ `Nam 2001 la the ky 20`

- **Nhập:** scanf("các ký tự định dạng",biến1, biến2,...). `scanf("%d%d",&a,&b);`

B.CẤU TRÚC ĐIỀU KIỆN, RẼ NHÁNH VÀ VÒNG LẶP:

- **if (bieuthuc) lệnh1;**

  hay **if (bieuthuc) lệnh1;**

  else lệnh2;

  Trong cấu trúc thứ 1, nếu **bieuthuc** cho giá trị khác 0 thì thực hiện lệnh1 và =0 thì thôi, còn trong cấu trúc thứ 2 nếu **bieuthuc** cho giá trị khác 0 thì thực hiện lệnh1 và =0 thì thực hiện lệnh2. Chú ý là trong thân if và else chỉ **là 1 câu lệnh đơn**, nếu có nhiều lệnh thì phải lồng vào {...};

- **switch** (biểu thức)

  {

  case x: lệnh; break;

  .....

  case n: lệnh; break;

  default :lệnh; break;

  }

  Ý nghĩa: câu lệnh rẽ nhánh switch sẽ tính giá trị của **biểu thức** và thực hiện lệnh tương ứng trong case nào có giá trị này. Nếu không có giá trị trong case nào = giá trị của biểu thức thì thực hiện lệnh sau default.

- **for** (biếnđkh[= giá trị đầu]; **điều kiện ; phép toán thay đổi giá trị biến đkh**) **lệnh ;**

  Nếu có nhiều lệnh thì phải lồng vào {... };

- **while** (biểu thức) **lệnh**; Nếu có nhiều lệnh thì lồng vào{... };

  Ý nghĩa: Trong khi **biểu thức <>0** thì thực hiện lệnh.

- **do** lệnh **while**(biểu thức) **;** Nếu có nhiều lệnh thì lồng vào {}.

  Ý nghĩa**:** Thực hiện lệnh nhiều lần trong **biểu thức** <>0 .

- **goto và nhãn:** (*Nhãn (handle) có dạng như tên biến, đứng trước dấu 2 chấm :*)

Cú pháp: **goto** nhãn**;** Khi gặp dòng này máy sẽ nhãy đến câu lệnh có nhãn viết sau từ khoá goto.

❖ CONTINUE, BREAK VÀ SIZEOF:

- **Câu lệnh continue** dùng để bắt đầu một vòng lặp mới của chu trình bên trong nhất chứa nó. Trong thân for máy sẽ chuyển đến bước khởi đầu kế tiếp. Còn trong while và do while máy sẽ chuyển đến xác định giá trị biểu thức sau while và tiến hành kiểm tra điều kiện kết thúc vòng lặp, (continue không dùng cho switch).
- **Câu lệnh break** cho phép ta thoát khỏi for, while, do while và switch. Khi có nhiều vòng lặp lồng nhau, break sẽ đưa ra khỏi vòng lặp hay switch bên trong nhất.
- **Toán tử sizeof** cho kích thước (byte) của một kiểu hay một đối tượng dữ liệu, cú pháp:

  sizeof(kiểudữliệu) hay **sizeof(đốitượngdữliệu).** Các kiểu dữ liệu như int, char, float, kiểu tự định nghĩa,... Các đối tượng dữ liệu như mảng, cấu trúc, biến,... Thường dùng để xác định số phần tử của mảng khi được khởi đầu:

```
int a[]={1,2,3,4,5,6};
int spt=sizeof(a)/sizeof(int);
```

C. HÀM (FUNCTION):

- Định nghĩa Hàm:

  <Kiểudữliệu / void> TênHàm (Danhsáchthamsố)

  <Khai báo **kiểu cho các tham số**>;

  {Khai báo **biến cục bộ và thân hàm**};

- Sử dụng Hàm:

  Tên hàm (Danhsáchcácthamsốthựcnếucó);

- Truyền tham số:

  + Truyền bằng trị: đây là chế độ truyền tham số mặc định của C. Với cách truyền này giá trị của tham số thực sẽ không bị thay đổi sau khi hàm thực hiện.

  +Truyền bằng biến: tức muốn truyền cả nội dung lẫn địa chỉ của biến. Khi đó ta phải sử dụng biến con trỏ.

D. MẢNG (ARRAY):

- Mảng 1 chiều:
  - Khai báo tường minh: **Kiểu tenmang[số phần tử]** ;
  - Khai báo không tường minh: **Kiểu tenmang []**; Cách khai báo này được sử dụng trong các trường hợp: vừa khai báo vừa gán trị (vd: *int a[]={1,2,3,7};*), mảng là tham số hàm (vd: *int nhapmang(int a[],int n, int m) ;*).
  - Xuất/ nhập mảng: ta dùng vòng lặp for(vd *for (i=1; i<n; i++) ....;*), hay ta có thể tạo một hàm để nhập mảng với số phần tử không biết trước, kết thúc khi nhập số 0.

```
void nhap(int a[], int*n)
   {
    for (*n=0;*n<20,(*n)++)
      {
     printf("A[%d]:",*n);
     scanf("%d",&a[*n]);
     if (a[*n]==0) break;
     }
    }
```

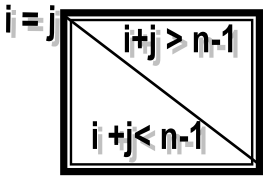
  - Sắp xếp mảng giảm dần (Buble sort).

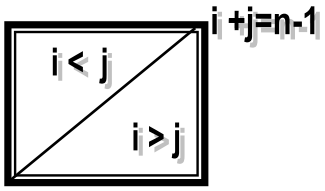
```
for (i=0;i<n-1;i++)
      for (j=i+1;j<n;j++)
      if (A[i]<A[j])
         {
             tam=A[i];
             A[i]=A[j];
             A[j]=tam;
         }
```

- Mảng 2 chiều:
  - Khai báo tường minh: **Kiểu tenmang[sốcột][sốdòng]** ; Vd: *a[20][10];*
  - Khai báo không tường minh: **Kiểu tenmang [ ][sốcột]** ; sử dụng cách này trong các trường hợp: vừa khai báo vừa gán trị(Vd: *int a[][3]={{1,2,3},{-1,-2,-3}};*), mảng là tham số hàm(Vd: *void nhap(a[][max], int n);*).

- Xuất/ nhập mảng: ta dùng 2 vòng lặp **for I:=1 to n do for j:=1 to m do...** Chú ý là ta không thể dùng phép toán địa chỉ để nhập các phần tử của mảng hai chiều (ngoại trừ mảng kiểu int), mà phải thông qua một biến tạm trung gian.
- Thao tác: (i: đkh dòng, j: đkh cột, n: số dòng)

+Đường chéo:





+Các phần tử đối xứng:

-Qua đường chéo chính: a[i][j] và a[j][i].

-Qua đường chéo phụ: a[i][j] và a[n-j-1][n-i-1].

-Qua tâm: a[i][j] và a[n-i-1][n-j-1].

- Duyệt ma trận:

+ Nữa trên đ/c chính:

```
for (i=0; i<n-1; i++)
        for (j=i+1; j<n; j++) ....
```

+ Nữa dưới đ/c chính:

```
for (i=1; i<n; i++)
        for (j=0; j<i; j++) ....
```

+ Nữa trái ma trận:

```
for (i=0; i<n; i++)
        for (j=0; j<n/2; j++) ....
```

+ Nằm trên đường chéo chính:

```
for (i=0; i<n; i++)
        với a[i][i] là các pt trên đ/c chính.
```

+ Đ/C song song với đ/c chính:

```
for (i=0; i<n-k; i++)
        với a[i][i+k] là các pt // với đ/c chính.
```

E. CON TRỎ (POINTER):

- Khai báo:

  + **<Kiểu dữ liệu> *Tên_con_trỏ; Vd:** `int *pt,*x;`

  + Mảng con trỏ: **<Kiểu dữ liệu> *Tên_con_trỏ[sốphầntửmảng]; Vd**: `int *pt[10];`

  + Hàm con trỏ: ***Tên_hàm (Danh sách các tham số)**...;

- Các phép toán trên con trỏ:

  + Phép gán: khi thực hiện phép gán cho biến con trỏ thì biểu thức bên phải phép gán phải có giá trị là địa chỉ của một biến hay mảng và phải cùng **kiểu với kiểu con trỏ**.

  Vd: `int x,*pt; ... p=&a;`

  + Phép toán lấy nội dung của biến do con trỏ chỉ đến(*): ***tên_biến_con_trỏ. VD:** `y=*x;`

  + Phép toán tăng giảm: **pt++,pt--;** Chú ý trong phép toán này là mỗi lần tăng (giảm) là tăng (giảm) kích thước của kiểu dữ liệu mà nó trỏ đến (chứ không phải là một đơn vị).

- Con trỏ là tham số Hàm: Nếu một hàm có tham số là kiểu con trỏ thì khi truyền tham số này cho hàm phải truyền địa chỉ của biến có kiểu tương ứng.

  Vd: `void input (int a[], int*x);` Gọi hàm: `input (a,&x);`

- Con trỏ và mảng:

  +**Mảng một chiều**:

  - Đối với mảng một chiều khi sử dụng mảng nếu chỉ ghi tên mảng mà không ghi chỉ số thì có nghĩa là địa chỉ của phần tử thứ zero của mảng.(array <=> &array[0]).
  - Khi một hàm có tham số là mảng thì khi gọi hàm ta chỉ truyền tên của mảng, không phải ghi chỉ số mảng. Vd: input (**a** , n); nhập mảng A gồm n phần tử.

  +**Mảng nhiều chiều**:

  - Đối với mảng hai chiều khi sử dụng mảng nếu chỉ ghi tên mảng mà không ghi chỉ số thì có nghĩa là địa chỉ của hàng đầu tiên trong ma trận. (a ≡ &a[0] ≡ (&a[0][0]).
  - Không thể dùng phép toán địa chỉ để nhập cho mảng hai chiều (ngoại trừ kiểu int).

- Không thể dùng biến con trỏ để nhập cho mảng hai chiều.

Giải pháp cho trường hợp này là ta nhập thông qua một biến tạm sau đó gán lại biến tạm này cho cho phần tử tương ứng của mảng.

F. CHUỖI (STRING):

- Khai báo: tương tự như khai báo mảng ta cũng có hai cách khai báo là khai báo tường minh và không tường minh:

  char tenchuoi [chiều dài thực+1];

  char tenchuoi [ ];

- Các thao tác trên kiểu chuỗi:

  + Nhập:

  - Lệnh scanf("%s",tenchuoi)**;** Lệnh này không thể nhập vào một chuỗi có chứa ký tự khoảng trắng, tab, enter, và để lại ký tự "\n" trong bộ đệm bàn phím.
  - Lệnh gets(tenchuoi); Được khai báo trong "string.h". Lệnh này có thể nhập được chuỗi có khoảng trắng, kết thúc khi gặp ký tự xuống dòng("\0").

  + Xuất:

  - Lệnh printf("%s",tenchuoi);
  - Lệnh puts(tenchuoi); Xuất chuỗi và xuống dòng.

  + Gán chuỗi:

  Muốn sử dụng phép gán chuỗi ta phải sử dụng biến con trỏ và hàm calloc để cấp phát vùng nhớ cho biến này.

  Vd:
  ```
  char *str;
  str=calloc(40,sizeof(char));
  str="MediaSpace Club";...
  ```

- Các hàm thường dùng với kiểu chuỗi, ký tự: (phần lớn được khai báo trong "string.h").
  - **strcat(char*dest, char *scr)**: nối chuỗi dest với chuỗi scr;
  - **strspn(*str, *str_con)**: cho độ dài đoạn đầu tiên lớn nhất của str mà mọi ký tự của đoạn đều có mặt trong chuỗi str_con.
  - **strlen(char *str)**: cho chiều dài thực của chuỗi str.
  - **char *strupr(char *s):** chuyển chuỗi từ ký tự thường sang HOA.

- **char *strlwr(char *s):** chuyển chuỗi từ ký tự HOA sang thường.
- **char *strrev(char *s):** đảo ngược cả chuỗi trừ ký tự kết thúc("\0").
- **char *strcpy(char *dest, char *scr):** sao chép toàn bộ chuỗi src sang chuỗi dest.
- **char *strncpy(char *dest, char *scr,n):** sao n ký tự đầu của chuỗi src sang chuỗi dest.
- **char *strchr(char *str, int c):** tìm sự xuất hiện đầu tiên của c trong str. Nếu tìm thấy hàm trả về địa chỉ của ký tự tìm được trong str, ngược lại trả về NULL.
- **char *strstr(char *str, char *str_con):** tìm sự xuất hiện đầu tiên của str_con trong str. Nếu thấy, hàm cho địa chỉ của chuỗi con (trùng với str_con) trong str, ngược lại hàm trả về NULL.
- **char *strbrk(char *s1, char *s2):** duyệt trong s1 để tìm ký tự đầu tiên khớp với bất kỳ ký tự nào của s2. Nếu có xuất hiện, hàm cho địa chỉ của ký tự tìm thấy trong s1, ngược lại hàm trả về NULL.
- **int strcmp(char *s1, char *s2):** so sánh 2 chuỗi theo thứ tự, hàm trả về **< 0** nếu chuỗi s1 < chuỗi s2 ; **= 0** nếu chuỗi s1 = chuỗi s2 ; **>0** nếu chuỗi s1 lớn hơn chuỗi s2.
- **int stricmp(char *s1, char *s2)**: giống như **strcmp** nhưng không phân biệt chữ HOA, chữ thường khi so sánh.
- **int memcmp(char *s1, char *s2, unsighed n)**;
  So sánh chuỗi s1 và s2 một cách chính xác, phân biệt HOA thường trong phạm vi n byte và trả về: <0 nếu s1 nhỏ hơn s2; = 0 nếu s1 trùng s2 ; >0 nếu s1 lớn hơn s2.
- **char * strncat(char *s,char *s1,int n):** ghép n ký tự đầu tiên của s1 vào s.
- **char *itoa(int x,char *s,int cs):** Chuyển số nguyên x trong hệ đếm cơ số cs (cs=10: hệ 10, cs=8:hệ 8, cs=16:hệ 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng s . (stdlib.h)
- **int atoi(char *s)**: chuyển chuỗi s sang giá trị int.
- **long atol(char *s): c**huyển chuỗi s sang giá trị long.

G. CẤU TRÚC (STRUCTURE):

- Định nghĩa kiểu và khai báo biến tách biệt:

  + **Định nghĩa kiểu:**

  struct tên_cấu_trúc

  {

  kiểu tên_thành_phần 1;

  kiểu tên_thành_phần 2;...

  kiểu tên_thành_phần n;

  };

  + **Khai báo biến:**

  struct tên_cấu_trúc biến_1, biến_2,...,biến_n;

  + **Mảng cấu trúc**: **struct** tên_cấu_trúc tênmảng[spt];

  *chúng ta cũng có thể sử dụng **typedef** trước từ khoá **struct** để định nghĩa kiểu cấu trúc, khi đó muốn khai báo biến ta không cần đặt **struct** trước tên_cấu_trúc.*

- Định nghĩa kiểu và khai báo biến kết hợp:

  struct tên_cấu_trúc

  {

  các thành phần cấu trúc giống như trên;

  } biến_1,biến_2,...,biến_n;

- Truy xuất đến các thành phần cấu trúc:

  + Đơn giản: **biến_cấu_trúc.biến_thành_phần;**

  + Phức tạp: **biến_cấu_trúc.biến_cấu_trúc_2.biến_thành_phần;** được sử dụng khi biến cấu trúc 2 là thành phần trực tiếp của một biến câu trúc lớn hơn (Cấu trúc lồng nhau).

- Một số phép toán cơ bản trên kiểu cấu trúc:

  các thao tác trên biến cấu trúc phải được thực hiện thông qua các thành phần của nó (ngoại trừ phép gán). Thao tác trên các thành phần của cấu trúc cũng như thao tác trên các biến có kiểu tương ứng. **Vd:**

  + **Nhập**: `scanf("%s",&nhanvien.ten);`

  + **Xuất:** `printf("\n Ten nhan vien la: %s.",nhanvien.ten);`

  + **Cộng:** `luong=nhanvien1.luong + nhanvien2.luong;`

+ **Gán:** ta có thể gán hai biến cấu trúc cho nhau; gán biến cho phần tử mảng; gán phần tử mảng cho biến; gán hai phần tử mảng cho nhau. Điều cần lưu ý là các biến và các phần tử mảng phải cùng kiểu.

+ **Khởi tạo giá trị đầu cho cấu trúc:** ta khởi tạo giá trị bằng cách viết vào sau khai báo của chúng các giá trị của các thành phần.

Vd: `struct nhanvien2{"Nguyen Van A", 1990, 500000};`

- Con trỏ kiểu cấu trúc:

  + **Khai báo**: kiểu_cấu_trúc *tên_biến_con_trỏ;

  + **Truy xuất**: có 2 cách truy xuất đến thành phần của biến con trỏ:

    - Tên_biến_con_trỏ -> tên_thành_phần
    - (*Tên_biến_con_trỏ).tên_thành_phần

8. KIỂU TẬP TIN (FILE):

- Khai báo: FILE *fp; // Khai báo biến con trỏ kiểu File với tên fp
- Mã chuyển dòng và kết thúc File:
  - **Mã chuyển dòng**: Đối với File mở theo kiểu nhị phân mã chuyển dòng là mã 10 (LF), còn với kiểu văn bản sẽ là mã **13** và **10**.
  - **Mã kết thúc**: khi gặp mã kết thúc hàm feof(fp) cho giá trị <>0. Trong khi đọc nếu gặp ký tự có mã 26 hoặc ký tự cuối cùng trong File, ta sẽ nhận được mà kết thúc File **EOF** (số **-1**).
- Đóng mở File và đọc ghi dữ liệu:
  - **Mở File**: **FILE *fopen(const char *tênfile, const char*kiểu);** Hàm dùng để mở một File, nếu thành công trả về con trỏ kiểu File ứng với File vừa mở, ngược lại trả về NULL. Trong đó đối số **kiểu** mang các giá trị sau:

    + Kiểu văn bản: "r"(read), "w"(write), "a"(append), "r+", "w+" . "a+".

    + Kiểu nhị phân: "rb", "wb", "ab", "r+b", "w+b", "a+b". (*Ctrl+F1->detail*).
  - **Đóng File: int fclose(FILE *fp);** Với fp là con trỏ tương ứng với File cần đóng. Để đóng tất cả các File đang mở ta dùng hàm: **int fcloseall(void)**; Nếu thành công 2 hàm này cho giá trị <> **0**, ngược lại trả về **EOF**.
  - **Ghi ký tự: int putc(int ch, FILE *fp)**; **int fputc(int ch, FILE *fp)**; Với **ch** là giá trị nguyên, **fp** là con trỏ File. Hàm này sẽ ghi vào file một ký tự có mã =

ch%256. Nếu thành công trã về mã ký tự được ghi, ngược lại trả về **EOF**. Làm việc với Text&Binary.

- **Đọc ký tự từ File: int getc(FILE *fp); int fgetc(FILE *fp)**; Hàm đọc một ký tự từ File fp, nếu thành công hàm cho mã đọc được (0 ... 255), ngược lại trả về **EOF**.
- **Ghi dữ liệu theo khuôn dạng: int fprintf(FILE *fp, const char *control, ...)**; Với fp là con trỏ File, **control** chứa địa chỉ của chuỗi điều khiển (giống như trong hàm printf), **...** là danh sách các đối mà giá trị của chúng cần ghi vào File.
  VD: `fprintf(fp,"\n this year: %d",iYear);`
- **Đọc dữ liệu theo khuôn dạng: int scanf(FILE *fp, const char *control,...);** Với fp là con trỏ File, **control** chứa địa chỉ của chuỗi điều khiển (giống như trong hàm printf), **...** là danh sách các đối chứa kết quả đọc được từ File.
  VD: `fscanf(fp,"%d%f",&ia,&fa);`
- **Ghi chuỗi ký tự: int fputs(const char *s, FILE *fp);** Với s là con trỏ chỉ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng '\0'. Hàm sẽ ghi chuỗi s lên File fp, nếu thành công trả về ký tự cuối cùng được ghi vào File, ngược lại trả về EOF.
- **Đọc chuỗi (dãy) ký tự từ File: char *fgets(char *s, int n, FILE *fp);** Với s là con trỏ kiểu char trỏ tới vùng nhớ để chứa chuỗi đọc từ File, n là số nguyên xác định độ dài cực đại của dãy cần đọc, fp là con trỏ File. Hàm sẽ đọc một dãy ký tự từ File fp vào vùng nhớ s, nếu thành công hàm trả về địa chỉ vùng nhận kết quả, ngược lại trà về NULL. **VD:** `while(!eof(fp)) fgets(s,256,fp);`
- **Ghi một số nguyên theo kiểu nhị phân: int putw(int n, FILE *fp);** Với n là giá trị nguyên, fp là con trỏ File. Nếu thành công hàm trả về số nguyên được ghi, ngược lại trả về EOF.
- **Đọc một số nguyên theo kiểu nhị phân: int getw(FILE *fp);** Với fp là con trỏ File. Hàm đọc một số nguyên(2 byte), nếu thành công trả về số nguyên đọc được, khi có lỗi hoặc cuối File hàm trả về EOF.
- **Ghi các mẫu tin lên File: int fwrite(void *pt, int size, int n, File *fp);** Với **pt** là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi, **size** là kích thước của mẫu tin (byte), **n** là số mẫu tin, **fp** là con trỏ File. **Hàm** sẽ ghi n mẫu tin kích thước size

byte từ vùng nhớ pt lên File fp. Hàm trả về một giá trị = số mẫu tin thực sự được ghi.

- **Đọc các mẫu tin từ File: int fread(void *pt, int size, int n, File *fp);** Với **pt** là con trỏ trỏ tới vùng nhớ chứa dữ liệu đọc được, **size** là kích thước của mẫu tin (byte), **n** là số mẫu tin cần đọc, **fp** là con trỏ File. **Hàm** sẽ đọc n mẫu tin kích thước size byte từ File fp vào vùng nhớ pt. Hàm trả về một giá trị = số mẫu tin thực sự đọc được. Các hàm fread, fwrite thường dùng để đọc ghi các đối tượng có cùng độ lớn như cấu trúc, số thực,...

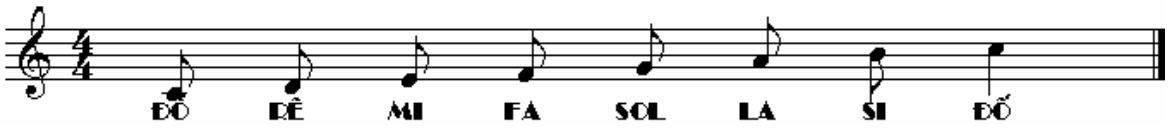
- Một số hàm khác thao tác trên FILE: (cấp II)
  - **int fflush(FILE *fp);** làm sạch vùng đệm File fp, thành công ->0, ngược lại ->EOF.
  - **int fflushall(void);** làm sạch vùng đệm của các File đang mở, nếu thành công hàm trả về giá trị số tệp đang mở, ngược lại trả vè EOF.
  - **int ferror(FILE *fp);** dùng kiểm tra lỗi thao tác trên File fp. Hàm trả về giá trị <>0 nếu có lỗi, ngược lại trả về 0.
  - **void perror(const char *s);** in chuỗi s và thông bào lỗi hệ thống.
  - **int unlink(const char *filename);** xoá một File trên đĩa, nếu thành công hàm cho giá trị 0, ngược lại trả về EOF.
  - **void rewind(FILE *fp);** chuyển con trỏ chỉ vị của File fp về đầu File.
  - **int fseek(FILE *fp, long sb, int xp);** di chuyển con trỏ chi vị của File fp từ vị trí xác định xp qua |sb| byte. Chiều di chuyển là về cuối tệp nếu sb>0 và cuối tệp nếu sb<0. Khi thành công hàm trả về 0, ngược lại hàm trả về giá trị <>0. Không nên dùng trên kiểu văn bản vì sự chuyển vị sẽ thiếu chính xác. **xp** cho biết vị trí xuất phát, có thể nhận các giá trị: **0** (xuất phát từ đầu tệp), **1** (từ vị trí hiện tại của con trỏ chuyển vị), **2** (xuất phát từ cuối tệp).
  - **long ftell(FILE *fp);** trả về vị trí hiện tại của con trỏ chỉ vị (byte) được tính từ 0. Nếu có lỗi hàm trả về -1L.

9. MÀN HÌNH VĂN BẢN VÀ ÂM THANH:

- Màn hình văn bản:

- **void textmode(int mode)**; chọn mode màn hình văn bản, **mode** bao gồm LASTMODE, BW40, C40, BW80, C80, MONO. (conio.h). Các giá trị của **mode** được mô tả trong help online với index là **text_modes.**
- **void textbackground(int color)**; đặt màu cho nền (cửa sổ).
- **void textcolor(int color);** đặt màu chữ. Với **color** mang giá trị từ 0-15 hay các hằng số định nghĩa sẵn như: BLACK, WHILE, BLUE, RED, CYAN, YELLOW, BROWN,.. Các giá trị của **color** được trình bày trong help online với index là **COLORS.**
- **void window(int xtr, int ytr,int xd,int yd);** thiết lập một cửa sổ với góc trên trái (xt, yt) và dưới phải (xd, yd). Chú ý rằng nên dùng hàm cprintf và scanf thay cho printf và scanf sẽ cho một màn hình đẹp hơn.
- **void clrscr(void);** xoá cửa sổ hiện tại, đưa con trỏ về góc trên trái của cửa sổ.
- **void clreol(void);** xoá đến cuối dòng trong cử sỗ.
- **void gotoxy(int x, int y);** chuuển con trỏ đến vị trí x,y trong cửa sổ
- **int wherex(void), wherey(void);** cho biết vị trí ngang, dọc của con trỏ trong cửa sổ.

- Âm thanh:
  - **sound(n)**: Dùng loa CPU phát ra một âm thanh với tần số n.
  - **delay(m)**: Kéo dài một nốt nhạc trong thời gian là m mili giây.
  - **nosound()**: Tắt âm thanh đã phát.



Dùng các hàm trên chúng ta sẽ phát ra các âm thanh là các note nhạc với cao độ là n và trường độ m. Sau đây là 7 tần số tương ứng với 7 note nhạc cơ bản ờ quãng tám thứ I:

- Đô: 132
- MI: 164
- SOL: 192
- SI: 244
- Rê: 146
- FA: 172
- LA: 216

Đề được note thăng ta lấy (note thường tương ứng cộng với note cao hơn kế tiếp)/2, và note giáng thì cũng lấy (note thường tương ứng cộng với note thấp hơn kế tiếp)/2.Đối với các note ở các quảng tám cao hơn hay thấp hơn thì ta chỉ việc nhân hay chia đôi các note tương ứng. Các note lặng ta sẽ phát một note với tần số khoảng 30.000Hz (Siêu âm).

VD: Rê#=(Rê+Mi)/2; Rêb(Rê giáng)=(Rê+Đô)/2; Đố=Đô*2;...

Về trường độ ta lấy note Đen làm chuẩn, và qui định trường độ của các note khác (note móc, nốt tròn, đen chấm, tròn chấm,...) theo qui tắc nhạc lý và lấy theo chuẩn của note Đen. Thông thường với tiết tấu trung bình thì note Đen được qui định khoảng từ 300-400, tuỳ theo tiết tấu nhanh hay chậm của mỗi bài hát cụ thể mà ta qui định cho hợp lý.

10. ĐỒ HOA:

- Khởi động chế độ Đồ hoạ:

void initgraph(int *graphdriver, int *graphmode, char *driverpath); Với driverpath là đường dẫn chứa các file điều khiển đồ hoạ (thường là "C:\\BorlandC\\Bgi"), graphdriver và graphmode dùng xác định màn hình và kiểu đồ hoạ sẽ sử dụng. Nếu không biết kiểu màn hình đang sử ta sử dụng DETECT cho graphdriver để chương trình tự nhận dạng, Các giá trị của graphriver được trình bày trong help online với index là **graphics_rivers** và **graphics_modes**.

VD:
```
#include <stdio.h>;
#include <conio.h>;
#include <graphics.h>;
main()
{
  int gr_drive = DETECT, gr_mode;
  initgraph(&gr_drive, &gr_mode, "c:\\tc\\bgi");
  outtext("Graphic example");
  getch();
}
```

- Các hàm thường dùng:

- **Lỗi đồ hoạ**: phát sinh khi khởi động chế độ dồ hoạ và khi dùng các hàm. Hàm **int graphresult(void)** trả về lỗi khi phát hiện, hàm **char grapherrormsg(int errorcode)** trả về thông bào lỗi do graphresult phát hiện. Các mã lỗi của graphresult được trình bày trong help online với index là **graphics_errors.**
- **Chọn màu nền**: **void setbkcolor(int color)**;
- **Chọn màu nét: void setcolor(int color)**;

- **Chọn mẫu và màu tô**: **void setfillstyle(int pattern, int color);** Các giá trị của **pattern**(mẫu tô) được trình bày trong help online với index là **fill_parttens**.
- **Chọn mẫu cho nét**: **void setlinestyle(int linestyle, int pattern, int thickness**); cho phép ta qui định 3 yếu tố là kiểu linestyle(0[nét liền],1[nét chấm], 2[nét chấm gạch], 3[nét gạch], 4[mẫu tự tạo]) bề dày thickness(1[bình thường[, 3 [dày gấp ba] và mẫu tự tạo patern (nếu qui định linestyle là 4). Để nhận giá trị hiện hành cua 3 yếu tố trên ta dùng hàm **void getlinesettingstyle(struct linesettingtype, * linenifo);** , Cách định nghĩa mẫu tự tạo và cấu trúc của **linesettingtype** được trình bày trong help online với index là **setlinestyle** và **linesettingstyle.**
- **Thay đổi giải màu mặc định: void setpalette(int colornum, incolor);**
- **Xác nhận giải màu hiện hành**: **void palette(struct palettetype *palette**); Cấu trúc của palettetype được trình bày trong help online với index là **palettetype**.
- **Các hàm getcolor(), getbkcolor() và getmaxcolor()** trả về màu đã xác định với setcolor, setbkcolor, và số lượng màu cực đại thuộc dãy màu hiện hành.
- **Vẽ cung tròn: void arc(int x, int y, int gocdau, int goccuoi, int bankinh); G**óc: độ.
- **Vẽ đường tròn: void circle(int x, int y,int bankinh);**
- **Vẽ cung ellipse: void ellipse(intx, int y, int gocdau, int goccuoi, int xr, int yr**); Với xr là bán trục ngang và yr là bán trục đứng.
- **Vẽ và tô hình quạt: void pieslice(int x, int y, int gocdau, int goccuoi, int bankinh**);
- **Vẽ đường gấp khúc và đa giác: drawpoly(n,a)**; Với n là số điểm, a là mảng kiểu int chứa toạ độ của các điểm(a[0]=x1, a[1]=y1, a[2]=x2, a[3]=y2,...). Nếu nếu điểm cuối **xn, yn** trùng với điểm đầu **x1, y1** ta sẽ được một **đa giác**.
- **Tô màu đa giác**: **fillpoly(n,a)**; hàm này sẽ tô màu 1 đa giác có các n đỉnh.
- **Vẽ đường thẳng: void line(int x1, int y1, int x2, int y2);** Vẽ đường thẳng nối hai điểm (x1,y1) và (x2,y2) nhưng không làm thay đổi vị trí con trỏ. Hàm **void lineto(int x, int y);** Vẽ đường thẳng từ điểm hiện tại tới (x,y) và chuyển con trỏ đến điểm (x,y). Hàm **void linerel(int dx, int dy);** sẽ vẽ một đường thẳng từ vị trí hiện

hành (x,y) của con trỏ đến điểm (x+dx, y+dy), con trỏ được di chuyển đến vị trí mới.

- **Di chuyển con trỏ tới toạ độ (x,y)**: **void moveto(intx, inty)**;
- **Vẽ hình chữ nhật. Hàm void rectangle(int x1, int y1, int x2, int y20;** sẽ vẽ một đường chữ nhật có các // các cạnh của màn hình, (x1,y1) là toạ độ đỉnh trên, (x2,y2) là toạ độ đỉnh dưới. Hàm **void bar(intx1, int y1, int x2, int y2);** sẽ vẽ và tô màu một hình CN. Hàm **void bar3d(int x1, int y1, int x2, int y2, int depth, int top);** vẽ và tô một khối hộp chữ nhật, **depth** qui định số điểm ảnh trên chiều sâu của khối 3D, **top** có thể nhận các giá trị là 1(có nắp), 0(không nắp).
- **Thiết lập cửa sổ viewport: void setviewport(int x1, int y1, int x2, int y2, int clip);** Với 2 điểm là 2 góc trên trái và dưới phải của cửa sổ, clip=1 không cho phép vẽ ra ngoài viewport và clip=0 thì cho phép. Để nhận viewport hiện hành ta dùng hàm void getviewsettings(struct viewporttype *vp). (Help online với **index** là **viewporttype).**
- **Xoá viewport: void clearviewport(void);**
- **Xoá màn hình và đưa con trỏ về (0,0): void cleardevice(void);**
- **Tô và lấy màu tô Điểm.** Tô**: void putpixel(int x, inty, int color); Hàm getpixel(int x, int y);** sẽ trã về giá trị màu tại điểm (x,y), nếu điểm chưa được tô hàm sẽ trả về 0.
- **Tô miền: void floodfill(int x, int y, int border);** border chứa giá trị của một màu.
- **Hiển thị văn bản.** Hàm **void outtext (char *s);** sẽ hiện thị chuổi do s trỏ tới tại vị trí hiện tại của con trỏ. Hàm outtextxy(int x, inty, char *s); sẽ hiển thị VB ở toạ độ (x,y).
- **Chọn Font (.CHR): void settextstyle(int font, int direction, int charsize);** Với **direction** =0(ngang), =1(dọc). **Charsize** có giá trị từ 1(8*8pixel)->10(80*80pixel). f**ont** nhận các giá trị 0(default), 1(triplex), 2(small), 3(sanserif), 4(gothic).
- **Vị trí hiển thị Text: void settextjustify(int horiz, int vert);** Với **horiz = 0**(VB ở bên phải con trỏ), 1(tâm VB theo vị trí con trỏ), 2(bên trái con trỏ). **Vert =** 0(VB ở bên phải con trỏ), 1(tâm VB theo vtr của con trỏ), 2(phía dưới con trỏ).
- **Lấy chiều cao và rộng của Text: int textheight(char*s); int textwidth(char *s);**

- **Hàm imagesize(int x1, int y1, int x2, int y2);**trả về số byte dùng lưu ảnh trong 1 phạm vi.
- **Hàm void getimage(int x1, int y1, int x2, int y2, void *bitmap);** dùng lưu các điểm ảnh trong phạm vi HCN vào vùng nhớ do bitmap trỏ tới. Vùng nhớ và biến bitmap cho bởi hàm **void *malloc(unsigned n)**;
- Hàm putimage(int x, int y, void *bitmap, int copymode); dùng sao chép ảnh trong vùng nhớ bitmap ra màn hình tại vị trí (x,y). Tham số copymode =0(chép nguyên xi), =1(các điểm ảnh trong bitmap kết hợp với các điểm trên màn hình = phép XOR), =2(...”.... =phép or), =3(.....”.....”and), =4(....’’.....NOT).

11. KỸ THUẬT SỬA LỖI VÀ RỞ RỐI CHƯƠNG TRÌNH:

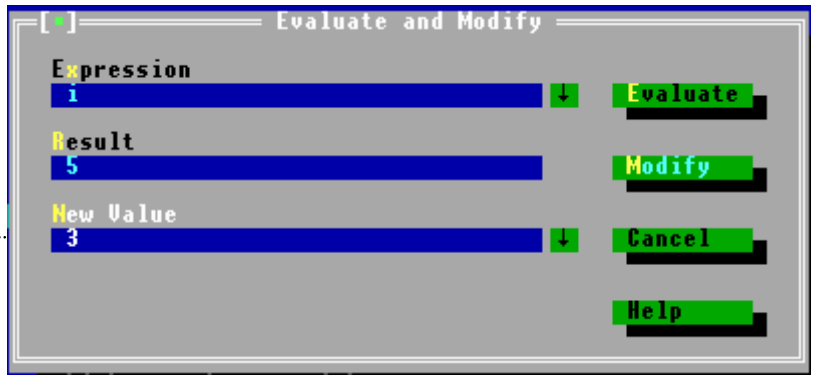
+ Người ta nghiên cứu và cho rằng khoảng 80% thời gian của các lập trình viên là dùng để kiểm tra và sửa chữa những chương trình đã có. Thông thường có 3 loại lỗi sau: lỗi cú pháp (syntax), lỗi run-time (chia cho zero, gán sai kiểu, sử dụng biến không khai báo, ...), và lỗi logic. Lỗi logic là lỗi khó sửa nhất, vì đây là lỗi sai trong thuật giải chương trình.

+ Phương pháp duy nhất để sữa các lỗi cú pháp là bạn phải biết chương trình của mình mắc phải lỗi gì, điều này được Borland C++ thông báo trong cửa sổ Message, và từ đây bạn có thể tra cứu các phương pháp sữa chữa trong các sách tra cứu, đặt biệt là trong help online của Borland C++.

+ Khi gặp các lỗi Logic làm cho chương trình chạy không đúng hay cho kết quả sai ta có thể sử dụng các chức năng Debug của Borland C để tiến hành rở rối:

-Chạy từng bước: dùng phím F7, với mỗi lần nhấn F7 là dòng được làm sáng sẽ được thực hiện. Nhờ chạy từng bước như vậy ta sẽ dễ dàng xử lý được các lỗi Logic trong chương trình. Có thể thay phím F7 bằng F8. Sự khác nhau duy nhất giữa hai phím này là là F7 sẽ thực hiện từng dòng bên trong hàm được gọi, còn F8 thì sẽ thực hiện hàm này và bước qua nó.

-Kết hợp với chạy từng dòng là ta sử dụng cửa sổ EVALUATION (nhấn Ctrl+F4), với cửa sổ này ta



có thể quan sát giá trị của các biến, mảng,... qua mỗi dòng thực chương trình (gõ vào tên biến, mảng,.. cần xem trong khung Evaluate và xem giá trị trong khung result), hay làm thay đổi giá trị các biến, tính toán các biểu thức, các hàm...

-Dùng cửa sổ Watch (Window\Watch) để theo dõi sự thay đổi giá trị của các biến, mảng. Để xem các biến trong cửa sổ Watch, ta chọn Debug\Watch\Add Watch và gõ tên biến cần xem vào, nhấn F7 hay F8 để chạy từng bước chương trình và xem gía trị của biến. Có thể nhập vào nhiều biến, mỗi lần chỉ được nhập một biến. Để xoá một biến ta chỉ di chuyển đến biến muốn xoá trong cửa sổ Watch và nhấn Delete. Nếu không thấy cửa sổ Watch khi chạy từng bước chương trình, ta nhấn F5.

-Dùng cửa sổ User Screen để quan sát kết quả thực hiện của chương trình: Window\ User Screen hay Alt+F5.

Ngoài ra còn nhiều kỹ thuật khác như sử dụng điểm gãy, inspect, call stack,... để gở rối các hàm, chương trình. Các bạn có thể xem chi tiết trong các sách tham khảo về Lập trình C.

Lưu ý:

HTD - TH0003 (5.2001).

Se (9.2001).