Ayuda para la programación en C

Estructura de un programa C

```
/*
Programa de Ejemplo
Fecha_
Autor_
*/
#include ____
#define ____
typedef ____
[Prototipos]

int main(void)
{
   [variables] /* descripcion */

   [instrucciones]
   return 0;
}
```

Caracteres especiales

`'\n'` cambio de línea (newline)
`'\r'` retorno de carro
`'\0'` caracter 0 (`NULL`)
`'\t'` TAB
`'\''` comilla simple '
`'\"'` comilla doble "
`'\\'` la barra \

Formatos de `printf` y `scanf`

`%d` int
`%hd` short
`%ld` long
`%u` unsigned int
`%hu` unsigned short
`%lu` unsigned long
`%f` float, double
`%lf` double (sólo `scanf`)
`%c` char
`%s` cadena de caracteres

Operadores

Aritméticos `int`: `+ - * / %`
Aritméticos `double`: `+ - * /`
Otros aritméticos: `++ -- += -= *= /=`
Lógicos y relacionales:
`> < >= <= == != && || !`

Bucles

Bucle `for`

```
for(inicialización, condición, instrucción_final)
    {
   [instrucciones]
}
```

Ejemplo: `for(i=0; i<10; i++)`

Bucle `while`

```
while (condición) {
   [instrucciones]
}
```

Bucle `do-while`

```
do  {
   [instrucciones]
} while(condición);
```

Bloque if

caso 1:

```
if (condición)  {
   [instrucciones]
}
```

caso 2:

```
if (condición)  {
   [instrucciones_1]
} else  {
   [instrucciones_2]
}
```

caso 3:

```
if (condición_1)  {
   [instrucciones_1]
} else if (condición_2) {
   [instrucciones_2]
...
} else if (condición_n) {
   [instrucciones_n]
} else  {
   [instrucciones]
}
```

Sintaxis del switch

```
switch(expresión_entera)  {
case constante_1:
   [instrucciones_1]
   break;
case constante_2:
   [instrucciones_2]
   break;
...
case constante_3:
   [instrucciones_3]
   break;
default:
   [instrucciones]
}
```

Vectores y matrices

```
double vector[10];
char cadena[256];
char matriz[10][20];

vector[2]=3;
scanf("%lf",&vector[7]);
```

Cadenas de caracteres

```
char cadena[N];
```

Lectura:

`scanf("%s",cadena);`
lee una palabra

`gets(cadena);`
lee una frase hasta fin de linea

`fgets(cadena, N, stdin);`
lee una frase con control de tamaño. También lee `\n`

Escritura:

`printf("%s",cadena);`
escribe una cadena por pantalla, vale para frase o palabra

Funciones estandar de string.h

`size_t strlen( char *str );`
devuelve la longitud de la cadena

`strcpy( char *to, char *from );`
copia o inicializa

`int strcmp(char *s1, char *s2 );`
compara las cadenas `s1` y `s2`
`0` → `s1` es igual a `s2`
`<0` → `s1` es menor que `s2`
`>0` → `s1` es mayor que `s2`

Funciones

Prototipo:
```
tipo NombreFun(tipo var1, ... , tipo varN);
```

Estructura de la función:
```
tipo NombreFun(tipo var1, ... , tipo varN)
/* Descripción general
Argumentos: ...
Valor Retornado: ...
Advertencias de uso: ...
*/
{
   [variables locales]

   [instrucciones]

   return expresión;
}
```

Ejemplos de prototipos y llamadas:
```
int Sumar(int a, int b);
void Cambio(int *a, int *b);
double CalcularMedia(double a[], int n);
float Traza(float mat[][20], int n, int m);

res=Sumar(x,y);
Cambio(&x, &y);
med=CalcularMedia(vec,n);
tra=Traza(mat,n,m);
```

Asignación Dinámica de Memoria
```
char *pc;
pc=(char *)calloc(100, sizeof(char);
pc=(char *)malloc(100*sizeof(char));
pc=(char *)realloc(pc, 200*sizeof(char));
free(pc);  /*libera memoria */
```
Estas funciones devuelven NULL en caso de error

Estructuras

Declaración de un tipo estructura
```
typedef struct persona {
   char nombre[N];
   int edad;
   long dni;
} PERSONA;
```

Declaración de variables:
```
PERSONA p;   /* una estructura */
PERSONA *pp; /* puntero a estructuras */
PERSONA vec[20]; /* vector de estructuras */
```

Acceso a los miembros:
```
p.edad=27;
pp->edad=30;
vec[7].edad=37;
```

Declaración de listas enlazadas:
```
typedef struct lista {
   char nombre[N];
   int edad;
   long dni;
   struct lista *siguiente;
} LISTA;
```

Archivos

Abrir y cerrar
```
FILE *fopen(char *nombre, char *modo);
```
Devuelve NULL en caso de error
modo="r" Lectura
modo="r+" Lectura (y escritura)
modo="w" Escritura
modo="w+" Escritura (y lectura)
modo="a" Añadir al final
modo="a+" Añadir al final (y lectura)
modos=rb, rb+, wb, wb+, ab, ab+ binario
```
int fclose(FILE *puntero al archivo);
```
Devuelve 0 si no hay error

Archivos de texto
```
int fscanf(FILE *fp, char *cadena_formato, ...);
```
Devuelve el número de variables leídas
Devuelve 0 si no hay podido leer ninguna variable
Devuelve EOF si ha llegado al final de fichero
```
int fprintf(FILE *fp, char *cadena_formato, ...);
char *fgets(char *cadena, int tam_cad, FILE *fp);
```
Devuelve el puntero a la cadena si no hay error
Devuelve NULL en caso de error
```
int fputs(char *cadena, FILE *fp);
```

Archivos binarios (acceso directo)
```
int fwrite(void *variable, size_t tamaNo, size_t num, FILE *fp);
int fread (void *variable, size_t tamaNo, size_t num, FILE *fp);
```
Devuelve el número de elementos leídos, normalmente `num`

```
int fseek(FILE *fp, long desplazamiento, int origen);
```
El tercer argumento puede tomar los valores: `SEEK_SET` (comienzo), `SEEK_END` (final), `SEEK_CUR` (actual)

Otras Funciones generales
```
int fgetc(FILE *fp);
```
Devuelve el caracter leído (lo devuelve como `int`)
Devuelve EOF si ha llegado al final de fichero

```
int fputc(int caracter, FILE *fp);
int feof( FILE *fp );
```
Devuelve distinto de cero si estamos al final del fichero. En caso contrario, devuelve cero

```
void rewind( FILE *fp );
```
Vuelve al principio del archivo. Equivale a `fseek(fp,0,SEEK_SET);`