Clase 13. Funciones con número variable de argumentos. Librerias estándar. Achivos binarios.

Funciones con número variable de parámetros

Funciones con número variable de parámetros

Para la declaración-definición de una función con una lista variable de argumentos se utilizan los 3 puntos.

```
1  int funcion(int n,...);

3  int funcion(int n,...)
   {
5  va_list argumentos;
   return 0;
7  }
```

- **Debe de haber al menos un argumento con nombre primero en la función.**

Funciones con número variable de parámetros

Para la declaración-definición de una función con una lista variable de argumentos se utilizan los 3 puntos.

```
1  int funcion(int n,...);

3  int funcion(int n,...)
   {
5  va_list argumentos;
   return 0;
7  }
```

- **Debe de haber al menos un argumento con nombre primero en la función.**
- **No existe forma de saber cuantos argumentos se pasaron ni de que tipo..**

va_list


```
#include <stdarg.h>
int fun(int n,...){
  va_list arg;
  va_start(arg, n);
   for (int i=0; i<n; i++)
        printf("%lf\n",va_arg(arg,double));
   va_end(arg);
 return 0;
}
int main(int argc, char *argv[]){
   fun(5,3.5,4.6,7.9,-6.4,8.7);
   return 0;
}
```

- **va_list** la implementación de la variable es dependiente del compilador y sistema. Sirve para almacenar una lista variable de argumentos. Solo debe usarse en funciones del estándar.

va_list


```
#include <stdarg.h>
int fun(int n,...){
 va_list arg;
 va_start(arg, n);
  for (int i=0; i<n; i++)
      printf("%lf\n",va_arg(arg,double));
  va_end(arg);
 return 0;
}
int main(int argc, char *argv[]){
  fun(5,3.5,4.6,7.9,-6.4,8.7);
  return 0;
}
```

- **va_list** la implementación de la variable es dependiente del compilador y sistema. Sirve para almacenar una lista variable de argumentos. Solo debe usarse en funciones del estándar.
- **void va_start (va_list ap, paramN);** ap, es una variable de tipo va_list, y paramN es el último parámetro con nombre en la llamada a la función.

va_list


```
#include <stdarg.h>
int fun(int n,...){
 va_list arg;
 va_start(arg, n);
  for (int i=0; i<n; i++)
      printf("%lf\n",va_arg(arg,double));
  va_end(arg);
 return 0;
}
int main(int argc, char *argv[]){
  fun(5,3.5,4.6,7.9,-6.4,8.7);
  return 0;
}
```

- **va_list** la implementación de la variable es dependiente del compilador y sistema. Sirve para almacenar una lista variable de argumentos. Solo debe usarse en funciones del estándar.
- **void va_start (va_list ap, paramN);** ap, es una variable de tipo va_list, y paramN es el último parámetro con nombre en la llamada a la función.
- **void va_end (va_list ap);** Ayuda a que la función regrese de forma normal. **Debe ser llamada después de va_start.**

va_list


```c
#include <stdarg.h>
int fun(int n,...){
 va_list arg;
 va_start(arg, n);
  for (int i=0; i<n; i++)
      printf("%lf\n",va_arg(arg,double));
  va_end(arg);
 return 0;
}
int main(int argc, char *argv[]){
  fun(5,3.5,4.6,7.9,-6.4,8.7);
  return 0;
}
```

Salida:

```
 3.500000
4.600000
7.900000
-6.400000
8.700000
```

Librerias Estándar: stdio.h

stdio.h, variables

Variable	Description
size_t	En general es un entero sin signo y puede almacenar el tamaño del objeto mas grande posible que puede ser reservado.
FILE	Estructura para almacenar información acerca de un archivo. La definición de la estructura depende de la implementación del compilador y del sistema operativo.
fpos_t	Estructura para almacenar una posición en un archivo.


```
typedef struct
{
 short level ;
 short token ;
 short bsize ;
 char fd ;
 unsigned flags ;
 unsigned char hold ;
 unsigned char *buffer ; //Buffer usado para transferir
     datos
 unsigned char * curp ;
 unsigned istemp;
}FILE ;
```

stdio.h, constantes (macros)

- NULL El valor de un apuntador constante que no apunta a un lugar valido de la RAM, usualmente (void *)0.
- _IOFBF, _IOLBF y _IONBF Macros que indican el comportamiento de un buffer definido por el programador.
- BUFSIZE Macro que indica el tamaño del buffer.
- EOFM Entero negativo que indica el fin de un archivo.
- FOPEN_MAX Indica el número máximo de archivos que el sistema garantiza que se pueden abrir simultaneamente.
- FILENAME_MAX Macro que es un entero que representa el tamaño maximo de un arreglo que pueda almacenar el nombre de tamaño máximo de un archivo.
- L_tmpnam Macro que es un entero con el tamaño maximo del nombre de un archivo temporal creado por tmpnam.
- SEEK_CUR, SEEK_END, and SEEK_SET Macros utilizados por fseek para buscar diferentes posiciones en un archivo.
- TMP_MAX El número máximo de nombres de archivos únicos que tmpnam puede generar.
- stderr,stdin y stdout Macros apuntadores a archivos, que apuntan a la salida de error estándar, salida estándar y entrada estándar.

stream

Un *stream* es un objeto (estructura) de alto nivel que representa un canal de comunicación entre el programa y un archivo, dispositivo u otro proceso.

stdio.h, funciones vistas

- **FILE *fopen(const char *filename, const char *mode)**.Abre archivo de texto.
- **int fclose(FILE *stream)**. Cierra archivo de texto.
- **int fprintf(FILE *stream, const char *format, ...)**. Imprime a archivo.
- **int printf(const char *format, ...)**. Imprime a la salida estándar.
- **int fscanf(FILE *stream, const char *format, ...)**. Lee de archivo.
- **int scanf(const char *format, ...)**. Lee de la entrada estándar.
- **int getchar(void)**. Leé un char de la entrada estándar.
- **char *gets(char *str)**. Leé una línea de la entrada estándar.
- **int putc(int char, FILE *stream)**. Imprime un caracter a un archivo.
- **int putchar(int char)**. Imprime un caracter a la salida estándar.

- **int puts(const char *str)**. Imprime una cadena a la salida estándar.

stdio.h, Otras funciones de lectura/escritura

Ejemplos después de esta diapositiva:

- **int sprintf(char *str, const char *format, ...)**. Imprime a str de la misma forma que printf a stdout.
- **int vprintf(const char *format, va_list arg)**. Imprime a la salida estándar una lista de argumentos.
- **int vfprintf(FILE *stream, const char *format, va_list arg)**. Imprime a un archivo una lista de argumentos.
- **int vsprintf(char *str, const char *format, va_list arg)**. Imprime a una cadena una lista de argumentos.
- **int sscanf(const char *str, const char *format, ...)**. Lee desde una cadena.
- **int fgetc(FILE *stream)**. Obtiene un char de un archivo.
- **char *fgets(char *str, int n, FILE *stream)**. Obtieve un string de un archivo.
- **int fputc(int char, FILE *stream)**. Imprime un caracter a un archivo.
- **int fputs(const char *str, FILE *stream)**. Imprime una cadena a un archivo.
- **int getc(FILE *stream)**. Obtiene un caracter de un archivo.
- **int fputc(int char, FILE *stream)**. Caracter a stream.
- **int ungetc(int char, FILE *stream)**. Caracter a stream, lo

sprintf


```c
#include <stdio.h>
int main(int argc, char *argv[])
{
  char str[1024];
  sprintf(str,"Escribo sobre cadena:%lf \n",5.8);
  printf("%s",str);
  return 0;
}
```

Salida:

```
Escribo sobre cadena:5.800000
```

vprintf


```c
#include <stdio.h>
#include <stdarg.h>
int fun(const char *format,...){
 va_list arg;
 va_start(arg, format);
  vprintf(format,arg);
  va_end(arg);
 return 0;
}
int main(int argc, char *argv[]){
  fun("%lf %lf\n", 5.3, 8.5);
  return 0;
}
```

Salida:

```
5.300000 8.500000
```

sscanf


```c
#include <stdio.h>
#include <stdarg.h>
int main(int argc, char *argv[]){
  char str[]={"dato= 128 \n"};
  char texto[6];
  int dato;
  sscanf(str,"%s %d",texto, &dato);
  printf("dato=%d\n",dato);
  return 0;
}
```

Salida:

```
dato=128
```

fgetc, fgets

El archivo tiene: Hola mundo!


```c
#include <stdio.h>
#include <stdarg.h>
int main(int argc, char *argv[]){
  FILE *in=fopen("archivo.txt","r");
  char str[128];
  char c=fgetc(in);
  printf("Primer caracter del archivo: %c\n",c);
  c=fgetc(in);
  printf("Segundo caracter del archivo: %c\n",c);
  fgets(str,128,in);
  printf("%s\n",str);
  fclose(in);
  return 0;
}
```

Salida:

```
Primer caracter del archivo: H
Segundo caracter del archivo: o
la mundo
```

stdio.h, funciones

- **int feof(FILE *stream)**. Regresa positivo si se llegó al final del archivo.
- **int fflush(FILE *stream)**. Vacía los datos del buffer al archivo.
- **int ferror(FILE *stream)**. Devuelve diferente de 0 si hubo un error en la operación anterior que se haya realizado con *stream*.
- **void perror(const char *str)**. Imprime el error del archivo.
- **void clearerr(FILE *stream)**. Limpia el error o el eof producido por un stream .
- **int fseek(FILE *stream, long int offset, int whence)**. Posiciona el stream a una distancia *offset* de la referencia *whence*.
- **int fgetpos(FILE *stream, fpos_t *pos)**.
- **int fsetpos(FILE *stream, const fpos_t *pos)**.
- **size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)**.
- **FILE *freopen(const char *filename, const char *mode, FILE *stream)**.

fseek, fsetpos, fgetpos

El archivo tiene: Hola mundo!


```c
#include <stdio.h>
int main(int argc, char *argv[]){
 FILE *in=fopen("archivo.txt","r");
 char str[128];  fpos_t pos;
 fgetpos(in, &pos);
 char c=fgetc(in);  printf("Primer caracter: %c\n",c);
 c=fgetc(in);       printf("Segundo caracter: %c\n",c);
 fsetpos(in,&pos);    fgets(str,128,in);
 printf("%s\n",str);
 fseek(in,2,SEEK_SET);fgets(str,128,in);
 printf("%s\n",str);
 fclose(in);
return 0;
}
```

Salida:

```
Primer caracter del archivo: H
Segundo caracter del archivo: o
Hola mundo
la mundo
```

feof,fflush


```c
#include <stdio.h>
#include <stdarg.h>
int main(int argc, char *argv[]){
  FILE *in=fopen("arch.txt","w+");
  char str[]={"Otro archivo\n"};
  fputs(str,in);
  fflush(in);
  fseek(in,0,SEEK_SET);
  while(!feof(in))
    putchar(fgetc(in));
  fclose(in);
  return 0;
}
```

Salida:

```
Otro archivo
```

ferror, clearerr,perror


```c
#include <stdio.h>
#include <stdarg.h>
int main(int argc, char *argv[]){
  FILE *in=fopen("arch.txt","r");
  char str[]={"Otro archivo\n"};
  fputs(str,in);
  if (ferror(in))
    printf("Error 1, no pudo realizar el fputs!\n");
  if (ferror(in))
    printf("Error 2, el error se mantiene!\n");
  if (ferror(in))
    perror("Error 3, el error fue:");
  clearerr(in);
  if (ferror(in))
    printf("Error 4, esto no se imprime!\n");
  fclose(in);
  return 0;
}
```

Salida:

```
Error 1, no pudo realizar el fputs!
Error 2, el error se mantiene!
Error 3, el error fue:: Bad file descriptor
```

Archivos binarios

Archivos binarios

- Los archivos binarios son un archivo en el disco como cualquier otro, pero el contenido solo tiene sentido si se sabe que se desea leer/escribir.

Archivos binarios

- Los archivos binarios son un archivo en el disco como cualquier otro, pero el contenido solo tiene sentido si se sabe que se desea leer/escribir.
- La información se imprime como bytes y uno puede desplazarse en le archivo para lectura de bytes. Eso evita complicaciones de formato y ayuda a almacenar solo los datos necesarios (no espacios, saltos de lineas,etc.)

Archivos binarios

- Los archivos binarios son un archivo en el disco como cualquier otro, pero el contenido solo tiene sentido si se sabe que se desea leer/escribir.
- La información se imprime como bytes y uno puede desplazarse en le archivo para lectura de bytes. Eso evita complicaciones de formato y ayuda a almacenar solo los datos necesarios (no espacios, saltos de lineas,etc.)

Funciones:

- **fopen y fclose con formato wb y rb** Abren y cierran un archivo binario.
- **long int ftell(FILE *stream)**. Obtiene la posición en el stream (pareja de fseek).
- **int fseek(FILE *stream, long int offset, int whence)**. Busca la posición de offset (pareja de ftell).
- **size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)**. Imprime un arreglo de datos a un stream.
- **void rewind(FILE *stream)**. Situa la posición al principio del archivo.
- **int fgetpos(FILE *stream, fpos_t *pos)**. Obtiene la posicion .
- **int fsetpos(FILE *stream, const fpos_t *pos)**. Fija la posición.

fopen, fclose, fwrite


```c
#include <stdlib.h>
#include <stdio.h>
typedef struct TRESDOU{
   double x,y,z;    struct TRESDOU *ptr;
}TRESDOU;
int main(int argc, char *argv[]){
  int n=7;
  FILE *out=fopen("arch.bin","wb");
  TRESDOU *array=malloc(n*sizeof(TRESDOU));
  for (int i=0; i<n;i++){
    array[i].x=i; array[i].y=n-i; array[i].z=i*i;
  }
  fwrite(&n,sizeof(int),1,out);
  fwrite(array,sizeof(TRESDOU),n,out);
  fclose(out);
  printf("Tam:%d\n",n*sizeof(TRESDOU)+sizeof(int));
  free(array);
  return 0;
}
```

Salida:

```
Tam:228
```

fopen, fclose, fread


```c
#include <stdlib.h>
#include <stdio.h>
typedef struct TRESDOU{
        double x,y,z;
        struct TRESDOU *ptr;
}TRESDOU;
int main(int argc, char *argv[]){
  int n=7;
  FILE *out=fopen("arch.bin","rb");
  TRESDOU *array=malloc(n*sizeof(TRESDOU));
  fread(&n,sizeof(int),1,out);
  fread(array,sizeof(TRESDOU),n,out);
  fclose(out);
  printf("%d %lf %lf %lf\n",n, array[5].x,array[5].y,
    array[5].z);
  free(array);
  return 0;
}
```

Salida:

```
7 5.000000 2.000000 25.000000
```

stdio.h, funciones

- **FILE *tmpfile(void)**. Genera un archivo temporal.
- **char *tmpnam(char *str)**. Genera un nombre de archivo temporal.
- **int remove(const char *filename)**. Elimina el archivo.
- **int rename(const char *old_filename, const char *new_filename)**. Renombra el archivo.
- **void setbuf(FILE *stream, char *buffer)**. Fija un buffer para el archivo.
- **int setvbuf(FILE *stream, char *buffer, int mode, size_t size)**. Fija el buffer, con tamaño y forma de hacer el buffering,

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.
2. Suponiendo que la pareja de caracteres es 'ab', calcule un equivalente entero como: (256*256)97*+(256)*98=16842752, porque (int)'a'=97 e (int)'b'=98.

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.
2. Suponiendo que la pareja de caracteres es 'ab', calcule un equivalente entero como: (256*256)97*+(256)*98=16842752, porque (int)'a'=97 e (int)'b'=98.
3. Calcule la frecuencia de cada par de caracteres.

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.
2. Suponiendo que la pareja de caracteres es 'ab', calcule un equivalente entero como: (256*256)97*+(256)*98=16842752, porque (int)'a'=97 e (int)'b'=98.
3. Calcule la frecuencia de cada par de caracteres.
4. Almacene la frecuencia y, caracteres (o valor entero de la multiplicación) en un arreglo de estructuras.

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.
2. Suponiendo que la pareja de caracteres es 'ab', calcule un equivalente entero como: (256*256)97*+(256)*98=16842752, porque (int)'a'=97 e (int)'b'=98.
3. Calcule la frecuencia de cada par de caracteres.
4. Almacene la frecuencia y, caracteres (o valor entero de la multiplicación) en un arreglo de estructuras.
5. Utilizando qsort de stdlib.h ordene las estructuras.

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.
2. Suponiendo que la pareja de caracteres es 'ab', calcule un equivalente entero como: (256*256)97*+(256)*98=16842752, porque (int)'a'=97 e (int)'b'=98.
3. Calcule la frecuencia de cada par de caracteres.
4. Almacene la frecuencia y, caracteres (o valor entero de la multiplicación) en un arreglo de estructuras.
5. Utilizando qsort de stdlib.h ordene las estructuras.
6. Represente todas las parejas del archivo como alguna de las 256 parejas mas frecuentes. Por ejemplo si *ab* es muy frecuente pero *aa* no lo es, puede considerar que *aa* es equivalente a *ab* y representarla por esta pareja.

Tarea 7

prog7.2 y 7.3 Escriba dos programas que:

1. Lea un libro de Ray Bradbury descargado de Gutenberg.org como texto. Se leera de 2 en 2 caracteres.
2. Suponiendo que la pareja de caracteres es 'ab', calcule un equivalente entero como: (256*256)97*+(256)*98=16842752, porque (int)'a'=97 e (int)'b'=98.
3. Calcule la frecuencia de cada par de caracteres.
4. Almacene la frecuencia y, caracteres (o valor entero de la multiplicación) en un arreglo de estructuras.
5. Utilizando qsort de stdlib.h ordene las estructuras.
6. Represente todas las parejas del archivo como alguna de las 256 parejas mas frecuentes. Por ejemplo si *ab* es muy frecuente pero *aa* no lo es, puede considerar que *aa* es equivalente a *ab* y representarla por esta pareja.
7. Cada pareja de caracteres se representará por un byte. Ejemplo: Supongamos que en lugar de usar las 256 parejas mas frecuentes usamos solo 4. Las parejas mas frecuentes podrían ser como: {"ab", "co","to","an"} a esto le llamaremos biblioteca.
   Podríamos representar cada elemento de la biblioteca con un byte como:
   {0x00, 0x01, 0x02, 0x03 } este sería el indice de la pareja en la biblioteca.
   Entonces sabemos que si leemos un 0x02 de un archivo (binario), corresponde a la pareja "to".

Dirección del libro de prueba:

Tarea 7,continuación

prog7.2

8 Almacene en un archivo binario: Los 256 valores de los equivalentes enteros de la biblioteca. y despues almacene todo el archivo de texto original, utilizando los indices de la biblioteca para representar cada pareja de caracteres del archivo original. De acuerdo al ejemplo anterior:
Para almacenar “ab” utilizariamos solo un byte que sería el 0x00. y la posición 0x00 de nuestra biblioteca deberia de tener el valor de (256*256)97*+(256)*98=16842752, porque (int)’a’=97 e (int)’b’=98.

Tarea 7,continuación

prog7.2

8 Almacene en un archivo binario: Los 256 valores de los equivalentes enteros de la biblioteca. y despues almacene todo el archivo de texto original, utilizando los indices de la biblioteca para representar cada pareja de caracteres del archivo original. De acuerdo al ejemplo anterior: Para almacenar “ab” utilizariamos solo un byte que sería el 0x00. y la posición 0x00 de nuestra biblioteca deberia de tener el valor de (256*256)97*+(256)*98=16842752, porque (int)’a’=97 e (int)’b’=98.

9 A partir del archivo binario reconstruya (con errores por que solo usa parejas de la biblioteca) el archivo de texto original.

Tarea 7,continuación

prog7.2

8 Almacene en un archivo binario: Los 256 valores de los equivalentes enteros de la biblioteca. y despues almacene todo el archivo de texto original, utilizando los indices de la biblioteca para representar cada pareja de caracteres del archivo original. De acuerdo al ejemplo anterior: Para almacenar “ab” utilizariamos solo un byte que sería el 0x00. y la posición 0x00 de nuestra biblioteca deberia de tener el valor de (256*256)97*+(256)*98=16842752, porque (int)’a’=97 e (int)’b’=98.

9 A partir del archivo binario reconstruya (con errores por que solo usa parejas de la biblioteca) el archivo de texto original.

10 El programa 7.2 toma como argumentos el nombre del archivo de texto original y el nombre del archivo binario que escribirá.

11 El programa 7.3 toma como argumentos el nombre del archivo binario (escrito en el programa anterior) y el nombre del archivo de texto reconstruido que escribirá. Los programas de diferentes estudiantes podrían imprimir diferentes archivos binarios (por la forma en que se cambien las parejas menos frecuentes por las mas frecuentes), pero deberian reconstruir (a patir del mismo binario) el mismo archivo de texto. El archivo binario solo contiene 256 elementos de la biblioteca y los indices correspondientes, en ese orden.

Proyectos y examen.

- Recepción de primer proyecto: 2 de noviembre a las 23:50

Proyectos y examen.

- Recepción de primer proyecto: 2 de noviembre a las 23:50
- Recepción de propuesta de segundo proyecto: del 14 de octubre al 2 de noviembre a las 23:50. Especificar: cual será la entrada (tentativa), proceso o función y salida (tentativa) del proyecto.

Proyectos y examen.

- Recepción de primer proyecto: 2 de noviembre a las 23:50
- Recepción de propuesta de segundo proyecto: del 14 de octubre al 2 de noviembre a las 23:50. Especificar: cual será la entrada (tentativa), proceso o función y salida (tentativa) del proyecto.
- Respuesta de propuesta de segundo proyecto(tentativamente): 2 días despues de recibido (y discutido en su caso).

Proyectos y examen.

- Recepción de primer proyecto: 2 de noviembre a las 23:50
- Recepción de propuesta de segundo proyecto: del 14 de octubre al 2 de noviembre a las 23:50. Especificar: cual será la entrada (tentativa), proceso o función y salida (tentativa) del proyecto.
- Respuesta de propuesta de segundo proyecto(tentativamente): 2 días despues de recibido (y discutido en su caso).
- Recepción de segundo proyecto: 1 de diciembre a las 23:50. Enviar: reporte donde indique que hace, como, como se ejecuta y ejemplo de entrada y salida. Código, ejemplo de ejecución y presentación muy pequeña. La parte que indica *cómo* debe de permitir que un programador reproduzca el mismo *funcionamiento* del programa. Es decir si calculan un promedio o integral, no es necesario describir como se calcula porque eso todo el mundo lo sabe. Sin embargo, si deben de especificar que integran o promedian.

Proyectos y examen.

- Recepción de primer proyecto: 2 de noviembre a las 23:50
- Recepción de propuesta de segundo proyecto: del 14 de octubre al 2 de noviembre a las 23:50. Especificar: cual será la entrada (tentativa), proceso o función y salida (tentativa) del proyecto.
- Respuesta de propuesta de segundo proyecto(tentativamente): 2 días despues de recibido (y discutido en su caso).
- Recepción de segundo proyecto: 1 de diciembre a las 23:50. Enviar: reporte donde indique que hace, como, como se ejecuta y ejemplo de entrada y salida. Código, ejemplo de ejecución y presentación muy pequeña. La parte que indica *cómo* debe de permitir que un programador reproduzca el mismo *funcionamiento* del programa. Es decir si calculan un promedio o integral, no es necesario describir como se calcula porque eso todo el mundo lo sabe. Sin embargo, si deben de especificar que integran o promedian.
- Presentaciones de 6-7 minutos de segundo proyecto: 2 de diciembre. La presentación dice: qué hace el programa, entrada, salida y ejemplo de ejecución, y si les sobra uno o dos minutos, una idea muy general del como.

Proyectos y examen.

- Recepción de primer proyecto: 2 de noviembre a las 23:50
- Recepción de propuesta de segundo proyecto: del 14 de octubre al 2 de noviembre a las 23:50. Especificar: cual será la entrada (tentativa), proceso o función y salida (tentativa) del proyecto.
- Respuesta de propuesta de segundo proyecto(tentativamente): 2 días despues de recibido (y discutido en su caso).
- Recepción de segundo proyecto: 1 de diciembre a las 23:50. Enviar: reporte donde indique que hace, como, como se ejecuta y ejemplo de entrada y salida. Código, ejemplo de ejecución y presentación muy pequeña. La parte que indica *cómo* debe de permitir que un programador reproduzca el mismo *funcionamiento* del programa. Es decir si calculan un promedio o integral, no es necesario describir como se calcula porque eso todo el mundo lo sabe. Sin embargo, si deben de especificar que integran o promedian.
- Presentaciones de 6-7 minutos de segundo proyecto: 2 de diciembre. La presentación dice: qué hace el programa, entrada, salida y ejemplo de ejecución, y si les sobra uno o dos minutos, una idea muy general del como.
- Último examen 4 de diciembre a la hora de clase.