Clase 17. Librerias, muestreo Montecarlo.

setjmp.h

Excepción

Una **excepción** es un comportamiento, dato, o valor de retorno de una función, para el cual el programa no está diseñado para manejar explícitamente. Es decir esa situación *excepcional* no constituye el *core* o función principal del programa.
El manejo de las excepciones en C es más bien manual (como con signal), la libreria setjmp.h provee una forma de manejar las excepciones.

setjmp.h, variables,macros

Variable	Descripción
jmp_buf	Es un arreglo que se utiliza para almacenar la informacion de la macro setjmp() y de la función longjmp().

Macro	Descripción
int setjmp(jmp_buf environment)	Guarda el ambiente actual en la variable *environment* para utilizarlo después por la función longjmp. Cuando la macro regresa de la invocación de si misma, regresa cero. La misma macro es llamada por longjmp, cuando es llamada lor longjmp entonces regresa el valor (value) enviado por esta.
void longjmp(jmp_buf environment, int value)	Un posible uso de longjmp es el manejo de excepciones, es decir cuando una función o programa llega a un punto donde se obtiene un resultado no esperado, este se puede *reiniciar* desde donde se fijó **setjmp**, por medio de una llamada a longjmp.

setjmp.h


```c
#include <stdio.h>
#include <setjmp.h>

int main(int argc, char *argv[])
{
    int val;
    jmp_buf env_buffer;
    ///Reiniciar a partir de aqui
    val = setjmp( env_buffer );
    if (val!=0){
      //Manejo la excepci\'on
       printf("Hubo reinicio , val=%d\n",val);
      //El manejo debe incluir una forma
      //de no regresar "infinitamente"
       exit(0);
    }
    ///Regresa 3 lineas
    longjmp(env_buffer,3);
    return 0;
}
```

Salida:

```
Hubo reinicio, val=3
```

setjmp.h


```c
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
 int val; size_t n=99999999;
 double *x;
 jmp_buf env_buffer;
 val = setjmp( env_buffer );
 if (val!=0){
   //Manejo la excepcion
   n=val;
 }
 ///Regresa a donde esta setjmp
 x=malloc(n*n*sizeof(double));
 if (x==NULL){
   printf("No reservo! n=%d\n",n);
   longjmp(env_buffer,n/10+1);
 }
 printf("Ya quedo n=%d\n",n);
 free(x);
 return 0;
}
```

Salida:

```
No reservo! n=99999999
No reservo! n=10000000
No reservo! n=1000001
No reservo! n=100001
Ya quedo n=10001
```

setjmp.h


```c
#include <setjmp.h>
#include <stdlib.h>
#include <string.h>

int manejo_de_excepciones(int
    val,jmp_buf env_buffer){
if (val==0)
 return 0;
if (val==1)
 printf("No hay memoria\n");
else if(val==2)
 printf("No existe archivo\n"
    );
else if(val==3)
 printf("No hay argumentos\n"
    );
else
 printf("Excepcion
    desconocida\n");
longjmp(env_buffer,val);
return 0;
}
```

```c
int main(int argc, char *argv
    []){
int val; double *x;
size_t n=999999;
FILE *in=fopen("archivo.
    inexistente","r");
jmp_buf env_buffer;
val = setjmp( env_buffer );
if (val!=0)
  exit(0);
x=malloc(n*n*sizeof(double));
if (x==NULL)
 manejo_de_excepciones(1,
    env_buffer);
free(x);
if (in==NULL)
 manejo_de_excepciones(2,
    env_buffer);
if (argc<1)
 manejo_de_excepciones(3,
    env_buffer);
return 0;
}
```

assert.h

assert.h

La cabecera *assert* se puede utilizar para verificar que las suposiciones hechas por el programador se cumplen y en caso de que no se cumplan abortar el programa. La única macro definida en la librería assert.h es:

void assert(int expression)
assert recibe cualquier expresion, si la expresión evalua VERDADERO (diferente de cero) assert no hace nada, si la expresión evalua 0, assert envia un mensaje de error y aborta.

assert.h


```c
#include <assert.h>
#include <stdio.h>
int main()
{
    int x;
    printf("Ingrese un numero: ");
    scanf("%d", &x);
    assert(x<3);
    printf("Ingrese otro numero: ");
    scanf("%d", &x);
    return(0);
}
```

Salida:

```
Ingrese un numero: 4
ej04: ej04.c:12: main: Assertion 'x<3' failed.
Aborted
```

assert.h

Es facil hacer mi propio assert:


```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#define mejor_assert(expr, msg, print) \
if(expr && print) printf("Expresion %s,ok!\n", #expr);\
else{ printf("\n%s\n",#msg);\
  printf("Expresion %s aborta\n", #expr); abort();  }
int main(){
   int x;
   printf("Ingrese un numero: "); scanf("%d", &x);
   mejor_assert(x<3,"Error personal 1", 1);
   printf("Ingrese otro numero: "); scanf("%d", &x);
   mejor_assert(x>11,"Error personal 2", 1);
   return(0); }
```

Salida:

```
Ingrese un numero: 2
Expresion x<3, ok!
Ingrese otro numero: 11
"Error personal 2"
Expresion x>11 provoca aborto
Aborted
```

math.h

setjmp.h, variables,macros

Variable	Descripción
HUGE_VAL	Si el valor de una función es muy grande se regresa este valor y un error..

double acos(double x)	double asin(double x)	double atan(double x)
double atan2(doubly y, double x)	double cos(double x)	double cosh(double x)
double sin(double x)	double sinh(double x)	double tanh(double x)
double exp(double x)	double frexp(double x, int *exponent)	double ldexp(double x, int exponent)
double log(double x)	double log10(double x)	double modf(double x, double *integer)
double pow(double x, double y)	double sqrt(double x)	double ceil(double x)
double fabs(double x)	double floor(double x)	double fmod(double x, double y)
double exp(double x)	double frexp(double x, int *exponent)	double ldexp(double x, int exponent)

Generación de números con distirbución normal

Existen algoritmos que mediante transformaciones de otras distribuciones, entre ellos:

- El método de Marsaglia.
- El método de Box-Muller.
- El método de Zigurat.

criterio de aceptación de metropolis

El criterio de aceptación de metropolis sirve para simular muestras de un distribución proporcional a $f(x)$, por lo cual se pueden generar números de alguna distribución arbitraria.

El criterio de aceptación de Metropolis:

1. Sea x_t la solución actual y $\hat{x}$ la solución perturbada.

criterio de aceptación de metropolis

El criterio de aceptación de metropolis sirve para simular muestras de un distribución proporcional a $f(x)$, por lo cual se pueden generar números de alguna distribución arbitraria.

El criterio de aceptación de Metropolis:

1. Sea x_t la solución actual y $\hat{x}$ la solución perturbada.
2. $\Delta f = f(\hat{x}) - f(x_t)$

criterio de aceptación de metropolis

El criterio de aceptación de metropolis sirve para simular muestras de un distribución proporcional a $f(x)$, por lo cual se pueden generar números de alguna distribución arbitraria.

El criterio de aceptación de Metropolis:

1. Sea x_t la solución actual y $\hat{x}$ la solución perturbada.
2. $\Delta f = f(\hat{x}) - f(x_t)$
3. **If** $(\Delta f(x) \leq 0\ )$ **then** $x_{t+1} = \hat{x}$

criterio de aceptación de metropolis

El criterio de aceptación de metropolis sirve para simular muestras de un distribución proporcional a $f(x)$, por lo cual se pueden generar números de alguna distribución arbitraria.

El criterio de aceptación de Metropolis:

1. Sea x_t la solución actual y $\hat{x}$ la solución perturbada.
2. $\Delta f = f(\hat{x}) - f(x_t)$
3. **If** $(\Delta f(x) \leq 0\ )$ **then** $x_{t+1} = \hat{x}$
4. **else** El nuevo punto se acepta con probabilidad $\exp(\frac{-\Delta f}{T})$

Tarea 9

prog9.5 Implementar el muestro de metropolis para x continua en el rango de -10 a 10 y con una distribución de probabilidad proporcional a $|x+3|sin(1.9x)+0.1(x-3)^2$. Graficar histograma de 1000 bins para 1e7 muestras contra la función (escalarlo donde se pueda contrastar la función contra el histograma).

prog9.6 Implementar algún método de generación de números pseudo-aleatorios con distribución Normal.

1. En la diapositiva anterior $f(x) = (|x+3|sin(1.9x)+0.1(x-3)^2)$.
2. Para $T = 1$ graficamos $exp(-(|x+3|sin(1.9x)+0.1(x-3)^2)/T)$ (Noten el menos en la función).
3. Graficar el histograma de las soluciones **aceptadas** por el método de metropolis.
4. Poner encima de la gráfica del histograma una gráfica de $factor * exp(-(|x+3|sin(1.9x)+0.1(x-3)^2)/T)$ donde $factor$ es un número que deben escoger para

Librerias faltantes

- **locale.h**. Podemos definir cual es el punto decimal, cual es la separación para los miles, el símbolo de moeda,etc.

Librerias faltantes

- **locale.h**. Podemos definir cual es el punto decimal, cual es la separación para los miles, el símbolo de moeda,etc.
- **erno.h**. Sirve para reportar errores de llamadas a sistema. Por ejemplo: apertura de archivo, memoria, etc.

Librerias faltantes

- **locale.h**. Podemos definir cual es el punto decimal, cual es la separación para los miles, el símbolo de moeda,etc.
- **erno.h**. Sirve para reportar errores de llamadas a sistema. Por ejemplo: apertura de archivo, memoria, etc.
- **ctype.h** Verificación del uso de los caracteres, ej. si se pueden imprimir, si son números, si son mayusculas o minusculas, si son caracteresd de control, etc.

Librerias faltantes

- **locale.h**. Podemos definir cual es el punto decimal, cual es la separación para los miles, el símbolo de moeda,etc.
- **erno.h**. Sirve para reportar errores de llamadas a sistema. Por ejemplo: apertura de archivo, memoria, etc.
- **ctype.h** Verificación del uso de los caracteres, ej. si se pueden imprimir, si son números, si son mayusculas o minusculas, si son caracteresd de control, etc.
- **float.h** Define limites de los valores de los flotantes: máximos y mínimos valores de dobles y flotantes.

Librerias faltantes

- **locale.h**. Podemos definir cual es el punto decimal, cual es la separación para los miles, el símbolo de moeda,etc.
- **erno.h**. Sirve para reportar errores de llamadas a sistema. Por ejemplo: apertura de archivo, memoria, etc.
- **ctype.h** Verificación del uso de los caracteres, ej. si se pueden imprimir, si son números, si son mayusculas o minusculas, si son caracteresd de control, etc.
- **float.h** Define limites de los valores de los flotantes: máximos y mínimos valores de dobles y flotantes.
- **limits.h** Maximos y mínimos de enteros.

ejemplo erno.h

Nota curiosa: si no se incluye string.h strerror produce segfault.


```c
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include <string.h>
int main(){
 FILE *in=fopen("Archivo.inexistente","r");
 if (in==NULL){
    printf("errno=%d\n",errno);
    fprintf(stderr,"Error:%s\n",strerror(errno));
 }
 sqrt(-2.35);
 if(errno==EDOM) printf("Error:%s\n",strerror(errno));
 log(0.0);
 if(errno==ERANGE) fprintf(stdout,"Error:%s\n",strerror
    (errno));
return 0;
}
```

Salida:

```
errno=2
Error:No such file or directory
Error:Numerical argument out of domain
Error:Numerical result out of range
```

ejemplo ctype.h

```c
#include <stdio.h>
#include <ctype.h>
int main(){
  char c='a';  if (isalpha(c))  printf("Es letra\n");
  c=',';  if (ispunct(c))  printf("Puntuacion\n");
  c='2';  if (isdigit(c))  printf("Digito\n");
  c='\t';  if (iscntrl(c))  printf("Control\n");
  c='a';  if (islower(c))  printf("Miniscula\n");
  c='A';  if (isupper(c))  printf("Mayuscula\n");
return 0;
}
```

Salida:

```
Es letra
Puntuacion
Digito
Control
Miniscula
Mayuscula
```

ejemplo locale.h


```
1  typedef struct {
       char *decimal_point;
3      char *thousands_sep;
       char *grouping;
5      char *int_curr_symbol;
       char *currency_symbol;
7      char *mon_decimal_point;
       char *mon_thousands_sep;
9      char *mon_grouping;
       char *positive_sign;
11     char *negative_sign;
       char int_frac_digits;
13     char frac_digits;
       char p_cs_precedes;
15     char p_sep_by_space;
       char n_cs_precedes;
17     char n_sep_by_space;
       char p_sign_posn;
19     char n_sign_posn;
   } lconv;
```

Tarea 9

prog9.7 Realice un programa que reciba 2 números de entrada m y n. El primero es el tamaño de un vector de doubles y el segundo el número de vectores que se generarán.

1. Usando la libreria setjmp.h valide que el número de bytes utilizados por el vector sea menor que 100 MB, si no es así reduzca el tamño del vector pero aumente el número n. Por ejemplo si originalmente el vector ocupa 200 MB y $n = 5000$, entonces se reduce el vector a 100 MB pero se aumenta $n = 10000$.
2. Realice un 10% de las operaciones totales y tome el tiempo utilizando una macro, la macro llamara a la función clock() en windows para tomar el tiempo en microsegundos, y usleep en linux. Si el tiempo del 10% de las operaciones es mayor a 1 segundo entonces aborte el programa con assert.
3. Las operaciones que se llevarán a cabo con el vector son: generar números aleatorios con distribución normal de media 3 y varianza 1, obtener la raíz cuadrada de cada número y almacenarla en el vector.
4. Algúnos números de los generados serán menores que cero y por lo tanto generarán un error de dominio. Contabilice el porcentaje de errores con respecto del total de operaciones utilizando errno. Imprima un mensaje donde muestre cues la proporción esperada (utilizando la normal) de errores, y cual es la proporción que entrega el programa.

Terminamos las librerias estándar de C!