Clase 16. Preprocesador. Algoritmos de generación de números pseudoaleatorios.

Directivas de preprocesador

- **Una macro** es una regla para reemplazar o mapear un conjunto de caracteres en otro conjunto de caracteres.

- **Una macro** es una regla para reemplazar o mapear un conjunto de caracteres en otro conjunto de caracteres.
- **El preprocesador de C** es un procesador de macros que utiliza de forma automática el compilador de C para transfformar el programa antes de la compilación. Las macros del preprocesador permiten:

1. Incluir archivos de cabecera.
2. La expansión del macro. Se pueden definir macros que son abreviaciones para fragmentos arbitrarios de código C, entonces el preprocesador sustituye las macros con sus definiciones antes de hacer la compulación.

- **Una macro** es una regla para reemplazar o mapear un conjunto de caracteres en otro conjunto de caracteres.
- **El preprocesador de C** es un procesador de macros que utiliza de forma automática el compilador de C para transfformar el programa antes de la compilación. Las macros del preprocesador permiten:

1. Incluir archivos de cabecera.
2. La expansión del macro. Se pueden definir macros que son abreviaciones para fragmentos arbitrarios de código C, entonces el preprocesador sustituye las macros con sus definiciones antes de hacer la compulación.
3. Compilación condicional. Se utiliza para incluir o excluir partes del programa de la compilación.

- **Una macro** es una regla para reemplazar o mapear un conjunto de caracteres en otro conjunto de caracteres.
- **El preprocesador de C** es un procesador de macros que utiliza de forma automática el compilador de C para transfformar el programa antes de la compilación. Las macros del preprocesador permiten:

1. Incluir archivos de cabecera.
2. La expansión del macro. Se pueden definir macros que son abreviaciones para fragmentos arbitrarios de código C, entonces el preprocesador sustituye las macros con sus definiciones antes de hacer la compulación.
3. Compilación condicional. Se utiliza para incluir o excluir partes del programa de la compilación.
4. Control de líneas. Si se reordena un programa moviendo líneas de código a archivos externos. Se pueden usar directivas para decirle al compilador cual era la posición original del código y te reporte en esa posición los errores en la compilación.

Directivas de preprocesador

Las directivas de preprocesador comienzan con **#**. Antes de compilar el programa se realizan las siguientes transformaciones:

1. Todos los comentarios se reemplazan con espacios simples.
2. Las diagonales invertidas (backslash) que indican continuación de línea en la siguiente línea se eliminan, ya que solo son para fines cosméticos y no cambian el significado.
3. Los macros predefinidos son reemplazados con sus expasiones.

directiva #include para gcc

- **#include** <>. Esta variante de la macro *include* se utiliza para incluir archivos de la lista estándar de directorios. Se pueden incluir directorios para buscar archivos con la opción *-I* y eliminar las rutas estándar con la opción **-nostdinc**. Los comentarios o caracteres especiales dentro de <> no son sustituidos. Ej. si uso $< /**/ >$ busca el archivo /**/.

directiva #include para gcc

- **#include** <>. Esta variante de la macro *include* se utiliza para incluir archivos de la lista estándar de directorios. Se pueden incluir directorios para buscar archivos con la opción *-I* y eliminar las rutas estándar con la opción **-nostdinc**. Los comentarios o caracteres especiales dentro de <> no son sustituidos. Ej. si uso $< /**/ >$ busca el archivo /**/.
- **#include "archivo"**. Esta variante de la macro *include* se utiliza para incluir archivos del programador. primero busca en el directorio actual, y luego en los directorios estándar. Los backslash en los nombres se consideran parte del nombre del archivo.

directiva #include para gcc

- **#include** <>. Esta variante de la macro *include* se utiliza para incluir archivos de la lista estándar de directorios. Se pueden incluir directorios para buscar archivos con la opción *-I* y eliminar las rutas estándar con la opción **-nostdinc**. Los comentarios o caracteres especiales dentro de <> no son sustituidos. Ej. si uso $< /**/ >$ busca el archivo /**/.
- **#include "archivo"**. Esta variante de la macro *include* se utiliza para incluir archivos del programador. primero busca en el directorio actual, y luego en los directorios estándar. Los backslash en los nombres se consideran parte del nombre del archivo.
- **#include *lo que sea***. *Lo que sea* puede ser una macro o nombre de archivo. Si la macro se expande debe de resultar en alguna de las formas de arriba (o sea al expadir debe incluir un archivo como los anteriores). Puede servir para que las inclusiones se realicen dependiendo del sistema operativo o de algún tipo de archivos locales a la maquina donde se compile.

Lo que se puede y no se debe hacer

header.h

```
1 #include <stdio.h>
  int func\
3 ion()
  {
```

main.c

```
#include "header.h"
2
    printf("Funcion a medias\n");
4 }
  int main()
6 {
    funcion();
8   return 0;
  }
```

Incluir solo una vez un archivo:

```
1 #ifndef _ARCHIVO_YA_INCLUIDO
  #define _ARCHIVO_YA_INCLUIDO
3  contenido del archivo...
  #endif
```

#define Macros en C

La macro en C es un tipo de abreviación que será sustituida antes de compilar. Antes de usar la macro hay que definirlas:

#define [nombre del iedntificador] [valor]

```
#define BUFFER_SIZE 10
#define TABLE_SIZE BUFFER_SIZE
#define DOUBLE_TABLE_SIZE 2*TABLE_SIZE
#include <stdio.h>
int main()
{
   printf("Double Table Size=%d\n",DOUBLE_TABLE_SIZE);
 return 0;
}
```

Se expande la operación, sustituyendo primero el primer #define, después el segundo, etc. Note que la operación se expande tal cual. (Ver ejemplo sig).

#define Macros en C, ejemplo, expansión (posiblemente incorrecta) de operaciones


```
1 #define BUFFER_SIZE 10
#define TABLE_SIZE BUFFER_SIZE
3 #define DOUBLE_TABLE_SIZE 2+TABLE_SIZE
#include <stdio.h>
5 int main()
{
7   printf("Double Table Size=%d\n",2*DOUBLE_TABLE_SIZE);
 return 0;
9 }
```

¿Cual es el resultado de la expasión?

Macros con argumentos

En la definición de la macro con argumentos NO DEBE de haber espacios entre los parentesis y la macro, aunque en la llamada puede haber espacios. NO SE DEBE de usar nombres de variables o funciones con el mismo nombre de la macro (aunque es posible), para evitar confunsiones.

```
1 #define SUMA(X,Y) (X+Y)
#include <stdio.h>
3 int main()
{
5   printf("SUMA=%d\n",SUMA(3,2));
 return 0;
7 }
```

Macros predefinidas (GNU)

- __FILE__ nombre del archivo actual.
- __LINE__ Es la línea actual.
- __DATE__ Fecha en la que se ejecuto el preprocesador.
- __TIME__ Hora en la que se ejecuto el preprocesador.
- __cplusplus Indica si se está utilizando un compilador de C++.
- __BASE_FILE__ Expande al nombre del archivo que se le da de entrada al compilador.
- __INCLUDE_LEVEL__ Se incrementa con cada include se decrementa con cada fin de archivo.
- __VERSION__ Versión del compilador.
- __OPTIMIZE__ Es una bandera para definir otras macros que ayudan a la optimización del código.
- ... otras.

Macros predefinidas (GNU)

header.h

```
#include <stdio.h>
#define
    printFileInformation
    printf("Archivo donde
    se sustituye el macro:
     %s\n",__FILE__);
#define
    printLineInformation
    printf("En la linea: %
    d\n",__LINE__);
```

main.c

```
#include "ej04.h"
int main()
{
  printFileInformation
  printLineInformation
  printf("Se compilo el
    dia: %s a las %s con
    la version: %s\n",
    __DATE__,__TIME__,
    __VERSION__);
 return 0;
}
```

```
 Archivo donde se sustituye el macro: ej04.c
En la linea: 6
Se compilo el dia: Oct 13 2014 a las 13:46:32
con la version: 4.7.2 20130108 [gcc-4_7-branch
 revision 195012]
```

Macros Stringification

La “cadenización” consiste en convertir en una cadena de caracteres un argumento de la macro.

```
#include <stdio.h>

#define NOMBREDELAVARIABLE(VAR)   \
{ \
    printf("Mi variable se llama " #VAR " \n"); \
}

int main()
{
  int x;
  NOMBREDELAVARIABLE(x)
 return 0;
}
```

```
Mi variable se llama x
```

Macros, Concatenación

La concatenación se realiza por medio de dos simbolos ##.

```
1  #include <stdio.h>
2
3  #define NOMBRE( V ) #V
4  #define definicion(type, var) type var ## _arreglo[10];
5
6  int main()
7  {
8    int x;
9    definicion(double, x);
10   printf("Se definio un arreglo tipo %s\n", NOMBRE(
11     double));
12   return 0;
13 }
```

```
Se definio un arreglo tipo double
```

Macros. Definición condicional de macros

Existen directivas de control para la compilacion como #if, #else, #ifdef #ifndef y #endif.

```
#include <stdio.h>

#define level 5

#if level>3
    #define SIZE 10
#else
    # define SIZE 20
#endif
int main()
{
   printf("SIZE=%d\n",SIZE);
 return 0;
}
```

```
SIZE=10
```

Macros. Definición condicional de macros

Existen directivas de control para la compilacion como #if, #else, #ifdef #ifndef y #endif.

```
#include <stdio.h>

#define level 5

#ifdef level
    #define SIZE level
#else
    # define SIZE 10
#endif
int main()
{
   printf("SIZE=%d\n",SIZE);
 return 0;
}
```

```
SIZE=5
```

Macros. Compilación condicional

Existen directivas de control para la compilacion como #if, #else, #ifdef #ifndef y #endif.

```
#include <stdio.h>

#ifdef _WIN32
    #define Plataforma "Windows 32 bits"
#elif __linux__
    #define Plataforma "Linux"
#elif __WIN64
    #define Plataforma "Windows 64 bits"
#elif __APPLE
    #define Plataforma "Macintosh"
#else
    #define Plataforma "\nNo se en que plataforma estoy
    !\n"
#endif
int main()
{
  printf("Plataforma: %s\n",Plataforma);
 return 0;
}
```

Macros. Compilación condicional

Existen directivas de control para la compilacion como #if, #else, #ifdef #ifndef y #endif.

```
#include <stdio.h>

#ifdef _WIN32
    #define retardo(N) sleep(N)
#elif __linux__
    #define retardo(N) usleep(1000000*N)
#endif
int main()
{
  retardo(1);

 return 0;
}
```

Otras directivas de preprocesador

- Existen otras directivas por ejemplo #pragma. Las cuales son definidas por algún estándar y proveén *pragmatic information* y son tratadas diferente por cada implementación.
- #error Detiene la compilación y emite un mensaje.
- #undef Elimina la definición de un macro (como si no se hubiese definido, sirve para redefinirlo o cambiar el comportamiento de la compilación).

Tambien existen muchas macros definidas que dependen del sistema, compilador, API en uso, etc.

Comentarios

- Las macros ayudan a la portabilidad del código, y definición de funciones, datos y operaciones genéricas.

Comentarios

- Las macros ayudan a la portabilidad del código, y definición de funciones, datos y operaciones genéricas.
- Sin embargo en la mayoría de las veces no es recomendable usarlas (es decir, si hay una función o variable que haría la misma función mejor uso la función o variable), debido a que el comportamiento bajo ciertas operaciones no puede decir bien predicho.

Comentarios

- Las macros ayudan a la portabilidad del código, y definición de funciones, datos y operaciones genéricas.
- Sin embargo en la mayoría de las veces no es recomendable usarlas (es decir, si hay una función o variable que haría la misma función mejor uso la función o variable), debido a que el comportamiento bajo ciertas operaciones no puede decir bien predicho.
- Podemos ver lo que produce el preprocesador de gcc con la opción -E, ejemplo:
  gcc -E programa.c

Tarea 9

●

Prog9.1 Escriba una macro que le ayude a definir funciones para requerir y devolver la memoria de arreglos unidimensionales y bidimensionales de diferentes tipos, ejemplo, si mi macro se llama **memoria**:

memoria(double,2)

Define una función para requerir una matriz bidimensional de tipo double y para regresar la memoria solicitada.

Prog9.2 Escriba un programa con una macro que se le envie un apuntador, un tipo y un número entero **n** y devuelva: el contenido del puntero, casteado al tipo que se le envio, después de recorrer **n** locaciones de ese tipo.

prog9.3 Escriba un programa con una macro que (ayude a usar) un arreglo unidimensional estático de tamaño **n** donde los indices del arreglo comienzan en **istart** y terminan en **iend**, con cierto tipo predefinido, si la macro se llamará **declarray**:

declarray(double,x,-5,10)

depués de esta línea el programador puede utilizar como un arreglo double a x, cuyos indices empiezan de -5 y terminan en 10.

Link interesante

The C preprocessor. Revisión de julio de 1992. **Richard M. Stallman**

`http://docs.freebsd.org/info/cpp/cpp.pdf`

Nota: La forma en que se preprocesan actualmente las directivas generalmente es del estándar de 1999, no cambia las anteriores, agrega unas y deja obsoletas otras, pero no hay cambio sustancial.

Generador de números pseudoaleatorios, LCG

Un generador de números pseudoaleatorios es básicamente un algoritmo que devuelve un conjunto de números cuyas propiedades parecen similares a las de los números aleatorios. Usualmente, parecen tener una distribución uniforme, la aleatoriedad consiste en el desconocimiento de una semilla (un número inicial). Uno de los más comunes es: el generador lineal congruencial:

$$x^{t+1} = (ax^t + c) mod m$$

Es el pseudo generador utilizado por gcc, con $m = 2^{31}$, $a = 1103515245$ y $c = 12345$. Y es, en general, el utilizado por muchos compiladores, utilizando diferentes valores de constantes. No se le considera un buen generador.

Linear Feedback Shift Register

Un generador llamado LFSR o combined Tausworthe generator, provee de una secuencia de números utilizando solo **XOR**:
Dado x^t, s_1, s_2, s_3 y M.:

- Recorrer los bits de x^t s_1 bits a la izquierda ($<<$). Almacenar en y.
- En z almacenar el XOR de y con x^t.
- Recorrer z s_2 bits a la derecha, almacenar en b.
- En w almacenar el AND de x^t y M.
- Recorrer w s_3 bits hacia la izquierda.
- Devolver el resultado del XOR de w y b.

Generador compuesto

Supongamos que la función que implementa el generador lineal congruencial con $a = 1664525$, $c = 1013904223UL$ y $m = 1$ (el UL es para indicar que es unsigned long), se llama $LCGStep(.)$. Y que la funcion que implementa el LFSR se llama: $TauStep(., s_1, s_2, s_3, M)$.
Un generador compuesto que llamaremos Hybrid, realiza las siguientes operaciones:
$x^{t+1} =$
$2.328306436538 7e-10(TausStep(., 13, 19, 12, 4294967294UL)$ XOR $TausStep(., 2, 25, 4, 4294967288UL)$ XOR $TausStep(., 3, 11, 17, 4294967280UL)$ XOR $LCGStep(.)$

Los . son los valores de la x^t (el valor anterior) para cada generador (es una variable diferente en cada generador).

Tarea 9

●

Prog9.4 Implente los generadores LCGStep, TausStep e Hybrid. Usando una estructura de 4 enteros sin signo: a) variables static para almacenar el valor anterior. b) Enviando y sobre escribiendo el último valor generado. c) Declarando y actualizando una variable global. Genere 1e6 números aleatorios con cada uno de los generadores (LCG, Tau e Hybrid) con cada una de las formas anteriores (a,b y c). Calcule el tiempo de cómputo de cada uno, compilado con y sin la bandera -02 y declarando las estructuras con y sin **register**. Comente sus conclusiones en su reporte.