ESTRUCTURA DE DATOS

TEMA 1 ESTRUCTURA DE DATOS PILA

ABSTRACCIÓN

Abstracción
El proceso de abstracción puede resumirse en 4 etapas:

- Abstracción. Análisis (determinación) de las propiedades que son importantes y de las que no lo son.
- Representación. Elección de símbolos para representar dichos conceptos y propiedades.
- Manipulación. Reglas para transformar las representaciones simbólicas anteriores con vistas a predecir hechos del mundo real.
- Axiomatización. Declaraciones rigurosas de las propiedades abstraídas del mundo real.

Ejemplo de Abstracción:
El concepto de número
Abstracción del concepto cantidad
Representación de los números
Manipulación de los números (operación Suma,. Resta, etc)
Axiomas de los números y sus manipulaciones.

Abstracción y programación

- Abstracción de las propiedades que nos interesan. Ejemplo: Turbulencias del ala de un avión (elasticidad y forma, color).
- Representación de las propiedades que nos interesan. Ejemplo: Discretización de la presión en el ala y representación en forma de matriz.
- Manipulación de las propiedades. Programación propiamente dicha. Ejemplo: Aplicación de ecuaciones diferenciales para determinar la evolución de la presión y las turbulencias en el ala.
- Axiomas. Implícita en la manipulación y programación.

El éxito de un programa depende de 3 condiciones básicas:

1 La axiomatización es una descripción correcta de aquellos aspectos de la realidad.
2 La axiomatización es una descripción correcta del comportamiento de un programa.
3 La representación elegida y la manipulación son tales que el coste de ejecutar el programa se considera aceptable.

TIPO ABSTRACTO DE DATOS

TIPO

Los diferentes objetos de información con los que un programa C trabaja se conocen colectivamente como datos. Todos los datos tienen un tipo asociado con ellos

La asignación de tipos a los datos tiene dos objetivos principales:

1. Detectar errores de operaciones en programas.
Determinar como ejecutar las operacionesUn tipo de dato determina la clase y rango de valores que pueden ser asumidos por una variable o expresión.

C maneja cuatro clases de tipos:

1. Los enteros
   char caracter
   short entero corto
   i**nt** entero
   long entero largo

1.Los enteros sin signo
   unsigned char
   unsigned short
   unsigned int
   unsigned long
   enum para determinar un tipo de dato con posibles valores que son consecutivos. El manejo es similar a una estructura. El primer valor va a tomar un 0, el segundo 1 y así sucesivamente.

3. Flotantes
   float
   double

4. Apuntadores

Datos definidos por el usuario:
Estructuras
Arreglos .. etc

Tabla 1 Rangos de los tipos de datos en C

Dato	Tamaño (bits)	Rango	Ejemplo
char	8 (1 byte)	-128...0...127	char nombre[15],sexo;
int	16	-32768...0...32767	int num;
short	16 (2 bytes)	-32768...0...32767	short x,y;
long	32 (4bytes)	-2147483648...2147483647	
unsigned char	8	0...255	
unsigned int	16	0...65535	
unsigned short	16	0...65535	
unsigned long	32	0...4 294 967 295	
enum	32		**enum** semana { domingo, martes,

			miercoles, jueves, sabado} dia;
float	32	negativos : -3.402823E+38...-1.40129E45 positivos 1.401293E-45 ... 3.40282E38	
double	64 (8 bytes)	negativos: -1.79769313316E308... – 4.94065E324 positivos: 4.94065E-324... 1.797334862316E308	
long double	80	3.4 E-4932... 1.1E+4932	
Apuntador	16	No aplica	

Estructuras de datos lineales, representaciones secuenciales

Concepto de estructuras de datos

Los términos tipo de datos (o simplemente <<tipo>>), <<estructura de datos>> y <<tipo de dato abstracto>> parecen semejantes pero su significado es diferente.

Los tipos abstractos de datos tienen mucho en común con el diseño descendente:

Ocultación de la información. Esconder los detalles de la implementación pero no la especificación.

Abstracción. Separar la especificación (propiedades) de la implementación (construcción).

Objetivo

El objetivo de los tipos abstractos de datos (TAD) es la manipulación de los datos del programa considerando sólo su representación lógica y despreocupándose de su ubicación física (implementación). Ejemplo: Cuando se define una variable de tipo de dato entera int x; el programador sólo necesita saber la definición y modo de uso del tipo pero no su implementación (complemento a 1 o a 2, etc.).

Definición

Es una colección de elementos de datos caracterizados por las operaciones de acceso de datos caracterizados por las operaciones de acceso usadas para almacenar y recuperar componentes.

Las estructuras de datos están definidas por:

Colocación lógica de los componentes y el conjunto de operaciones para acceder a los datos.

Una estructura de datos es una colección de datos organizados de un modo particular.

Las estructuras de datos pueden ser de dos tipos: estructuras de datos estáticas y estructuras de datos dinámicas o bien estructuras lineales o no lineales.

Las estructuras de datos estáticas son aquellas en las que se asigna una cantidad fija de memoria cuando se declara la variable, en numerosas ocasiones se necesitan, colecciones de datos que crezcan y reduzcan su tamaño en memoria a medida que el programa progresa, a estas estructuras de datos cuya ocupación en memoria puede aumentar o disminuir en tiempo de ejecución se denominan estructuras dinámicas de datos.

Son Estructuras Lineales en los que los distintos elementos se sitúan en línea, de ahí su nombre. Cada elemento de una estructura lineal, excepto el primero y el último, tienen un anterior y un siguiente.
Las estructuras lineales son las Listas, Pilas y Colas. La diferencia entre estos tres tipos de estructuras lineales es el modo de acceso a los elementos que la integran.

En una lista, los elementos pueden añadirse y borrarse en cualquier posición.
En una cola, los elementos pueden añadirse por un extremo y borrarse por el otro.
En una pila los elementos sólo pueden añadirse y borrarse por uno de sus extremos.

El componente básico de una estructura de datos es la celda. Se puede representar una celda como una caja capaz de almacenar un valor tomado de algún tipo de datos básico o compuesto, las estructuras de datos se crean dando nombres a agregados de celdas

El mecanismo de agregación más sencillo en C y en la mayor parte de los lenguajes de programación es el arreglo (unidimensional), que es una sucesión de celdas de un tipo dado al cual se llamará casi siempre <<tipo_celda>> .

PILA

Estructuras secuenciales

Las estructuras lineales más sencillas son aquellas en las que la información se encuentra en bloques contiguos o en un caso extremo, dividida en varios bloques contiguos, frecuentemente llamados segmentos, tal tipo de estructura recibe el nombre de secuencial. En general, en las estructuras secuenciales los elementos son de tamaño fijo, y la dirección de un elemento se puede obtener dada su posición relativa. Si cada elemento ocupa n bytes, la dirección del elemento i se obtiene mediante la expresión $L_0 + n_i$ donde L_0 es la dirección base, esto es, la dirección del primer elemento de la estructura y n_i recibe el nombre de desplazamiento.

El tipo de Dato Abstracto PILA

Una pila es un tipo especial de Lista en la que todas las inserciones y supresiones tienen lugar en un extremo denominado tope. Son estructuras LIFO (Last In First Out) o último en entrar, primero en salir en las que solo es posible quitar el elemento que se encuentra en la parte superior de la pila (tope), son muy útiles en muchos aspectos de la programación, tales como evaluación de expresiones, anidación de parentisis, asi como en la implementación de rutinas

recursivas, entre muchas otras aplicaciones. Una estructura de este tipo generalmente incluye las Operaciones de

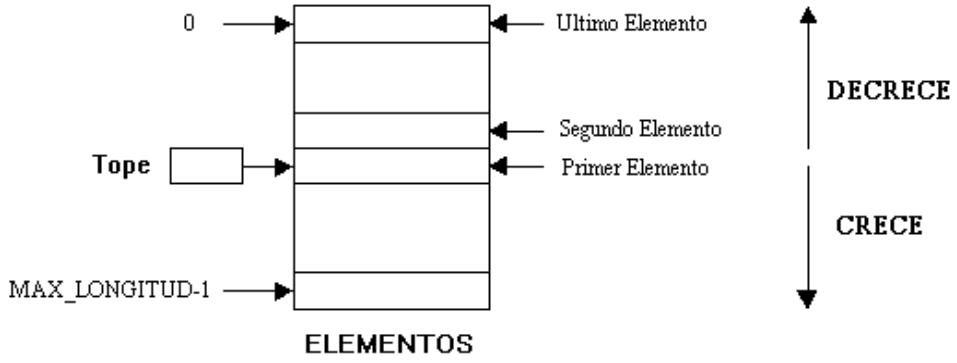
INSERCION. O bien son usados otro sinónimos como Agregar,
ELIMINACION. O bien son usados sinónimos como Borrar,Sacar,pop

Pero en forma integral se siguiere trabajar con las siguientes operaciones:

1. ANULA(P). Que convierte la pila en una Pila Vacía.
2. TOPE(P). Devuelve el elemento que está en el tope de la Pila.
3. POP(P). Suprime el elemento que está en el tope de la pila P.
4. PUSH(x, P). Inserta el elemento x en la parte superior de la Pila.
5. VACIA(P). Devuelve verdadero si la Pila está vacía y falso en caso contrario.
6. LLENA(P). Regresa verdadero si la Pila está llena y falso si no lo está.

Implementación de Pilas basadas en Arreglos

Para representar una Pila con arreglos es posible hacerlo de manera similar como se hizo con las Listas con arreglos, establecemos la Pila como un registro don dos campos, el primero un arreglo para contener los datos de la Pila y un campo para almacenar la posición del elemento superior de la pila (que en lo sucesivo llamaremos tope).



Código en C

```
#define  TAM 5          //Se define el tamaño del arreglo estatico

typedef char tipo_elem;    // Se define el tipo de dato

typedef struct{
    tipo_elem elemento[TAM];
    int tope;
    } Pila;
```

```
void inicializa(Pila *P){
   P->tope = -1; // Se crea una pia
}
tipo_elem tope(Pila *P){
    if (vacia(P))
         return printf("Error la pila esta vacía");
    else
         return P->elemento[P->tope];
}

void pop(Pila *P){
   if (vacia(P))
      printf("Error, la pila esta vacia");
   else
      P->tope--;
}

void push(tipo_elem x,Pila *P){
   if (llena(P))
      printf("Error, la pila esta llena");
   else
      {
      P->tope++;
      P->elemento[P->tope]=x;
      }
}

int vacia(Pila *P){
   if (P->tope == -1)    return 1;
    else   return 0;
  // o return P->tope == -1;
}

int llena(Pila *P){
   if (P->tope == TAM-1) return 1;
   else return 0;
}
```

Implementación de Pilas dinámicas

calloc y malloc

La función malloc es empleada comúnmente para reservar una porción de memoria, dicha porción es contigua. Se define en C como:

```
void *malloc(size_t size);
```

Lo anterior indica que regresará un apuntador del tipo `void *`, el cual es el inicio en memoria de la porción reservada de un tamaño `size`. Si no puede reservar esa cantidad de memoria la función regresa un apuntador nulo o NULL

Como `void *` es regresado, C asume que el apuntador puede ser convertido a cualquier tipo. El tipo de argumento `size_t` esta definido en la cabecera stddef.h y es un tipo entero sin signo.

Ejemplo 1:

```
char *cp;
cp = (char *) malloc(100);
```

Ejemplo 2:

```
int *ip;
ip = (int *) malloc(100 * sizeof(int) );
```

El compilador de C requiere hacer una conversión del tipo, por lo cual se realiza un cast `(int *)`, que permite convertir un apuntador void a un apuntador tipo char e int respectivamente. Hacer la conversión al tipo de apuntador correcto asegura que la aritmética con el apuntador funcionará de forma correcta.

Es una buena práctica usar `sizeof()` aún si se conoce el tamaño actual del dato que se requiere, esta función es usada para encontrar el tamaño de cualquier tipo de dato, variable o estructura.

Cuando se usa la función malloc() la memoria no es inicializada (a cero) o borrada. Si se quiere inicializar la memoria entonces se puede usar la función calloc. La función calloc es computacionalmente un poco más "cara" pero, ocasionalmente, más conveniente que malloc. Se debe observar también la diferencia de sintaxis entre calloc y malloc, ya que calloc toma el número de elementos deseados (nmemb) y el tamaño del elemento (size), como dos argumentos individuales.

Por lo tanto para asignar a 100 elementos enteros que estén inicializados a cero se puede hacer:

```
int *ip;
ip = (int *) calloc(100, sizeof(int) );
```

La declaración para un nodo o celda que forma parte de una pila es declarada como:

```
typedef struct {
    int valor;
    struct ELEMENTO *sig;
} ELEMENTO;
```

o bien

```
typedef struct _nodo {
   int dato;
   struct _nodo *siguiente;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Pila;
```

donde:
tipoNodo es el tipo para declarar nodos.
pNodo es el tipo para declarar punteros a un nodo.
Pila es el tipo para declarar pilas.

EVALUACIÓN DE EXPRESIONES ARITMÉTICAS MEDIANTE PILAS

Conteo de paréntesis

Recordar que las expresiones se componen de operandos, operadores y delimitadores. Los operandos son valores numéricos que se utilizan para calcular la expresión. Los operadores indican las operaciones matemáticas que van hacerse sobre los operandos respectivos. También determinan la cantidad de operandos necesarios para cada tipo de operación (binarios y unarios). A continuación se presentan dos aplicaciones de pilas que trabajan con las expresiones aritméticas

Suponer la siguiente expresión ((5+6)*4)/(17+9), una de las condiciones para que sea una expresión aritmética sea correcta o este bien formada en que tengas sus paréntesis balanceados, así que se debe saber si el número de paréntesis que abres es el mismo número de paréntesis que cierran.
Para resolver este problema es útil usar la estructura de pila. Un algoritmo de solución es presentado a continuacion:

La idea es leer cada elemento de la expresión, si se trata de un paréntesis que abre, entonces se insertar en una pila; si se trata de un paréntesis que cierra, entonces se sacamos un elemento de la pila. Al terminar de leer la expresión se revisa si la pila está vacía, en cuyo caso se habrá concluído que el número de paréntesis que abre es el mismo que el que cierra y la expresión tiene paréntesis balanceados.

Ejemplo:

‘(’ : push(‘(’,P)
‘(’ : push(‘(’,P)
‘5’ : nada que hacer
‘+’ : nada que hacer
‘6’ : nada que hacer
‘)’ : v=borraTope(P)
‘*’ : nada que hacer
‘4’ : nada que hacer
‘)’ : v=borraTope(P)
‘/’ : nada que hacer
‘(’ : push(‘(’,P)
‘17’: nada que hacer
‘+’ : nada que hacer
‘9’ : nada que hacer
‘)’ : v=borraTope(S)

Se comienza con un contador que se inicializa en 0, y por cada push se aumenta un contador, y por cada pop se decrementa el contador. Al final se evalúa el valor del contador, si el contador=0 entonces es éxito, en caso contrario se indica error.

Notación postfija o notación Polaca

El orden en que se calculan las operaciones puede ser muy importante, como en la expresión 6 + 4/2. Si se resuelve como (6+4)/2, la respuesta es 5, si se hace como 6 + (4/2), el resultado es 8.

Los operadores tienen su precedencia, por ejemplo:

Operador	Valor
pow	3
*,/	2
+,-	1

Dada una cierta expresión algebraica, existen básicamente tres formas diferentes de escribirla, notación prefija, notación infija y notación postfija, en función de la situación concreta en la que se pongan los operadores respecto de los operandos. Así la expresión algebraica que representa la suma entre un cierto valor A y otro B se podría poner de la siguiente forma:

+ A B Notación prefija
A + B Notación infija
A B + Notación postfija

La notación utilizada habitualmente es la infija.

Para cambiar el orden de cálculo de una expresión, pueden utilizarse paréntisis pero en su ausencia las operaciones de mayor precedencia se resuelven primero. Cuando una expresión incluye operaciones de igual precedencia, se calculan de izquierda a derecha.

Al utilizar las reglas de precedencia de operadores, se pueden convertir expresiones infijas a la notación polaca correspondiente, los pasos que se siguen son:

1. Encerrar entre paréntisis toda la expresión infija.
2. Volver a colocar (mover) los operadores, uno por uno y en orden de precedencia, a su posición final en notación postfija (a la derecha de sus operandos).
3. Quitar los paréntisis.

Ejemplo 1

Convertir la expresión a + b X c a notación postfija. Observar el algoritmo con respecto al del ejemplo2.

El primer paso es agregar los paréntesis:

a + (bXc)

Después se colocan los operadores en orden de precedencia, así el primero que debemos mover es el signo de multiplicación, X, para que la expresión resultante sea de este modo:

a + (bcX)

Los dos operandos del operador X son b y c, por lo que es fácil determinar la posición posfija de ese signo, pero cuales son los dos operandos del operador +?, la respuesta es a y el resultado de la subexpresión (bXc). Y ponemos el operador + después del parentisis de cierre:

a(bcX)+

El paso final es quitar el paréntesis

a b c X +

Ahora, usando paréntesis, vamos a cambiar el orden del cálculo de los operandos y a convertir la expresión (a + b) X c en notación polaca:

(a + b) X c expresión infija
(a + b) X c se añaden paréntesis sin cambio

(ab +)X c se convirtió el +
(ab +) c X se convirtió el X
ab + c X se eliminó el paréntesis

Ejemplo 2
Convertir la expresión infija A + B * C - D a postfija.

Se inicia la pila a vacía (pila = <>.)

Se obtiene el primer elemento de la expresión (A). Como es un operando, pasa automaticamente a la salida (Salida = "A".)

Se toma el siguiente elemento (+). Como es un operador, se compara su prioridad con la del último operador de apilado en la pila. Como la pila está vacía, el operador se apila en la pila (pila = <+>.)

El siguiente elemento es la B, un operando. Como antes pasa directamente a la salida (Salida = "A B".)

Ahora el elemento es *, un operador. Se compara su prioridad con la del operador de la cima de la pila. Como el * es más prioritario que el +, se apila, en espera de obtener la otra parte de la operación (pila = <+, *>.)

El siguiente elemento es C. Va directamente a la salida (Salida = " A B C".)

Luego se tiene el –. Se compara su prioridad con la cima de la pila. El '–' tiene menor prioridad que el *, luego desapilar el asterisco, que va a la salida (pila = <+>, Salida = "A B C *"). Comparar la prioridad del '–' ahora con la de la cima de la pila (que es el '+'.) Como la prioridad es la misma, desapilar el + y va a la salida (Salida = "A B C * +".) La pila esta vacía con lo que no se puede seguir comparando y se apila el '–' (pila = <–>.)

El siguiente elemento de la expresión es la D. Va directamente a la salida (Salida = "A B C * + D")

Como ya no hay más elementos en la expresión, entonces solo se desapilan todos los elementos restantes de la pila y se ponen en la salida (pila = <>, Salida = "A B C * + D –".)

El proceso con paréntesis sería similar, pero teniendo en cuenta que un parentesis abierto nunca tiene precedencia sobre ningún elemento, de manera que siempre se apila, y el parentesis cerrado desapila todos los símbolos hasta encontrar el parentesis abierto, que también se desapila.

Los paréntesis, no deben salir ya en la expresión postfija.

Algoritmo de Evaluación de una expresión algebraica en notación postfija

Una vez transformada la expresión a notación postfija se realizará un algoritmo que la evalue y dé su resultado. La idea básica del algoritmo es ir apilando los operandos y resultados parciales en una pila e ir deapilándolos a medida que van siendo necesarios (cuando encontremos en la expresión un cierto operador.)

Ejemplo:
Suponer la expresión "13 7 3 * + 5 –" (en infija sería "13 + 7 * 3 – 5" que tiene como resultado 29.)
La manera de evaluarla será:

Iniciar la pila a vacía (pila = <>)

Obtener el primer valor.

Como es un entero, se apila (pila = <13>)

Obtener el segundo elemento y el tercero, que también son enteros y se apilan (pila = <13, 7, 3>) .

Obtener el siguiente elemento que es un '*'. Como es un operador, se necesitan obtener los dos operandos involucrados que son los dos últimos de la pila, el 3 y el 7 que desapilamos. Se operamos con el * y se obtienes 21. El valor 21 se apila (pila = <13, 21>) .

El siguiente elemento es el +. Necesita de nuevo dos operandos que serán los dos últimos de la pila, el 21 y el 13, que se desapilan. Se operan con + y hay que apilar el resultado (pila = <34>).

Después obtener el 5 en la expresión. Se apila (pila = <34, 5>) .

Finalmente se obtiene de la expresión el '–'. Desafilar el 5 y el 34, se resta (34-5=29) y se apila el resultado (pila = <29>) .

Como ya se llegó al final de la expresión, el resultado de la operación es lo que esté en la cima de la pila.

Si en algún momento ya no hay operandos en la pila cuando es necesario desapilar dos elementos, se ha producido un error.

Si al finalizar el proceso, queda más de un elemento en la pila también se ha producido un error. En ambos casos la expresión estaba mal escrita.

Recursión con pilas

El concepto de pila es muy importante en computación y en especial en teoría de lenguajes de programación. En lenguajes procedurales como Pascal o C, la pila es una estructura indispensable, debido a las llamadas a función. Resulta que el flujo de instrucciones va de arriba hacia abajo, y cuando ocurre una llamada a alguna funci ón, el estado global del sistema se almacena en un registro y éste en una pila. Así que la pila va a contener todas las llamadas a procedimientos que se hagan.

Cuando se termina de ejecutar algún procedimiento o función, se recupera el registro que está en la cima de la pila. En ese registro están los valores de las variables como estaban antes de la llamada a la función, o algunas pueden haber cambiado su valor, dependiendo del ámbito de las variables. Cada elemento en la pila que es retirado, significa que se ha terminado de ejecutar alguna función. Cuando se termina de ejecutar el programa, la pila de llamadas a subprogramas debe haber quedado en 0 también, de otro modo podría causar algún tipo de error.
Esto nos lleva a pensar en otras utilidades de la pila. La pila sirve para encontrar errores.

Llamadas a subprogramas

Cuando dentro de un programa se realizan llamadas a subprogramas, el programa principal debe de recordar el lugar donde se hizo la llamada

Ejemplo:
Invocación de funciones usando pilas

```
main()
{
  int i = 5;
  foo(i);
}

foo(int j) {
  int k;
  k = j+1;
  bar(k);
}


bar(int m) {
...
}
```

```
bar
   PC = 1
   m = 6


foo
  PC = 3
  j  = 5
  k = 6


main
  PC = 2
  i = 5
```

- Cuando se invoca un método, se inserta en la pila una nueva entrada.
- Por cada método se guardan:
  variables locales
  valor devuelto
  contador de programa.
- Cuando el método finaliza se saca ese elemento de la pila y se devuelve el control al método que esté en la cabecera.