Universidad Autónoma del Estado de México

"2016, Año del 60 Aniversario de la Universidad Autónoma del Estado de México"

Unidad Académica Profesional Tianguistenco

Ingeniería de Software

Estructura de Datos

Unidad II Pilas

M.C. Angélica Millán Díaz

Semestre 2016J

2.1 Definición

Estructuras de datos

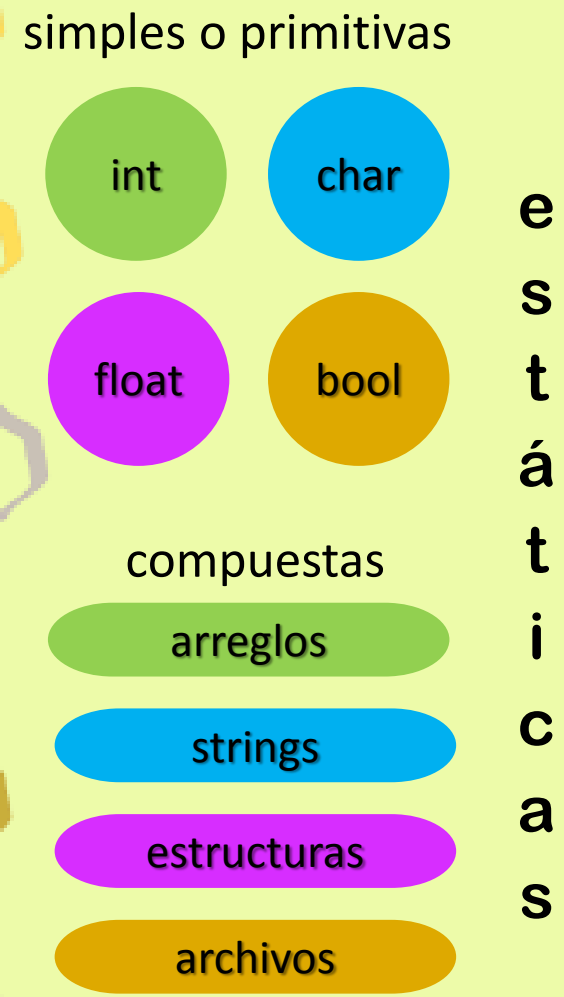
2.1 Definición

El tamaño ocupado en la memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa

estáticas

Estructuras de datos

2.1 Definición



estáticas

Estructuras de datos

2.1 Definición

Estructuras de datos

estáticas

simples o primitivas

- int
- char
- float
- bool

compuestas

- arreglos
- strings
- estructuras
- archivos

dinámicas

No tienen limitaciones en el tamaño de la memoria ocupado, utilizan los apuntadores para construir estructuras de datos dinámicas. Con un solo nombre se hace referencia a un grupo de casillas de la memoria.

2.1 Definición

Estructuras de datos

estáticas

simples o primitivas

- int
- char
- float
- bool

compuestas

- arreglos
- strings
- estructuras
- archivos

dinámicas

lineales

- pila
- cola
- lista

no lineales

- árboles
- grafos

2.1 Definición

lista

Organiza la información como un vector, pero no es necesario conocer la dimensión del mismo.

A cada uno de los elementos de la lista se le llaman **nodos**.

Las operaciones básicas a realizar con las listas son: insertar, eliminar, buscar, recorrer, vaciar y determinar el tamaño de la lista.

pila

cola

2.1 Definición

lista

Organiza la información como un vector, pero no es necesario conocer la dimensión del mismo.

A cada uno de los elementos de la lista se le llaman **nodos**.

Las operaciones básicas a realizar con las listas son: insertar, eliminar, buscar, recorrer, vaciar y determinar el tamaño de la lista.

pila

Estructura de datos dónde el último elemento en entrar es el primero en salir (LIFO).

Solo se tiene acceso a la cabeza o cima de la pila.

Las operaciones básicas a realizar son: insertar (push), eliminar (pop), vaciar la pila y llenar.

cola

2.1 Definición

lista

Organiza la información como un vector, pero no es necesario conocer la dimensión del mismo.

A cada uno de los elementos de la lista se le llaman **nodos**.

Las operaciones básicas a realizar con las listas son: insertar, eliminar, buscar, recorrer, vaciar y determinar el tamaño de la lista.

pila

Estructura de datos dónde el último elemento en entrar es el primero en salir (LIFO).

Solo se tiene acceso a la cabeza o cima de la pila.

Las operaciones básicas a realizar son: insertar (push), eliminar (pop), vaciar la pila y llenar.

cola

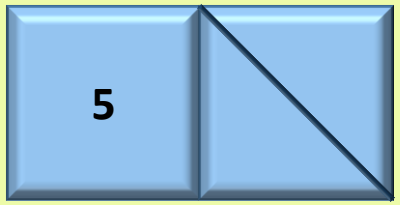
Estructura de datos dónde el primer elemento en entrar es el primero en salir (FIFO).

Una cola es una lista enlazada con acceso FIFO a la que solo se tiene acceso al final de la lista ara meter los elementos y al principio para sacarlos.

Las operaciones básicas a realizar son: insertar, eliminar, vaciar y llenar.

2.1 Definición

```
//declaración de una estructura
struct nodo{
        int num;  //cada elemento de la lista esta formado por un número
        struct nodo *sig; //apuntador al siguiente nodo
};
```

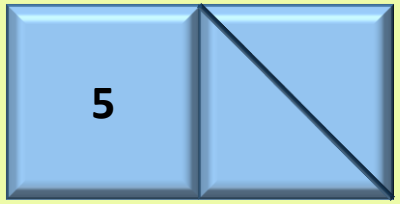


num *sig

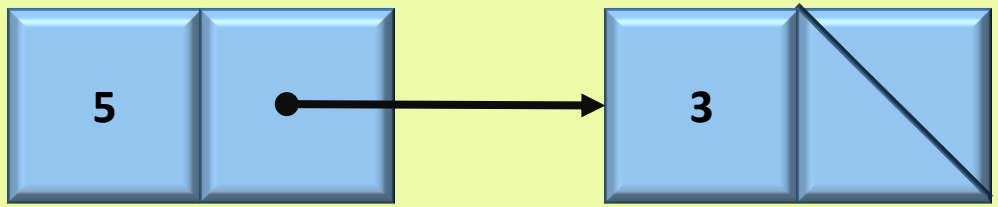
Ésta estructura contiene 2 miembros, num y *sig, éste último apunta a una estructura del mismo tipo en el cual se encuentra definido. *sig hace referencia a un enlace entre cada uno de los elementos de la lista.

2.1 Definición

```
//declaración de una estructura
struct nodo{
        int num;  //cada elemento de la lista esta formado por un número
        struct nodo *sig; //apuntador al siguiente nodo
};
```



num *sig



En esta representación gráfica, para indicar que el último elemento de la lista no apunta hacia otro dato – es decir, apunta a nulo (NULL)- se indica con una diagonal.

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa, con la finalidad de que se puedan crear nodos, así como para liberar espacio que ya no se necesita. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para liberar espacio que ya no se necesita. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

malloc

Toma como argumento el número bytes que serán asignados y regresa un puntero de tipo void* (puntero a nulo) de la memoria asignada.

sizeof

free

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

malloc

Toma como argumento el número bytes que serán asignados y regresa un puntero de tipo void* (puntero a nulo) de la memoria asignada.

sizeof

Determina el largo de la estructura en bytes.

free

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

malloc

Toma como argumento el número bytes que serán asignados y regresa un puntero de tipo void* (puntero a nulo) de la memoria asignada.

sizeof

Determina el largo de la estructura en bytes.

free

Es común utilizar ambas funciones para determinar si existe espacio disponible para asignar un nuevo nodo

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

malloc **sizeof** **free**

```
nuevoN = (Nodos *)malloc(sizeof(Nodos));
```

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

malloc **sizeof** **free**

```
nuevoN = (Nodos *)malloc(sizeof(Nodos));
```

Evalúa el tamaño del nodo

para determinar el tamaño en bytes de la estructura Nodos, asigna una nueva área en la memoria de esa cantidad de bytes, y almacena un apuntador de la localidad de memoria en la variable nuevoN. La localidd de memoria no se inicializa. Si no hay memoria disponible, malloc retorna un valor de NULL.

2.1 Definición

Como estos tipos de datos no tienen asignada una determinada cantidad de memoria, es necesario **asegurar que se tendrá suficiente espacio en el tiempo de ejecución del programa**, con la finalidad de que se puedan crear nodos, así como para **liberar espacio que ya no se necesita**. El límite de la asignación de memoria dinámica puede ser tan grande como la cantidad de memoria virtual disponible en el sistema.

malloc

Toma como argumento el número bytes que serán asignados y regresa un puntero de tipo void* (puntero a nulo) de la memoria asignada.

sizeof

Determina el largo de la estructura en bytes.

free

Libera espacio de memoria asignada, con lo cual dicha memoria puede ser reasignada en el futuro.

```
free(pTemp);
```

2.2 Operaciones

Se tiene la siguiente estructura

```
struct nodo{
        int num;
        struct nodo *sig;
};
```

Se denominan los siguientes alias a la estructura y al enlace de la misma

```
typedef struct nodo Nodos; //alias de la estructura nodo
typedef Nodos *nSig; //alias al apuntador del siguiente nodo
```

Se declaran los siguientes prototipos de funciones para la pila

```
void insertar(nSig *finP, int inf);
int eliminar(nSig *finP);
int vacia(nSig finP);
void imprimir(nSig nActual);
void instrucciones(void);
```

2.2 Operaciones

Se tiene la siguiente estructura

```
struct nodo{
        int num;
        struct nodo *sig;
};
```

Se denominan los siguientes alias a la estructura y al enlace de la misma

```
typedef struct nodo Nodos; //alias de la estructura nodo
typedef Nodos *nSig; //alias al apuntador del siguiente nodo
```

Se declaran los siguientes prototipos de funciones para la pila

```
void insertar(nSig *finP, int inf);
int eliminar(nSig *finP);
int vacia(nSig finP);
void imprimir(nSig nActual);
void instrucciones(void);
```

2.2 Operaciones

La función principal contiene los siguientes elementos:

```
int main(){
        nSig pilaA=NULL; //apunta la cima de la pila
        int op,valor;
        instrucciones();
        printf("Que desea hacer? ");
        scanf("%d",&op);
        while(op!=3){//mientras que el usuario no escriba un 3
                switch (op){
                        case 1:{//insertar elementos
                                printf("Numero a insertar: ");
                                scanf("%d",&valor);
                                insertar(&pilaA,valor);
                                imprimir(pilaA);
                                break;
                        }
```

2.2 Operaciones

La función principal contiene los siguientes elementos: (continuación)

```
case 2:{
        if(!vacia(pilaA)){
                printf("El ultimo valor es %d.\n",eliminar(&pilaA));
        }
        imprimir(pilaA);
        break;
        }
default:{
        printf("Opcion no valida");
        instrucciones();
        break;
        }
}
printf("Que desea hacer? ");
scanf("%d",&op);
}
```

2.2 Operaciones

La función principal contiene los siguientes elementos: (continuación)

```
}
printf("Fin de la ejecucion");
        return 0;
}
```

La definición de la función instrucciones sería:

```
void instrucciones(void){
        printf("Acciones a realizar \n"
        "1) Insertar un elemento a la pila\n"
        "2) Eliminar un elemento de la pila\n"
        "3) Salir del programa\n");
}
```

2.2 Operaciones

La definición de la función insertar sería:

```
void insertar(nSig *finP, int inf){
        nSig nuevoN;
        nuevoN = (Nodos *)malloc(sizeof(Nodos));
        if(nuevoN!=NULL){
                nuevoN->num=inf;
                nuevoN->sig=*finP;
                *finP=nuevoN;
        }
        else{
                printf("No se pudo insertar dato. No hay memoria suficiente");
        }
}
```

Crea un nuevo nodo llamando a malloc, asignando la localidad de memoria a nuevoN

2.2 Operaciones

La definición de la función insertar sería:

```
void insertar(nSig *finP, int inf){
        nSig nuevoN;
        nuevoN = (Nodos *)malloc(sizeof(Nodos));
        if(nuevoN!=NULL){
                nuevoN->num=inf;
                nuevoN->sig=*finP;
                *finP=nuevoN;
        }
        else{
                printf("No se pudo insertar dato. No hay memoria suficiente");
        }
}
```

Asigna el apuntador *finP para hacer el enlace al nuevo elemento

2.2 Operaciones

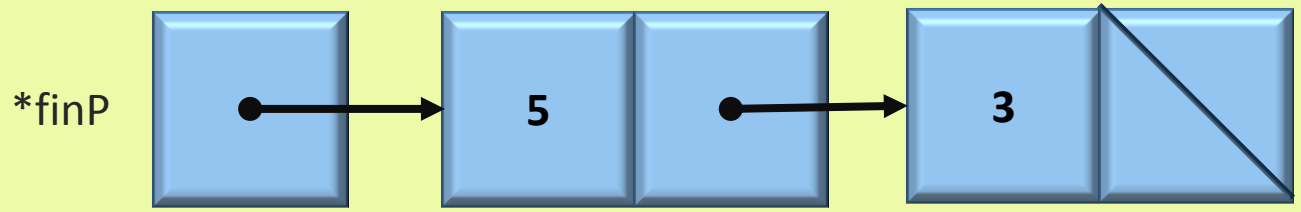
La definición de la función insertar sería:

```
void insertar(nSig *finP, int inf){
        nSig nuevoN;
        nuevoN = (Nodos *)malloc(sizeof(Nodos));
        if(nuevoN!=NULL){
                nuevoN->num=inf;
                nuevoN->sig=*finP;
                *finP=nuevoN;
        }
        else{
                printf("No se pudo insertar dato. No hay memoria suficiente");
        }
}
```

Hace al elemento insertado el inicio de la pila

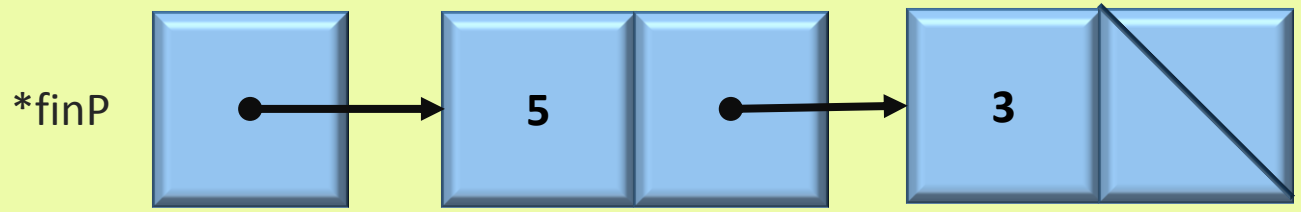
2.2 Operaciones

Lo que realiza la función insertar es:

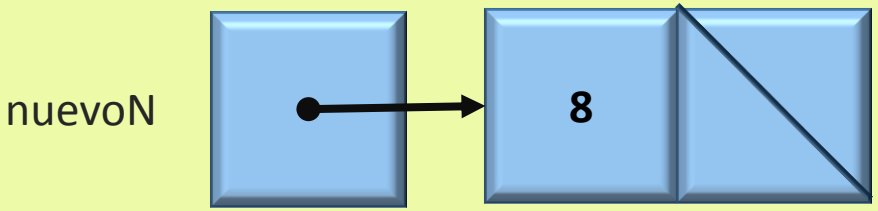


2.2 Operaciones

Lo que realiza la función insertar es:

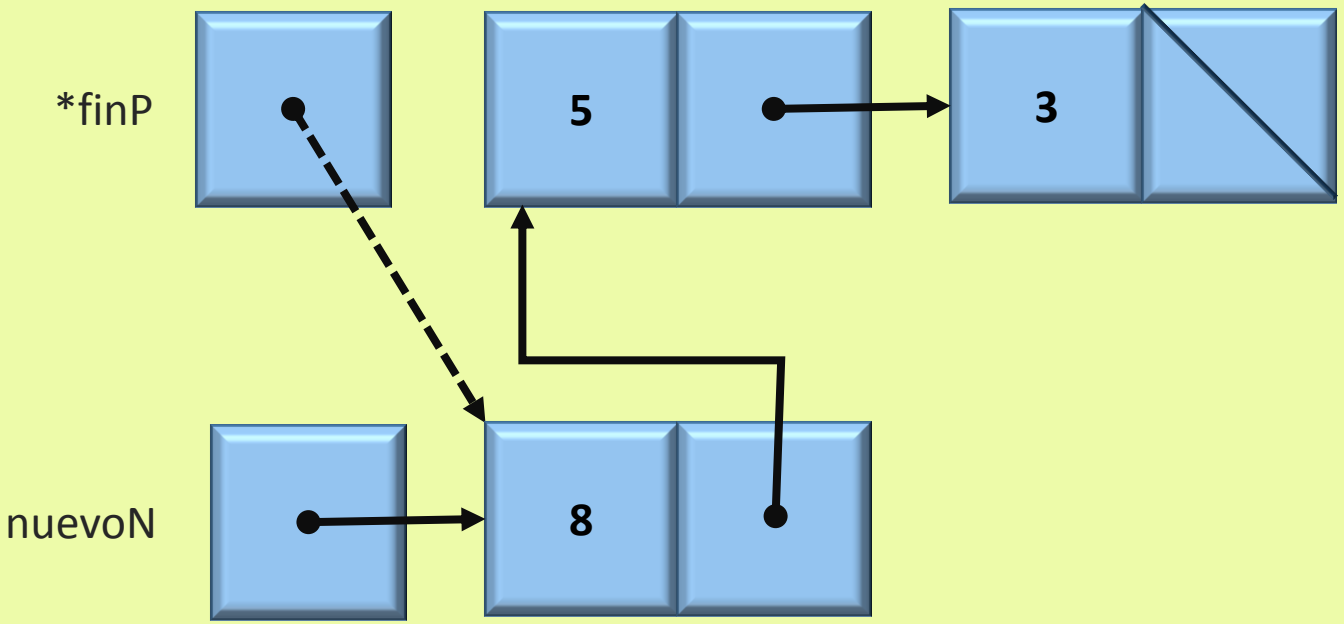


Se inserta un nuevo valor



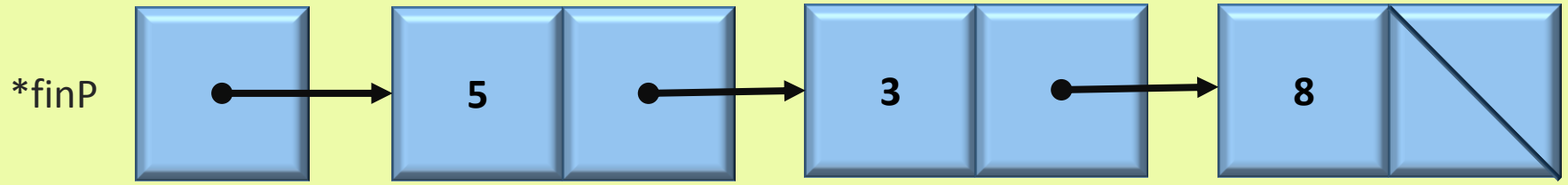
2.2 Operaciones

Lo que realiza la función insertar es:



2.2 Operaciones

Lo que realiza la función insertar es:



2.2 Operaciones

La definición de la función eliminar sería:

```
int eliminar(nSig *finP){
        nSig pTemp;
        int valorN;
        pTemp = *finP;
        valorN=(*finP)->num;
        *finP=(*finP)->sig;
        free(pTemp);
        return valorN;
}
```

2.2 Operaciones

La definición de la función imprimir sería:

```
void imprimir(nSig nActual){
        if(nActual==NULL){
                printf("La pila esta vacia\n");
        }
        else{
                printf("La pila es: \n");
                while(nActual!=NULL){
                        printf("%d-->",nActual->num);
                        nActual=nActual->sig;
                }
                printf("NULL\n\n");
        }
}
```

2.2 Operaciones

La definición de la función vacía sería:

```
int vacia(nSig finP){
        return finP==NULL;
}
```