MANUAL PROYECTO No. 2
MANEJO DE PILAS Y COLAS

Presentado por:

CRISTIAN CAMILO VILLADA GRANADA

STEVEN FLOREZ HERNÁNDEZ

Profesor (a):

CARLOS ALBERTO LONDOÑO

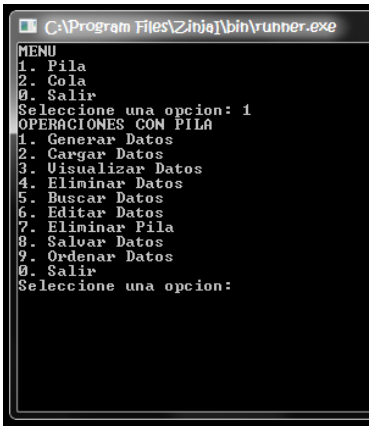
CORPORACIÓN DE ESTUDIOS TECNOLÓGICOS DEL NORTE DEL VALLE

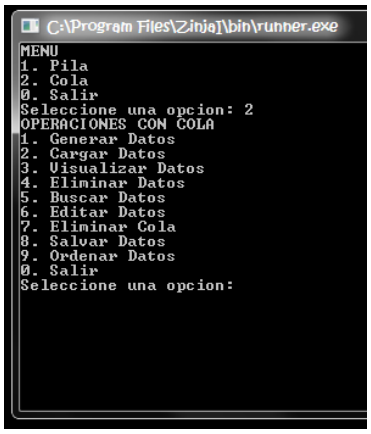
(COTECNOVA)

INGENIERÍA EN SISTEMAS

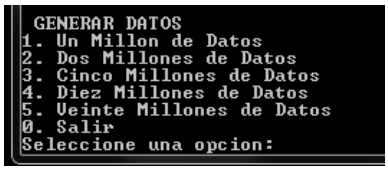
CARTAGO

2017





1. GENERAR DATOS:

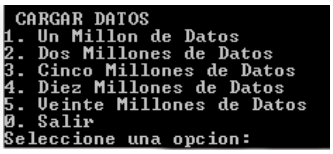


Con esta función vamos a generar los datos los cuales van a estar en las listas (pilas o colas), utilizando un ciclo y una variable opción, se escoge la cantidad de datos con los que se quiere trabajar. Se generan números aleatorios, los cuales se van asignando, por medio de la función insertar datos.

```c
//FUNCION PARA GENERAR DATOS EN PILAS
void generar_datos(int opc , nodo *&pila){
    clock_t iniciar, terminar;
    iniciar = clock();
    int cantidad, num_aleatorio, x;
    contador=1;
    while(opc != 0){
        switch(opc){
        case 1:{
            cantidad = 1000000;
            break;
            }
        case 2:{
            cantidad = 2000000;
            break;
            }
        case 3:{
            cantidad = 5000000;
            break;
            }
        case 4:{
            cantidad = 10000000;
            break;
            }
        case 5:{
            cantidad = 20000000;
            break;
            }
        }break;
    }
    srand(time(NULL));
    for(x = 1;x <= cantidad; x++ ){
        num_aleatorio = (rand() % (1999999 + 1)- 999999);
        //printf("%d %d \n",contador,num_aleatorio);
        insertar_datos_pila(num_aleatorio, pila);
        contador++;
    }
    printf("datos generados \n");
    terminar = clock();
    printf("El tiempo gastado en esta funcion fue de: %.0f milisegundos\n", (double)(terminar - iniciar));
}
```

```c
//FUNCION PARA GENERAR DATOS COLAS
void generar_datos_cola(int opc, nodo *&inicio, nodo *&fin){
    clock_t iniciar, terminar;
    iniciar = clock();
    int cantidad, aleatorio, i;
    contador=1;
    while(opc != 0){
        switch(opc){
        case 1:{
            cantidad = 10;
            break;
            }
        case 2:{
            cantidad = 2000000;
            break;
            }
        case 3:{
            cantidad = 5000000;
            break;
            }
        case 4:{
            cantidad = 10000000;
            break;
            }
        case 5:{
            cantidad = 20000000;
            break;
            }
        }break;
    }
    srand(time(NULL));
    for(i = 1; i <= cantidad; i++){
        aleatorio = (rand() % (1999999 + 1)- 1000000);
        //printf("%d %d \n", contador, aleatorio);
        insertar_datos_cola(aleatorio, inicio, fin);
        contador++;
    }
    terminar = clock();
    printf("El tiempo gastado en esta funcion fue de: %.0f milisegundos\n", (double)(terminar - iniciar));
    printf("\n");
}
```

2. CARGAR DATOS



Con esta función tenemos la posibilidad de cargar un número determinado de datos desde un archivo de texto con datos ya generados, el número de datos será escogido por medio de un ciclo y una variable opción. Cada uno de estos datos se asignara a la lista por medio de la función insertar datos.

```c
//FUNCION PARA CARGAR DATOS DE UN ARCHIVO
void cargar_datos(int opc , nodo *&pila){
    clock_t iniciar,terminar;
    iniciar = clock();
    FILE *archivo;
    int dato;
    contador=1;
    while(opc != 0){
        switch(opc){
        case 1:{
            archivo = fopen("un_millon.txt","rb");
            break;
            }
        case 2:{
            archivo = fopen("dos_millones.txt","rb");
            break;
            }
        case 3:{
            archivo = fopen("cinco_millones.txt","rb");
            break;
            }
        case 4:{
            archivo = fopen("diez_millones.txt","rb");
            break;
            }
        case 5:{
            archivo = fopen("veinte_millones.txt","rb");
            break;
            }
        }break;
    }
    while(fread(&dato,sizeof(dato),1,archivo)){
        insertar_datos_pila(dato, pila);
    }
    terminar = clock();
    printf("el tiempo de duracion es: %f \n",(double)(terminar - iniciar));
}
```

```c
//FUNCION PARA CARGAR DATOS A LA COLA DESCDE UN ARCHIVO DE TEXTO
void cargar_datos_cola(int opc, nodo *&inicio, nodo *&fin){
    clock_t iniciar, terminar;
    iniciar = clock();
    FILE * archivo;
    int dato;
    contador=1;
    while(opc != 0){
        switch(opc){
        case 1:{
            archivo = fopen("un_millon.txt", "rb");
            break;
            }
        case 2:{
            archivo = fopen("dos_millones.txt", "rb");
            break;
            }
        case 3:{
            archivo = fopen("cinco_millones.txt", "rb");
            break;
            }
        case 4:{
            archivo = fopen("diez_millones.txt", "rb");
            break;
            }
        case 5:{
            archivo = fopen("veinte_millones.txt", "rb");
            break;
            }
        }break;
    }
    while(fread(&dato, sizeof(dato), 1, archivo)){
        insertar_datos_cola(dato, inicio, fin);
    }
    terminar = clock();
    printf("El tiempo empleado fue de: %.0f milisegundos\n", (double)(terminar - iniciar));
    printf("\n");
}
```

3. INSERTAR DATOS

Esta función es la que se usa para insertar los datos en la lista, aquí es donde se reserva memoria para el nuevo nodo; en el caso de las pilas se le asigna un número o dato, y un apuntador al siguiente dato.

```c
//FUNCION PARA INSERTAR LOS DATOS EN LA PILA
void insertar_datos_pila(int num_aleatorio , nodo *& pila){
    nodo * nuevo_nodo = reservar_memoria;
    nuevo_nodo -> dato = num_aleatorio;
    nuevo_nodo -> siguiente = pila;
    pila = nuevo_nodo;
}
```

En el caso de las colas, el siguiente debe apuntar a NULL, y se comprueba si inicio = NULL, se le asigna el nuevo nodo, o si este se le asigna al siguiente de fin.

```c
//FUNCION PARA INSERTAR DATOS A LA COLA
void insertar_datos_cola(int aleatorio, nodo *&inicio, nodo *&fin){
    nodo * nuevo_nodo = reservar_memoria;
    nuevo_nodo -> dato = aleatorio;
    nuevo_nodo -> siguiente = NULL;
    if(inicio == NULL){
        inicio = nuevo_nodo;
    }else{
        fin -> siguiente = nuevo_nodo;
    }
    fin = nuevo_nodo;
}
```

4. VISUALIZAR LOS DATOS

   Aquí mostramos todos los datos que se encuentran en las listas, primero se comprueba si se encuentra vacía, y después recorremos por medio de un ciclo para realizar la impresión de los datos.

```c
//FUNCION PARA VISUALIZAR DATOS
void visualizar_datos(nodo *pila){
    clock_t iniciar, terminar;
    iniciar = clock();
    nodo *auxiliar = pila;
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
        while(auxiliar != NULL){
            printf("%d", auxiliar -> dato);
            auxiliar = auxiliar -> siguiente;
            printf(" ");
        }
        printf("\n");
    }
    terminar = clock();
    printf("el tiempo de duracion es: %f \n",(double)(terminar-iniciar));
}
```

```c
//FUNCION PARA VISUALIZAR LOS DATOS DE LA COLA
void visualizar_datos_cola(nodo *inicio){
    nodo * aux = inicio;
    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        while(aux != NULL){
            printf("%d ", aux -> dato);
            aux = aux -> siguiente;
        }
    }
    printf("\n");
}
```

5. ELIMINAR DATOS

   Esta función se encarga de eliminar un dato de cada una de las listas, según sea el caso, el último que ingresa en una pila, y el primero en una cola. En la pila creamos un nodo auxiliar, el cual lo borramos de la memoria al finalizar el ciclo, y en la cola solo eliminamos el nodo fin.

```c
//FUNCION PARA ELIMINAR DATOS
void eliminar_datos(nodo *&pila){
    int b;
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
        nodo *auxiliar = pila;
        pila = auxiliar->siguiente;
        b = auxiliar->dato;
        free(auxiliar);
        contaux=contador--;
        printf("Elemento eliminado \n");
    }
}
```

```c
//FUNCION PARA ELIMINAR LOS DATOS DE LA COLA
void eliminar_datos_cola(nodo *&inicio, nodo *&fin){
    clock_t iniciar, terminar;
    iniciar = clock();
    int b;

    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        fin = inicio;
        inicio = fin -> siguiente;
        free(fin);
        printf("Elemento eliminado\n");
    }
    terminar = clock();
    printf("El tiempo gastado fue de: %.0f milisegundos\n", (double)(terminar - iniciar));
    printf("\n");
}
```

6. BUSCAR DATOS

   En esta función digitamos el número del dato que deseemos buscar en la lista, esta nos dirá si la lista está vacía, si el dato no existe, o por el contrario si fue encontrado el dato, en qué posición se encuentra y su dirección de memoria.

```c
//FUNCION PARA BUSCAR DATOS
void buscar_datos(nodo *pila){
    clock_t iniciar, terminar;
    iniciar = clock();
    int b,contaux=contador-1,encontrado=0;
    printf("Digite el elemento a buscar \n");
    scanf("%d", &b);
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
        while(pila != NULL){
            if(b == pila->dato){
                printf("Dato encontrado en la posicion %d y su direccion de memoria es : %p \n" ,contaux,&pila->dato );
                encontrado=1;
                break;
            }pila = pila->siguiente;
                contaux--;
            }
            if(encontrado!=1){
                printf("dato no encontrado \n");
            }
        }terminar = clock();
    printf("el tiempo de duracion es: %f \n",(double)(terminar-iniciar));
}
```

```c
//FUNCION PARA BUSCAR DATOS EN LA COLA
void buscar_datos_cola(nodo *inicio){
    clock_t iniciar, terminar;
    iniciar = clock();
    int b, contador=1,encontrado=0;
    printf("Digite el elemento a buscar \n");
    scanf("%d", &b);
    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        while(inicio != NULL){
            if(b == inicio->dato){
                printf("Dato encontrado en la posicion %d y su direccion de memoria es : %p \n" , contador, &inicio->dato);
                break;
                }
            inicio = inicio->siguiente;
            contador++;
            }
            if(encontrado!=1){
                printf("dato no encontrado \n");
            }
        }
        terminar = clock();
    printf("El tiempo gastado en esta funcion fue de: %.0f milisegundos\n", (double)(terminar - iniciar));
}
```

7. EDITAR DATOS

   En esta función podemos reemplazar o cambiar cualquier dato, se implementa la función buscar datos, y adicional se pregunta si se quiere editar el dato encontrado, se digita el nuevo dato, y este se asigna a la estructura correspondiente, reemplazando el anterior dato, y conservando su posición.

```c
//FUNCION PARA EDITAR DATOS
void editar_datos(nodo *&pila){
    nodo *auxiliar = pila;
    int b, nuevo_dato,opceliminar=0,contaux=contador-1;
    printf("Digite el elemento a buscar \n");
    scanf("%d", &b);
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
        while(auxiliar != NULL){
            if(b == auxiliar->dato){
                printf("Dato encontrado en la posicion %d y su direccion de memoria es : %p \n" ,contaux,&pila->dato );
                printf("desea editar el dato 1.si - 2.no\n");
                scanf("%d",&opceliminar);
                if(opceliminar == 1){
                    printf("Ingrese el nuevo dato\n");
                    scanf("%d", &nuevo_dato);
                    auxiliar -> dato = nuevo_dato;
                    printf("Dato editado correctamente\n");
                }
                    break;
                }auxiliar = auxiliar->siguiente;
                    contaux--;
            }
        }
}
```

```c
//FUNCION PARA EDITAR DATOS EN COLAS
void editar_datos_colas(nodo *&inicio){
    clock_t iniciar, terminar;
    iniciar = clock();
    nodo * auxiliar = inicio;
    int b, nuevo_dato, opc=0, contador=1,encontrado=0;
    printf("Digite el elemento a buscar \n");
    scanf("%d", &b);
    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        while(auxiliar != NULL){
            if(b == auxiliar -> dato){
                printf("Dato encontrado en la posicion %d y su direccion de memoria es : %p \n" , contador, &inicio->dato);
                printf("Desea editar el dato 1.Si - 2.No\n");
                scanf("%d", &opc);
                encontrado=1;
                if(opc == 1){
                    printf("Ingrese el nuevo dato\n");
                    scanf("%d", &nuevo_dato);
                    auxiliar -> dato = nuevo_dato;
                    printf("Dato editado correctamente\n");
                    }
                break;
                }
            auxiliar = auxiliar -> siguiente;
            contador++;
            }
            if(encontrado!=1){
                printf("dato no encontrado \n");
            }
        }
        terminar = clock();
    printf("El tiempo gastado en esta funcion fue de: %.0f segundos\n", (double)(terminar - iniciar));
}
```

8. ELIMINAR PILA – ELIMINAR COLA

   Aquí eliminamos todos los datos que se encuentran en las listas, se usa la misma estructura de la función eliminar dato, se agrega un ciclo el cual recorra toda la lista, y elimine dato por dato, ya que debe ir eliminando memoria.

```c
//FUNCION PARA ELIMINAR PILA
void eliminar_pila(nodo *&pila){
    clock_t iniciar, terminar;
    iniciar = clock();
    int b;
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
    while(pila != NULL){
            nodo *auxiliar = pila;
            pila = auxiliar->siguiente;
            b = auxiliar->dato;
            free(auxiliar);
        }
        printf("Pila eliminada\n");
    }terminar = clock();
    printf("el tiempo de duracion es: %f \n",(double)(terminar-iniciar));
}
```

```c
//FUNCION PARA ELIMINAR LA COLA
void eliminar_cola(nodo *&inicio, nodo *&fin){
    clock_t iniciar, terminar;
    iniciar = clock();
    int b;
    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        while(inicio != NULL){
            fin = inicio;
            inicio = fin -> siguiente;
            free(fin);
        }
        printf("COLA ELIMINADA\n");
    }
    terminar = clock();
    printf("El tiempo de esta funcion fue de: %.0f segundos\n", (double)(terminar - iniciar));
    printf("\n");
}
```

9. SALVAR DATOS

   Esta función crea un archivo de texto en el cual va a guardar todos los datos que tengamos en las listas, en esta recorremos la lista con un ciclo, y vamos guardando dato por dato en el archivo según el orden en el que se encuentren.

```c
//FUNCION PARA SALVAR LOS DATOS DE LA PILA
void salvar_datos(nodo *pila){
    clock_t iniciar, terminar;
    iniciar = clock();
    int numero=0;
    FILE *datos_pila;
    datos_pila=fopen("datos_pila.txt","ab");
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
        while(pila!= NULL){
            fwrite(&pila->dato,sizeof(pila->dato),1,datos_pila);
            pila = pila -> siguiente;
        }
        printf("datos guardados \n");
    }
    fclose(datos_pila);
    terminar = clock();
    printf("el tiempo de duracion es: %f \n",(double)(terminar-iniciar));
}
```

```c
//FUNCION PARA SALVAR LOS DATOS DE LA COLA EN UN ARCHIVO DE TEXTO
void salvar_datos_cola(nodo *&inicio){
    clock_t iniciar, terminar;
    iniciar = clock();
    FILE *datos_cola;
    int aux;
    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        datos_cola=fopen("datos_cola.txt", "a+b");
        while(inicio != NULL){
            aux = inicio ->dato;
            fwrite(&aux, sizeof(aux), 1, datos_cola);
            inicio = inicio ->siguiente;
        }
        fclose(datos_cola);
        printf("DATOS SALVADOS\n");
    }
    printf("El tiempo gastado en esta funcion fue de: %.0f segundos\n", (double)(terminar - iniciar));
    printf("\n");
}
```

10. ORDENAR DATOS

Aquí se implementan dos métodos de ordenamiento, uno ineficiente como es burbuja y otro eficiente que se trató de llevar a cabo, el quick sort. Burbuja es un algoritmo de ordenamiento muy fácil de implementar, en éste solo se tienen que ir reemplazando los datos y asignándolos a variables auxiliares, pero es el más ineficiente ya que tiene una complejidad de O^n , lo que quiere decir que si se tienen 100 datos, será 100^{100} en el número de ciclos que haga para ordenar estos.

```
//FUNCION PARA ORDENAR LOS DATOS: METODO BURBUJA
void ordenar_datos_ineficientemente(nodo *&pila){
    clock_t iniciar, terminar;
    iniciar = clock();
    int aux;
    nodo * nodo_auxiliar_1 = pila;
    nodo * nodo_auxiliar_2 = NULL;
    if(pila_vacia(pila)){
        printf("Pila Vacia \n");
    }else{
        while(nodo_auxiliar_1 != NULL){
            nodo_auxiliar_2 = nodo_auxiliar_1 -> siguiente;
            while(nodo_auxiliar_2 != NULL){
                if(nodo_auxiliar_1 -> dato > nodo_auxiliar_2 -> dato){
                    aux = nodo_auxiliar_1 -> dato;
                    nodo_auxiliar_1 -> dato = nodo_auxiliar_2 -> dato;
                    nodo_auxiliar_2 -> dato = aux;
                }
                nodo_auxiliar_2 = nodo_auxiliar_2 -> siguiente;
            }
            nodo_auxiliar_1 = nodo_auxiliar_1 -> siguiente;
        }
    }
    terminar = clock();
    printf("el tiempo de duracion es: %f \n",(double)(terminar-iniciar));
    printf("datos ordenados\n");
}
```

```
//FUNCION PARA ORDENAR LOS DATOS EN LA COLA: METODO BURBUJA
void ordenar_datos_cola(nodo *inicio){
    clock_t iniciar, terminar;
    iniciar = clock();
    int aux;
    nodo * x = inicio;
    nodo * y = NULL;
    if(inicio == NULL){
        printf("COLA VACIA\n");
    }else{
        while(x != NULL){
            y = x -> siguiente;
            while(y != NULL){
                if(x -> dato > y -> dato){
                    aux = x -> dato;
                    x -> dato = y -> dato;
                    y -> dato = aux;
                }
                y = y -> siguiente;
            }
            x = x -> siguiente;
        }
    }
    printf("El tiempo empleado fue de: %.0f segundos\n", (double)(terminar - iniciar));
    printf("\n");
}
```

El quicksort, es un método más complejo de implementar, pero mucho más eficiente, ordena los datos mucho más rápido con una complejidad de O(n*log n), este algoritmo utiliza un dato pivote, y acomodara los datos mayores a un lado, y los menores a otro, el dato pivote ya tendrá su posición ordenada en la lista, se crean dos sublistas con estos datos, y ya queda mucho más fácil ordenarlas y pasar los datos a la lista original.

```
//FUNCION PARA ORDENAR DATOS: METODO QUICK SORT
void ordenar_datos_eficientemente(nodo *&pila){
nodo *nodo_auxiliar_1=pila;
nodo *nodo_auxiliar_2=NULL;
nodo *nodo_auxiliar_3=NULL;
nodo *copia_nodo_aux1=nodo_auxiliar_1;

int inicio,ultimo,pivote,central,i,j,con_pos=1,aux;
    while(nodo_auxiliar_1 != NULL){
        nodo_auxiliar_1 = nodo_auxiliar_1->siguiente;
        if(nodo_auxiliar_1==NULL)
        {
            ultimo = con_pos;
        }
        con_pos++;
    }
    inicio = 1;
    central = (inicio+ultimo)/2;
    nodo_auxiliar_1 = copia_nodo_aux1;
    while(nodo_auxiliar_1 != NULL)
    {
        int con_pos_aux=con_pos--;
        nodo_auxiliar_1=nodo_auxiliar_1->siguiente;
        if (central == con_pos-1)
        {
            pivote = nodo_auxiliar_1 -> dato;
        }con_pos_aux--;
    }
    nodo_auxiliar_1 = copia_nodo_aux1;
    while(nodo_auxiliar_1!= NULL){
        if(nodo_auxiliar_1->dato > pivote){
            insertar_datos_pila(nodo_auxiliar_1->dato, nodo_auxiliar_2);

        }else{
            insertar_datos_pila(nodo_auxiliar_1->dato, nodo_auxiliar_3);

        }nodo_auxiliar_1=nodo_auxiliar_1->siguiente;
    }
```

```
    }
    nodo_auxiliar_1 = copia_nodo_aux1;
    //se realiza copia de los nodos para que vuelvan a la poscicion inicial
    nodo *copia_nodo_aux2=nodo_auxiliar_2;
    nodo *copia_nodo_aux3=nodo_auxiliar_3;
    while(nodo_auxiliar_3!=NULL){
        nodo_auxiliar_1->dato=nodo_auxiliar_3->dato;
        nodo_auxiliar_3=nodo_auxiliar_3->siguiente;
        nodo_auxiliar_1=nodo_auxiliar_1->siguiente;
    }
    while(nodo_auxiliar_2!=NULL){
        nodo_auxiliar_1->dato=nodo_auxiliar_2->dato;
        nodo_auxiliar_2=nodo_auxiliar_2->siguiente;
        nodo_auxiliar_1=nodo_auxiliar_1->siguiente;
    }

    nodo_auxiliar_1=copia_nodo_aux1;
    nodo_auxiliar_2 = copia_nodo_aux2;
    nodo_auxiliar_3 = copia_nodo_aux3;
    int bandera,tem_aux;
    do{
        bandera=0;
        nodo_auxiliar_1=copia_nodo_aux1;
        for(int i=0;i < con_pos-1;i++){
        aux=nodo_auxiliar_1->siguiente->dato;
        if(nodo_auxiliar_1->dato > aux){
            tem_aux = nodo_auxiliar_1->dato;
            nodo_auxiliar_1->dato = aux;
            nodo_auxiliar_1->siguiente->dato=tem_aux;
            bandera++;
        }
        nodo_auxiliar_1=nodo_auxiliar_1->siguiente;
        }
    }while( bandera > 0);
}
```