Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ciencias de la Computación e Informática

Estructuras de Datos y Análisis de Algoritmos

CI-1221

Grupo 01

Tarea Programada I Etapa 2

Profesora:
Sandra Kikut

Elaborado por:

Acuña Díaz Jimmy B50060
Carvajal Rojas Luis B31494
Tretta Monge Isaac Dario B47040

Día 5 de octubre del 2017

Índice

1. Introducción **3**

2. Objetivos **3**

3. Enunciado (Descripción del Problema) **3**

4. Desarrollo **4**

4.1. Modelos 4

4.1.1. Cola 4

4.1.1.1. Definición 4

4.1.1.2. Operadores Básicos 5

4.1.2. Árbol n-ario 5

4.1.2.1. Definición 5

4.1.2.2. Operadores Básicos 6

4.2. Estructuras de Datos 7

4.2.1. Arreglo Circular 8

4.2.1.1. Diagrama y Descripción 8

4.2.1.2. Definición den C++ 8

4.2.2. Arreglo con Señalador al Padre 9

4.2.2.1. Definición y Diagrama 9

4.2.2.2. Definición en C++ 9

4.2.3. Lista de Hijos 10

4.2.3.1. Definición y Diagrama 10

4.2.3.2. Definición en C++ 11

4.2.4. Hijo más Izq-HermanoDerecho por punteros 11

4.2.4.1. Definición y Diagrama 11

4.2.4.2. Definición en C++ 12

4.2.5. Hijo más Izq-HermanoDerecho con señalador al padre(último hijo) 13

4.2.5.1. Definición y Diagrama 13

4.2.5.2. Definición en C++ 14

4.2.6. Hijo más Izq-HermanoDerecho con señalador al padre y al hermano más izquierdo 15

4.2.6.1. Definición y Diagrama 15

4.2.6.2. Definición en C++ 15

5. Manual de Ususario **16**
5.1. Requerimientos de Hardware 16
5.2. Requerimientos de Software 16
5.3. Arquitectura del Programa 16
5.4. Compilación 16
5.5. Especificación del Programa 18

6. Datos de Prueba **19**
6.1. Formato de los datos de prueba 19
6.2. Salida esperada 19
6.3. Salida obtenida 19

7. Lista de Archivos **19**

8. Referencias o Bibliografía **20**

1. Introducción

Por medio de este trabajo se verá reflejado en sus diferentes etapas los conocimientos adquiridos en el curso de Estructuras de Datos y Análisis de Algoritmos. Es por esta razón que por medio de modelos matemáticos junto con la implementación de algunos algoritmos y por supuesto que el análisis tiempo-espacio estos algoritmos se pondrá en evidencia no solo el conocimiento gracias al curso sino la comprensión y reconocimiento de la importancia de llegar a implementar un modelo determinado con un algoritmo asociado previamente estudiado, ya que sabemos perfectamente que un programador por definición acordada en clases es un programador eficiente, eficaz y capaz de justificar con pruebas fehacientes el por que de una elección, ya que a fin de cuentas, lo que se vaya a implementar dado un problema del mundo real va a requerir una forma distinta de ser realizada dependiendo de como esta se encuentre conformada.

Podemos concluir esta introducción que el aporte más significativo de este trabajo es el de mostrar y afianzar los conocimientos en Estructuras de Datos junto con el análisis de los algoritmos asociados para resolver de forma eficaz y permitiendo al usuario final que el costo en cuanto a tiempo y espacio sea lo menor posible.

2. Objetivos

- Definir, especificar, implementar y usar los modelos lógicos Cola y Árbol n-ario tal que SÍ importa el orden entre los hijos de un nodo.
- Realizar un análisis teórico y un análisis real del tiempo de ejecución de las diferentes estructuras de datos y algoritmos implementados.

3. Enunciado (Descripción del Problema)

1. Definir formalmente el modelo lógico Cola.
2. Especificar de manera lógica, formal y completa los operadores básicos de la Cola. Para cada operador debe incluir: nombre, parámetros con sus tipos y las cláusulas Efecto (claro, completo y conciso), Requiere y Modifica: Crear, Destruir, Vaciar, Vacía?, Encolar, Desencolar, Frente.
3. Definir formalmente el modelo lógico Árbol n-ario, tal que SÍ importa el orden entre los hijos de un nodo. Recuerde que el modelo árbol

se “maneja” por nodos y no por etiquetas y que el nodo es una noción abstracta cuyo tipo depende de la estructura de datos usada para implementar el árbol.

4. Especificar de manera lógica, formal y completa los operadores básicos del Árbol tal que SÍ importa el orden entre los hijos de un nodo. Para cada operador debe incluir: nombre, parámetros con sus tipos y las cláusulas Efecto (claro, completo y conciso), Requiere y Modifica. Nota: Su diseño puede permitir el uso de NodoNulo. Crear, Destruir, Vaciar, Vacío?, PonerRaíz, AgregarHijoIÉsimo?, BorrarHoja, ModificarEtiqueta, Raíz?, Padre?, HijoMásIzquierdo?, Hermano Derecho?, Etiqueta?, NumNodos?, NumHijos?

4. Desarrollo

A continuación se van a desarrollar distintos modelos matemáticos usados en la computación para resolver distintos tipos de problemas, cada modelo tiene su conjunto de operadores básicos y a su vez, cada operador tiene uno o varios requerimientos, efectos, objeto que modifica o tipo de dato que devuelve. Esta sección va a detallar los elementos que conforman al modelo matemático.

4.1. Modelos

Existen distintos modelos matemáticos usados para construir estructuras de datos, empleados en la resolución de problemas del mundo real por medio de la programación. Los modelos que se van a desarrollar son: La cola y el árbol.

4.1.1. Cola

4.1.1.1 Definición

Una cola es una estructura de datos que se caracteriza por ser una sucesión de elementos en la operación ’meter’ o encolar se hacen por un extremo y la operación ’sacar’ o desencolar se hace por el otro extremo. También se le conoce como la estructura FIFO(del inglés First In, First Out), debido a que el primer elemento en entrar será el primero en salir. (.Estructuras de Datos”, 2011)

4.1.1.2 Operadores Básicos

- *Iniciar(C)*
  Efecto: Crea una cola vacía.
  Requiere: —
  Modifica: —

- *Destruir(C)*
  Efecto: Destruye la cola. La deja inutilizable.
  Requiere: —
  Modifica: —

- *Vaciar(C)*
  Efecto: Vacía la cola.
  Requiere: Cola inicializada.
  Modifica: La cola L.

- *Vacía(C)* $\rightarrow$ *bool*
  Efecto: Retorna verdadero si la cola esta vacía.
  Requiere: Cola inicializada.
  Modifica: —

- *Encolar(e,C)*
  Efecto: Mete el elemento e en la cola.
  Requiere: Cola inicializada.
  Modifica: La cola C.

- *Desencolar(e, C)*
  Efecto: Saca el elemento e de la cola.
  Requiere: Cola inicializada.
  Modifica: La cola C.

- *Frente(C)*
  Efecto: Devuelve el elemento que se encuentra en el frente de la cola.
  Requiere: Cola inicializada y no vacía.
  Modifica: —

4.1.2. Árbol n-ario

4.1.2.1 Definición

Es un modelo no lineal que entre cada n cantidad de nodos hay un nodo (padre) en común, se puede visualizar recursivamente como: Un dato y varios

árboles adjuntos. Su principal característica es que requiere de una relación jerárquica entre los elementos que lo conforman.

4.1.2.2 Operadores Básicos

- *Iniciar(A)*
  Efecto: Crea un árbol (A) vacío.
  Requiere: —
  Modifica: —

- *Destruir(A)*
  Efecto: Destruye el árbol (A), lo deja inutilizable.
  Requiere: —
  Modifica: —

- *Vaciar(A)*
  Efecto: Vacia A, se puede volver a utilizar
  Requiere: A inicializado, no vacío.
  Modifica: El árbol A.

- *Vacía(A)* $\rightarrow$ *bool*
  Efecto: Retorna verdadero si A está vacío.
  Requiere: A inicializado.
  Modifica: —

- *agregarHijo(n, e, A)*
  Efecto: Agrega un hijo al nodo n, con la etiqueta e.
  Requiere: A no vacío, y n existente.
  Modifica: El árbol A.

- *borrarHoja(n, A)*
  Efecto: Elimina el n de A.
  Requiere: Que n exista en el árbol. A no vacío.
  Modifica: El árbol A.

- *modEtiqueta(n, e, A)*
  Efecto: Busca n y cambia su etiqueda por e.
  Requiere: n válido y A no vacio.
  Modifica: —

- *Raiz(A)* $\rightarrow$ *nodo*
  Efecto: Retorna un nodo con la raiz de A
  Requiere: A no vacío.
  Modifica: —

- *Padre(n, A) → nodo*
  Efecto: Busca el padre de n.
  Requiere: A válido. A no vacío.
  Modifica: —

- *hijoMasIzquierdo(n, A) → nodo*
  Efecto: Retorna el hijo que esta más a la izquierda de n
  Requiere: A inicializado, no vacío y n válido
  Modifica: —

- *hermanoDerecho(n, A) → nodo*
  Efecto: Retorna al hermano derecho inmediato de n.
  Requiere: n válido, A inicializado no vacío.
  Modifica: —

- *Etiqueta(n, A) → tipo elemento*
  Efecto: Retorna la etiqueta de A.
  Requiere: n válido, A inicializado no vacío.
  Modifica: —

- *numNodos(A) → entero*
  Efecto: Retorna el número de nodos.
  Requiere: A inicializado y un contador.
  Modifica: —

- *numHijos(n, A) → entero*
  Efecto: Retorna la cantidad de hijos que tiene n.
  Requiere: n válido, A inicializado no vacío.
  Modifica: —

4.2. Estructuras de Datos

En esta sección de describirán las estructuras dedatos usadas, cada una con su respectivo diagrama y forma de implementación. Las estructuras de datos usadas fueron: -**Cola** por *Arreglo circular* - **Árbol** por: *Arreglo con señalador al padre*, *Lista de hijos*, *Hijo más izquierdo-hermanoDerecho por ptr*, *Hijo más izquierdo-hermanoDerecho talque el último hijo de un nodo apunte al padre*, *Hijo más izquierdo-hermanoDerecho con puntero al padre y al hermano más izquierdo.*

4.2.1. Arreglo Circular

4.2.1.1 Diagrama y Descripción

Esta estructuras de datos principalmente se usa para implementar el modelo de Cola, se trata de aun Array con dos punteros, uno llamado *rear* y otro *front*, cuando se llena,*rear* vuelve al inicio del array sobrescribiendo lo que había. Por la naturaleza del modelo Cola, esta estructura de datos ayuda a un encolado y desencolado más rápido.

En el siguiente diagrama se verá mejor su funcionamiento:

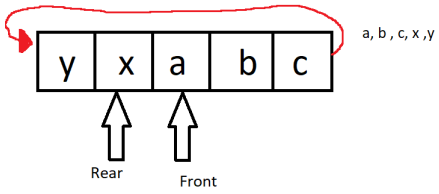

Figura 1: Array Circular

4.2.1.2 Definición den C++

```
#define MAX 10
class arrCircular {
public:
    struct{
        int a[MAX];
        int front;
        int rear;
    };
    arrCircular();
    virtual ~arrCircular();
    void iniciar();
    void destruir();
    void vaciar();
    bool vacia();
    void encolar(int e);
    int desencolar();
    int frente();

    void display();

private:
};
```

4.2.2. Arreglo con Señalador al Padre

4.2.2.1 Definición y Diagrama

Se trata de un arrglo de pares donde se guarda la etiqueta del nodo y un apuntador (que en este caso es un índice) a su padre. Es una estructura de datos iterativa aunque el modelo es por definicón recursivo. Diagrama:

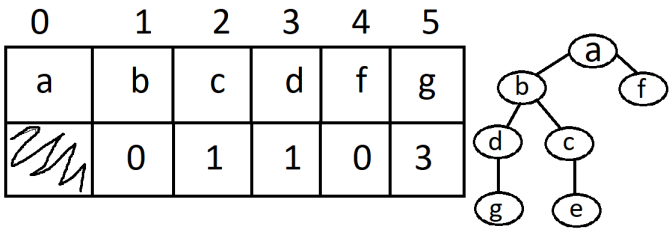

Figura 2: Arreglo con señalador al padre

4.2.2.2 Definición en C++

```
typedef int Nodo;
static int ndoNull = 0;
struct casilla {
    int etiqueta;
    int padre;
};
class arrSPadre {
public:
    arrSPadre();
    ~arrSPadre();
    void iniciar();
    void destruir();
    void vaciar();
    bool vacia();
    void ponerRaiz(int etq);
     casilla agregarHijo(int e, Nodo n);
    void borrarHoja(Nodo n);
    void modEtiqueta(Nodo n, int e);
    Nodo raiz();
    Nodo padre(Nodo n);
    Nodo hijoMasIzquierdo(Nodo n);
```

```
22      Nodo hermanoDerecho(Nodo n);
23      int  etiqueta(Nodo n);
24      int  numNodos();
25      int  numHijos(Nodo n);
26  private:
27      int  cantidadNodos;
28      int  ultLleno;
29       casilla  a[20];
30  };
```

4.2.3. Lista de Hijos

Se trata de una lista principal con todos los nodos de un árbol, cada nodo tiene una sublista con apuntadores a todos los hijos respectivos, se puede implementar con LSE-LSE, Array-Array o LSE-ARRAY.

4.2.3.1 Definición y Diagrama

Se trata de una lista principal con todos los nodos de un árbol, cada nodo tiene una sublista con apuntadores a todos los hijos respectivos, se puede implementar con LSE-LSE, Array-Array o LSE-ARRAY.

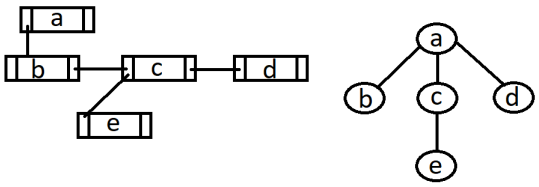

Figura 3: Lista de Hijos

4.2.3.2 Definición en C++

```
using namespace std;
struct Nodo{
    int etiqueta;
    Nodo *siguiente;
    Nodo *inferior;
    Nodo(){
        siguiente = 0;
        inferior = 0;
    }
    Nodo(int e){
        etiqueta = e;
        siguiente = 0;
        inferior = 0;
    }
};
typedef Nodo* nodo;
class listaHijos {
public:
    listaHijos();
    listaHijos(const listaHijos& orig);
    virtual ~listaHijos();
    void destruir();
    void vaciar();
    bool vacia();
    nodo agregarHijo(int e, int i, nodo n);
    void borrarHoja(nodo n);
    void modEtiqueta(nodo n, int e);
    void ponerRaiz(int etiqueta);
    nodo raiz();
    nodo padre(nodo n);
    nodo hijoMasIzquierdo(nodo n);
    nodo hermanoDerecho(nodo n);
    int etiqueta(nodo n);
    int numNodos();
private:
    nodo primero;
    int cantidadNodos;
};
```

4.2.4. Hijo más Izq-HermanoDerecho por punteros

4.2.4.1 Definición y Diagrama

Es una estructura de datos análoga al modelo, donde hay un puntero al hermano derecho y al hermano del izquierdo.

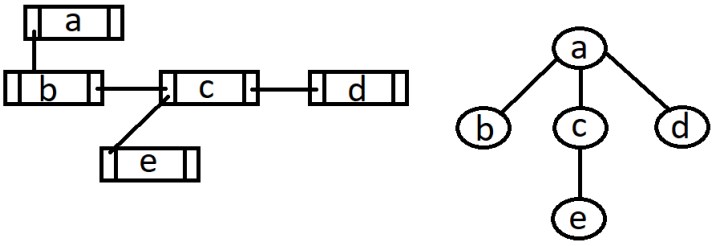

Figura 4: His más izq-hd por punteros

4.2.4.2 Definición en C++

```
struct Nodo {
    int etiqueta;
    Nodo *hijoIzq;
    Nodo *hermanoDer;
    int hijos;

    Nodo() {
        hermanoDer = 0;
        hijoIzq = 0;
        hijos = 0;

    };

    Nodo(int etiquetaR) {
        etiqueta = etiquetaR;
        hijoIzq = 0;
        hermanoDer = 0;
        hijos = 0;
    }

    ~Nodo() {
    };

};
class hijoIzqHD1 {
public:
    hijoIzqHD1();
    virtual ~hijoIzqHD1();
    void iniciar();
    void destruir();
    void vaciar();
    bool vacia();
    Nodo* agregarHijo(int e, int i, Nodo* n);
    void borrarHoja(Nodo* n);
```

```
35      void modEtiqueta(Nodo* n, int e);
36      Nodo* raiz();
37      void ponerRaiz(int e);
38      Nodo* padre(Nodo* n);
39      Nodo* hijoMasIzquierdo(Nodo* n);
40      Nodo* hermanoDerecho(Nodo* n);
41      int etiqueta(Nodo* n);
42      int numNodos();
43      int numHijos(Nodo* n);
44  private:
45      Nodo *primero;
46      int cantidadNodos;
47      Nodo* preOrden(Nodo* inicio, Nodo* n);
48  };
```

4.2.5. Hijo más Izq-HermanoDerecho con señalador al padre(último hijo)

4.2.5.1 Definición y Diagrama

Al igual que HMIHD, solo que el último hijo tiene un señalador directo al padre.

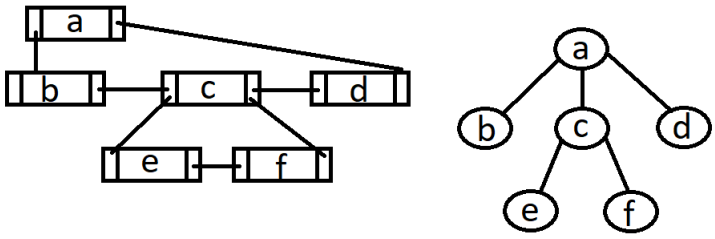

Figura 5: Con señalador al padre

4.2.5.2 Definición en C++

```
struct Nodo {

    Nodo() {
    };
    Nodo(int etiquetaR) : etiqueta(etiquetaR) {
    }
    ~Nodo() {
    };
    int etiqueta;
    bool ultimo;
    Nodo *padre;
    Nodo *hijoMI;
    Nodo *hD;
};
class hijoIzqHD2 {
public:
    hijoIzqHD2();
    virtual ~hijoIzqHD2();
    void iniciar();
    void destruir();
    void destruirRe(Nodo* n);
    void vaciar();
    bool vacia();
    void ponerRaiz(int etiqueta);
    void agregarHijo(int etiqueta, int posicion, Nodo* n);
    void borrarHoja(Nodo* n);
    void modEtiqueta(Nodo* n, int e);
    Nodo* raiz();
    Nodo* padre(Nodo* n);
    Nodo* hijoMasIzquierdo(Nodo* n);
    Nodo* hermanoDerecho(Nodo* n);
    int etiqueta(Nodo* n);
    int numNodos();
    int numHijos(Nodo* n);

private:
    static Nodo *nodoNulo;
    int cantidadNodos;
    Nodo *raizA;
};
```

4.2.6. Hijo más Izq-HermanoDerecho con señalador al padre y al hermano más izquierdo

4.2.6.1 Definición y Diagrama

Cada nodo tiene un puntero a su padre y al hermano más izquierdo.

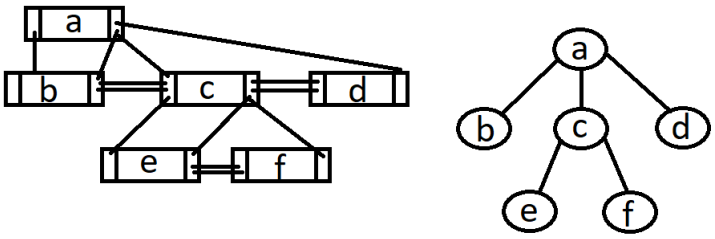

Figura 6: Con señaladores al padre y al HMI

4.2.6.2 Definición en C++

```
struct Nodo {
    Nodo() {
    };
    Nodo(int etiquetaR) : etiqueta(etiquetaR) {
    }
    ~Nodo() {
    };
    int etiqueta;
    Nodo *padre;
    Nodo *hI;
    Nodo *hijoMI;
    Nodo *hD;
};
class hijoIzqHD3 {
public:
    hijoIzqHD3();
    virtual ~hijoIzqHD3();
    void iniciar();
    void destruir();
    void destruirRe(Nodo* n);
    void vaciar();
    bool vacia();
    void ponerRaiz(int etiqueta);
    void agregarHijo(int e, int posicion, Nodo* n);
    void borrarHoja(Nodo* n);
    void modEtiqueta(Nodo* n, int e);
    Nodo* raiz();
```

```
28      Nodo* padre(Nodo* n);
29      Nodo* hijoMasIzquierdo(Nodo* n);
30      Nodo* hermanoDerecho(Nodo* n);
31      int etiqueta(Nodo* n);
32      int numNodos();
33      int numHijos(Nodo* n);
34  private:
35      static Nodo *nodoNulo;
36      int cantidadNodos;
37      Nodo *raizA;
38  };
```

5. Manual de Ususario

5.1. Requerimientos de Hardware

El programa no requiere componentes de hardware especiales o de un tipo específico. Por lo tanto puede ser ejecutado en cualquier arquitectura y configuración. El proyecto se elaboró en computadoras con procesadores Intel de distintas generaciones pero todos de 64bits y se ejecuta sin problemas.

5.2. Requerimientos de Software

Para el funcionamiento del programa se requiere la IDE NetBeans 8.2. Puede ser descargada en el siguiente enlace: `https://netbeans.org/downloads/`

5.3. Arquitectura del Programa

El programa principal cuenta con 2 carpetas, una para el modelo **Cola** donde hay 2 archivos llamados *arrCircular.cpp* y *arrCircular.h*. En la carpeta **Árbol** estan *arrSPadre.cpp, arrSPadre.h, hijoIzqHD1.cpp, hijoIzqHD1.h, hijoIzqHD2.cpp, hijoIzqHD2.h, hijoIzqHD3.cpp, hijoIzqHD3.h, listaHijos.cpp, listaHijos.h*, para esta parte, no fue necesario hacer una capa de interfaz ya que se utilizó la herramienta de *test* que tiene NetBeans.

5.4. Compilación

Para que el proyecto pueda ser compilado con éxito hace falta utilizar dos herramientas, GCC y C++11. GCC es un compilador creado originalmente para ambientes GNU y posteriormente migrado a Windows. Ese compilador permite el uso de C++11, la penúltima entrega del lenguaje, gracias a

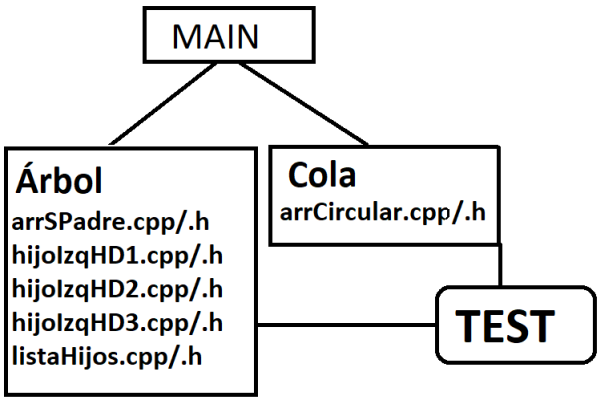

Figura 7: Arquitectura

el, se puede usar declaraciones como *nullptr* y un mejor uso de la memoria dinámica. Para el archivo *make* se utilizó la herramienta *Msys*. Aquí se adjunta el link del sitio principal para descargar el compilador TDM y el Msys en caso de que no lo tenga: `http://tdm-gcc.tdragon.net/index.php/` y `https://goo.gl/pNyTAs` (link acortado).

Se adjunta como debería verse su compilador:

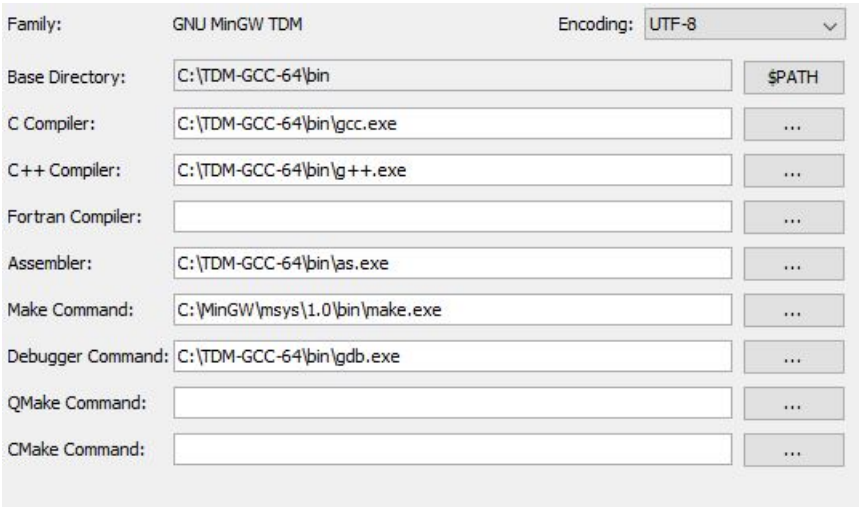

Figura 8: Compilador

5.5. Especificación del Programa

A contuación se le darán una serie de pasos a seguir para lograr que realizar las purebas del proyecto:

1. Descomprimir el archivo $TP1 - CI1221 - II\_B31494\_B47040\_B50060$.
2. Cargar el proyecto en NetBeans.
3. Dar Click izquierdo en el proyecto:

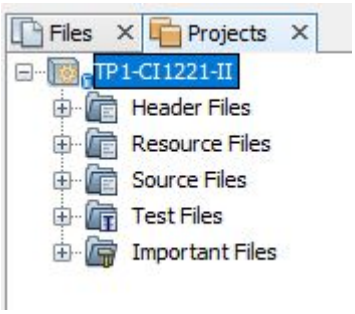

Figura 9: Paso

4. Seleccionar *test*:

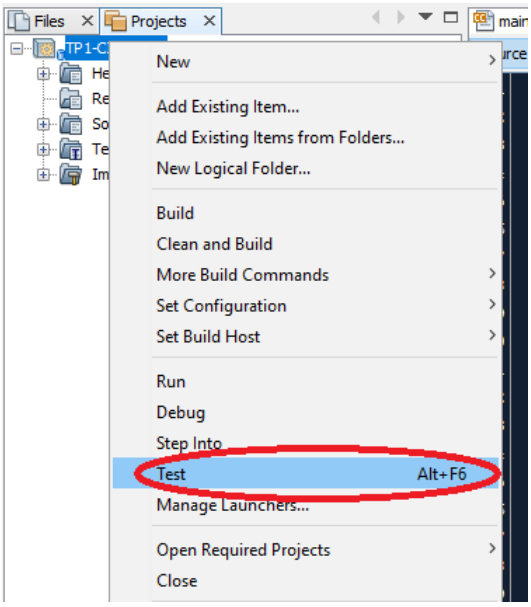

Figura 10: Paso

5. Ver la el output del test:

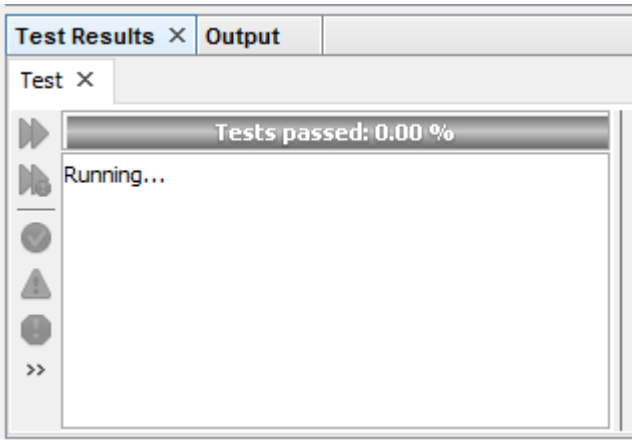

Figura 11: Paso

6. Datos de Prueba

6.1. Formato de los datos de prueba

El uso de los test de genera su propio formato de pruebas, en la carpeta test, están los .cpp y en estos, las pruebas que se realizaron.

6.2. Salida esperada

Gracias los test, se espera que la pestaña de un 100 % en todos los casos.

6.3. Salida obtenida

1. En las ED *hijoIzqHD1, hijoIzqHD2, listaHijos e hijoIzqHD3*, falla el operador Destruir y Vaciar. Elimina el hijo más izquierdo primero y al querer llegar al hermano derecho no puede.

2. Las purebas para la ListaHijos sale un fallo (error:127) total, después de varias horas de intentar ver que sucedía no se logró llegar a una solución, sin embargo, si se realizan pruebas en el main si funciona.

7. Lista de Archivos

1. Árbol:

   - arrSPadre.cpp

- arrSPadre.h
- hijoIzqHD1.cpp
- hijoIzqHD1.h
- hijoIzqHD2.cpp
- hijoIzqHD2.h
- hijoIzqHD3.cpp
- hijoIzqHD3.h
- listaHijos.cpp
- listaHijos.h

2. Cola:
   - arrCircular.cpp
   - arrCircular.h
3. test:
   - testhijoIzqHD2.cpp
   - testhijoIzqHD3.cpp
   - testhijoIzqHD1.cpp
   - testarrCir.cpp
   - testarrSPadre.cpp
4. MAIN

8. Referencias o Bibliografía

Estructuras de Datos. (2011). Colas (informática). Recuperado de http://estructura-de-datos-garo.blogspot.com/2011/10/colas-informatica.html