

# **ANALISTA PROGRAMADOR UNIVERSITARIO**

## **PROGRAMACIÓN ESTRUCTURADA**

**Trabajo Final Integrador**

### **SISTEMA DE CONCURSO**

### **“MISS PROGRAMACIÓN 2024”**

#### **Alumnos:**

- **Alfaro, Josué Dario**      **DNI: 41042253 LU: 3490**
- **Farfán, Ernesto Ángel**      **DNI: 40349837 LU: 3428**



**Facultad de Ingeniería**  
**Universidad Nacional de Jujuy**

# **2024**

# ÍNDICE

<b>Introducción</b>	2
<b>Análisis del Problema</b>	2
Entidades	2
Condiciones y Requisitos Especiales	4
<b>Diseño de la Solución</b>	4
<b>Tipos de datos y estructuras de datos utilizadas</b>	4
Declaración de Constantes	4
Definición de vectores de cadenas y proyectos	5
Declaración de Registros y Vectores Principales	5
Ejemplo de estructuras principales con datos cargados	8
<b>Diseño TOP-DOWN del sistema</b>	9
Menú Principal	9
Gestión de Participantes	10
Gestión de Jurados	11
Primera Etapa	12
Segunda Etapa	13
Coronación	14
<b>Consideraciones y aportes que ayudarán a entender la lógica</b>	15
<b>Codificación</b>	17
<b>Programa Principal</b>	17
<b>Administración de participantes</b>	19
<b>Gestión de Participantes</b>	21
<b>Gestión de Jurados</b>	25
<b>Gestión de la 1era Etapa</b>	26
<b>Gestión de la 2da Etapa</b>	29
<b>Coronacion</b>	32
<b>Pruebas</b>	34
Pruebas unitarias	34
Pruebas de integración	38
Pruebas de aceptación	46
<b>Conclusiones</b>	52

# Sistema de concurso “Miss Programación 2024”

## Introducción

Este informe tiene por objetivo explicar el funcionamiento detallado de los módulos más importantes del "Sistema de Concurso Miss Programación 2024", mostrando en cada fase el sistema desarrollado desde el planteo del problema, pasando por el análisis del sistema, el diseño propuesto, la codificación de los módulos principales y las pruebas necesarias para el correcto funcionamiento del sistema.

El sistema pretende dar solución a la problemática que surgió en el concurso de Miss Programación 2024. Dicho problema era la falta de una aplicación que proporcionara soporte para el almacenamiento y procesamiento de las candidatas y los jurados, a lo largo de las distintas etapas.

La solución diseñada pretende dar soporte al certamen, brindando un completo sistema de concurso basado en criterios evaluatorios que consisten en cumplir una serie de pasos (registrar a las participantes, registrar a los jurados, clasificar la primera y segunda etapa) antes de coronar a la campeona.

## Análisis del Problema

El problema que queremos resolver es la falta de una aplicación que ayude a gestionar el concurso de Miss Programación 2024. En el concurso participan muchas candidatas y también hay varios jurados que evalúan a las participantes en diferentes etapas. Sin una herramienta adecuada, es complicado llevar un control ordenado y preciso de toda la información.

### Entidades

#### 1. Candidatas

- **ID único:** Representa una única participante dentro del sistema
- **Primer Nombre:** Representa al primer nombre de la participante
- **Segundo Nombre:** Representa al segundo nombre
- **Apellido:** Representa al apellido
- **Edad:** Representa la edad de la participante (debe ser entre 21 y 40 años)
- **Altura:** Representa la altura de una participante (debe ser 1,50 a 1,90 mts.)
- **Cantidad de Idiomas:** Cant. De idiomas que conoce (Deben ser entre 1 a 6 idiomas)
- **Continente:** Continente de donde proviene (ASIA, EUROPA, ETC)
- **Cantidad de Lenguajes de Programación:** Cantidad que conoce (deben ser 1 a 10 lenguajes)
- **Años de Experiencia:** Debe tener entre 0 a 20 años de experiencia
- **Proyectos Finalizados:** Cantidad de Proyectos finalizados (0 en caso de años de exp 0)
- **Proyectos en Ejecución:** Cantidad de proyectos en ejecución
- **Proyectos Liderados (0 a Finalizados + Ejecución):** Cantidad de Proyectos en ejec. (0 en caso de años de exp 0)
- **Primera Etapa:** Representa una entidad de tipo primer etapa donde se clasificarán todas las participantes eligiendo las 20 mejores.
- **Segunda Etapa:** Se puntúan las 20 mejores según otros criterios de evaluación
- **Coronación:** Se elige a la Finalista y las 4 finalistas del certamen.

## 2. Jurados

- **ID Único:** Representa a un único Jurado dentro del sistema
- **Apellido:** El apellido del jurado
- **Nombre:** Representa el nombre del jurado
- **Empresa:** La empresa de cual proviene.
- **Cargo:** El cargo que ejerce el jurado
- **Años en la Industria:** La cantidad de años en la industria que tiene.

El problema en la parte de **jurados** es la necesidad de registrar, consultar, modificar, eliminar y listar jurados de manera eficiente y precisa.

Para solucionar este problema, se ha diseñado el módulo “**administracion\_jurados**”, el cual proporciona una interfaz interactiva que permite a los usuarios realizar todas las operaciones necesarias sobre los registros de los jurados. A través de un menú de opciones, los usuarios pueden acceder a las siguientes funcionalidades:

1. **Agregar Jurados:** Permite agregar nuevos jurados al sistema. Se verifica que haya al menos tres jurados registrados, ya que este es el mínimo necesario para un funcionamiento correcto del sistema.
2. **Consultar Jurados:** Permite consultar los datos de un jurado específico ingresando su ID. Utiliza una búsqueda binaria recursiva para encontrar rápidamente el jurado en cuestión.
3. **Modificar Jurados:** Permite modificar los datos de un jurado existente utilizando su ID.
4. **Eliminar Jurados:** Permite eliminar un jurado del sistema ingresando su apellido.
5. **Listar Jurados:** Genera y muestra una lista de todos los jurados registrados en el sistema.

El módulo “**administracion\_jurados**” asegura una gestión efectiva de los registros de jurados.

La gestión de **participantes**. La solución diseñada implica un módulo que permite registrar, consultar, modificar y eliminar participantes, así como controlar la distribución por continente.

La gestión de participantes presenta varios desafíos:

1. **Registro de Participantes:** La necesidad de registrar una gran cantidad de participantes de manera rápida y precisa, ya sea de forma automática o manual.
2. **Consultas:** La capacidad de consultar la información de cualquier participante de manera rápida y precisa.
3. **Modificaciones y Eliminaciones:** La posibilidad de modificar o eliminar los registros de los participantes.
4. **Distribución por continentes:** La distribución de los participantes por continente para asegurar una representación equilibrada.
5. **Menus y sub\_menus:** La importancia de contar con una interfaz fácil de usar para los usuarios del sistema.

Para solucionar estos problemas, se ha diseñado el módulo “**administracion\_participantes**”, el cual proporciona una interfaz interactiva que permite a los usuarios realizar todas las operaciones necesarias sobre los registros de los participantes. A través de un menú de opciones, los usuarios pueden acceder a las siguientes funcionalidades:

1. **Registrar Participantes:**
  - **Automático:** Permite generar automáticamente entre 30 y 1000 participantes, verificando que no se supere el máximo permitido de participantes en el sistema.
  - **Manual:** Permite agregar participantes manualmente.

2. **Consultar Participantes:**
  - Permite consultar los datos de un participante específico ingresando su ID. Esto facilita la rápida información precisa sobre cualquier participante.
3. **Modificar Participantes:**
  - Permite modificar los datos de un participante existente ingresando su ID.
4. **Eliminar Participantes:**
  - Permite eliminar un participante del sistema ingresando su ID.
5. **Listar Participantes:**
  - Genera y muestra una lista de todos los participantes registrados en el sistema. Esto facilita la revisión general de todos los registros.
6. **Distribución por continentes:**
  - Asegura que se mantenga una representación equilibrada de los participantes de diferentes continentes

El módulo “**administracion\_participantes**” garantiza que la distribución por continentes de los participantes se mantenga equilibrada, mejorando así la organización.

### Condiciones y Requisitos Especiales

- **Gestión de Participantes:**

No debe estar registrada una participante más de una vez.  
Debe haber un mínimo de 30 y un máximo de 1000 participantes.  
Debe existir al menos 4 participantes de cada continente para realizar las etapas
- **Jurados:**

No puede estar registrado el mismo jurado más de una vez.  
Deben haber al menos 3 a 6 jurados registrados para realizar las etapas.
- **Primera ETAPA:**

Las candidatas y jurados deben cumplir las condiciones (3 jurados, y 30 candidatas, al menos 4 de cada continente) antes de iniciar la clasificación.
- **Segunda ETAPA:**

No se puede realizar la segunda etapa del certamen si no se ejecutó la primera etapa.
- **Coronación:**

No se puede realizar la coronación si no se ejecutaron la primera y segunda etapas anteriores.

## Diseño de la Solución

### Tipos de datos y estructuras de datos utilizadas

#### Declaración de Constantes

```
// Declaracion de constantes
const int MAX_Nombre = 10;
const int MAX_Continentes = 5;
const int MAX_Proyectos = 3;
const int MAX_Participante = 1000;
const int MAX_Jurados = 6;
const int MAX_Finalistas = 5;
```

**MAX\_Nombre:** Define el tamaño de los vectores de primer, segundo nombre y apellidos creados para generar participantes automáticos.

**MAX\_Continentes:** Define el tamaño máximo del vector de continentes.

**MAX\_Proyectos:** Define el tamaño máximo del vector de proyectos de un participante.

**MAX\_Participante:** Define el tamaño máximo del vector de participantes.

**MAX\_Jurados:** Define el tamaño máximo del vector de jurados.

**MAX\_Finalistas:** Define el tamaño máximo del vector de participantes finalistas.

### Definición de vectores de cadenas y proyectos

```
typedef char tcad[30];
typedef tcad v_nombres[MAX_Nombre];
typedef tcad v_continentes[MAX_Continentes];
typedef int v_proyectos[MAX_Proyectos];
```

**tcad:** Define un vector de caracteres de tamaño 30 para almacenar caracteres en forma de cadena.

**v\_nombres:** Creamos el vector de nombres de tamaño MAX\_Nombre para almacenar nombres o apellidos (Ej: "María", "Diego", "Fernando", "Gutiérrez", ...)

**v\_continentes:** Creamos el vector de continentes de tamaño MAX\_Continentes para almacenar los distintos continentes (ASIA, AMERICA, EUROPA, OCEANIA, AFRICA).

**v\_proyectos:** Creamos un vector de tamaño MAX\_Proyectos para almacenar la cantidad de tipos de proyectos de un participante (0: Finalizados, 1: Ejecución, 2: Liderados)

### Declaración de Registros y Vectores Principales

#### Jurado:

```
typedef struct t_jurado
{
    int ID_jurado;           El registro representa una entidad de tipo jurado dentro del sistema,
    tcad apellido;          el cual tiene su id única, apellido, nombre, empresa, cargo y cantidad
    tcad nombre;            de años en la industria.
    tcad empresa;
    tcad cargo;
    int anios_indst;
};
```

**Nota del Jurado en la primera etapa:**

```
typedef struct nota_jurado_primer_etapa
{
    int ID_jurado;
    tcad apellido_jurado;
    tcad nombre_jurado;

    int nota_vestuario;
    int nota_elegancia;
    int nota_elocuencia;
    float nota_idiomas;
};
```

*Esta entidad se utiliza para determinar que jurado le pone dicha nota a la participante en la primera etapa, guardando para dicha nota el id del jurado, el apellido, nombre y las notas asignadas por el mismo.*

**Vector de notas del jurado en la primera etapa**

```
typedef nota_jurado_primer_etapa v_notas_jurado[MAX_Jurados];
```

*Declaramos el vector denominado v\_notas\_jurado de tamaño MAX\_Jurado para contemplar en un vector las distintas notas creadas aleatoriamente por el jurado en la etapa 1.*

**Primera etapa:**

```
typedef struct primera_etapa
{
    v_notas_jurado notas_jurado;
    float suma_nota_vestuario;
    float suma_nota_elegancia;
    float suma_nota_elocuencia;
    float suma_nota_idiomas;
    float puntaje_total;
};
```

*Declaramos la entidad que representará la primera etapa guardando dentro el vector de notas de primera etapa de cada jurado, y la suma del promedio de cada categoría*

*Al finalizar se suman todas las notas del promedio en una variable **puntaje\_total***

**Segunda etapa:**

```
typedef struct segunda_Etapa
{
    int lenguajes_pro;
    int experiencia;
    int proyectos_finalizados;
    int proyectos_en_ejecucion;
    float proyectos_liderados;
    float suma_total;
};
```

*Declaramos una entidad para las notas de la segunda etapa, basadas en reglas según cant de lenguaje, experiencia, etc que posee la participante. Las notas deben seguir un criterio, por lo que el promedio siempre es el mismo. Usamos variables enteras para cada categoría, y la nota de proyectos liderados sigue una regla según su cantidad. Al final, se suma el total en la variable **suma\_total**.*

**Participante:**

```
typedef struct t_participante
{
    int id;
    tcad apellido;
    tcad primer_nombre;
    tcad segundo_nombre;
    int edad;
    float altura;
    int cant_idiomas;
    tcad continente;
    int cant_lenguajes_prog;
    int anios_experiencia;
    v_proyectos proyectos;

    primera_etapa etapa1;
    segunda_etapa etapa2;
    float puntaje_Coronacion;
};
```

Definimos un registro para simbolizar la entidad de tipo Participante, dicho registro cuenta con los campos necesarios para identificar el participante, id, apellido, primer y segundo nombre, edad, y cantidad de idiomas, el continente y los lenguajes de prog. que maneja y los años de experiencia; La cantidad de proyectos finalizados, liderados y ejecución se contemplan en el vector **proyectos**. Y el puntaje obtenido se guarda en registros de **etapa1** y **etapa2**.

Al finalizar la primera y segunda etapa guardaremos el puntaje final del certamen en una variable llamada **puntaje\_Coronacion** donde se guarda el puntaje aplicado según la formula dada.

**Vectores Principales: Participantes, Jurados y Finalistas**

```
typedef t_jurado v_jurados[MAX_Jurados];
typedef t_participante v_participantes[MAX_Participante];
typedef t_participante v_finalistas[MAX_Finalistas];
```

Declaramos los vectores de cada tipo de registro declarado para el sistema.



Ejemplo de estructuras principales con datos cargados

#### PARTICIPANTE

Se muestra el ejemplo de un participante generado en el sistema con las estructuras de registros y vectores que contempla en su estructura, donde se pueden visualizar los distintos registros internos (etapa1 y etapa2) que se le agregaron y el puntaje total de la coronación

```
t_participante = {
  "id": 1000,
  "apellido": "Fernandez",
  "primer_nombre": "Leonela",
  "segundo_nombre": "Mariana",
  "edad": 25,
  "altura": 1.66,
  "cant_idiomas": 1,
  "continente": "ASIA",
  "cant_lenguajes_prog": 1,
  "anios_experiencia": 2,
  "proyectos": v_proyectos [1, 2, 3],
  "etapa1": primer_etapa {
    "notas_jurado": [
      {
        "ID_jurado": 8001,
        "apellido_jurado": "Miranda",
        "nombre_jurado": "Julian",
        "nota_vestuario": 10,
        "nota_elegancia": 10,
        "nota_elocuencia": 10,
        "nota_idiomas": 16.66
      },
      {
        "ID_jurado": 8002,
        "apellido_jurado": "Albornoz",
        "nombre_jurado": "David",
        "nota_vestuario": 10,
        "nota_elegancia": 8,
        "nota_elocuencia": 9,
        "nota_idiomas": 16.66
      },
      {
        "ID_jurado": 8003,
        "apellido_jurado": "Soria",
        "nombre_jurado": "Silvana",
        "nota_vestuario": 10,
        "nota_elegancia": 7,
        "nota_elocuencia": 9,
        "nota_idiomas": 16.66
      }
    ],
    "suma_nota_vestuario": 10,
    "suma_nota_elegancia": 8.33,
    "suma_nota_elocuencia": 9.33,
    "suma_nota_idiomas": 16.66,
    "puntaje_total": 44.32
  },
  "etapa2": segunda_etapa {
    "lenguajes_pro": 25,
    "experiencia": 25,
    "proyectos_finalizados": 25,
    "proyectos_en_ejecucion": 50,
    "proyectos_liderados": 75,
    "suma_total": 200
  },
  "puntaje_Coronacion": 244.32
}
```

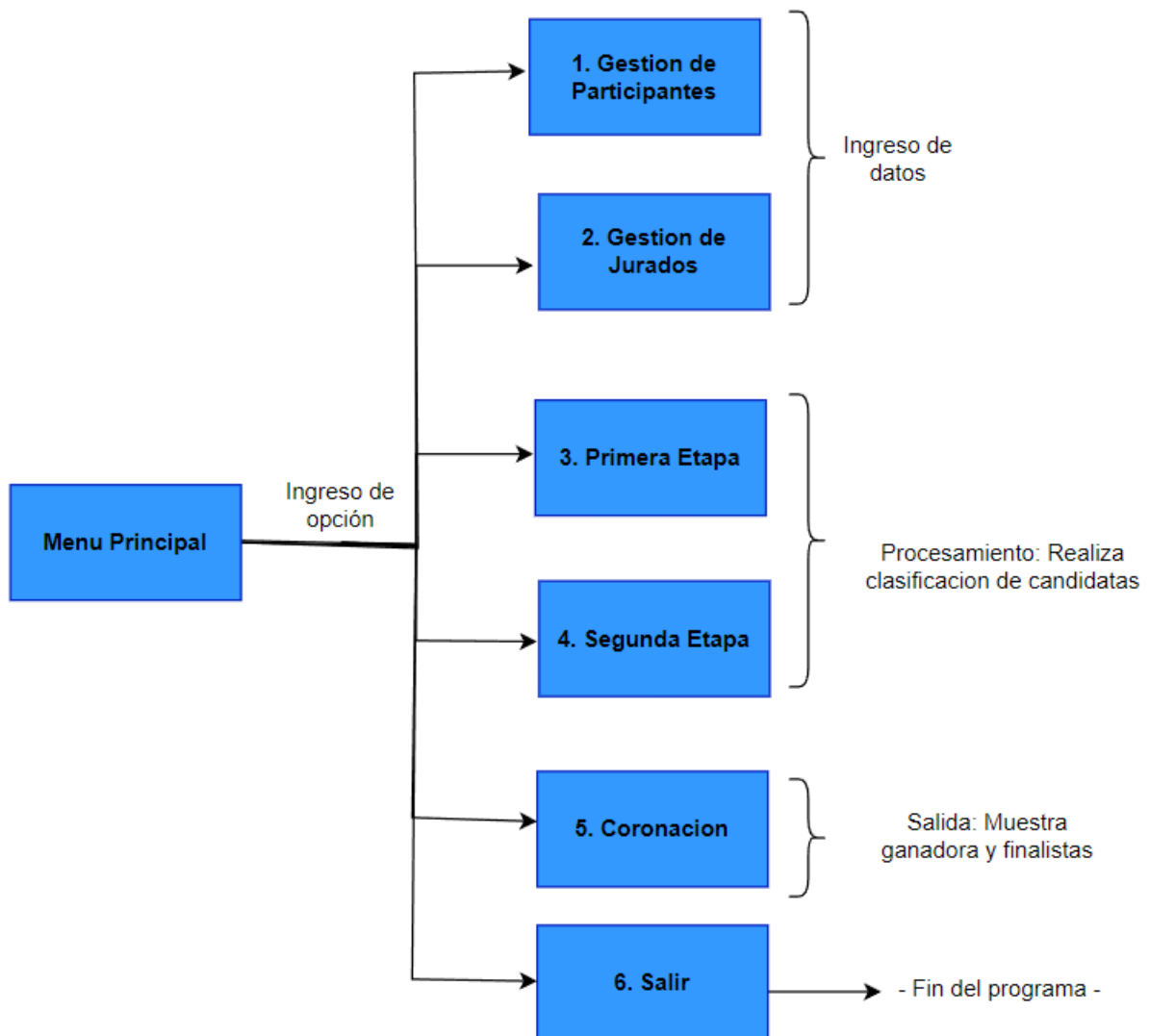
### JURADO

Se muestra el ejemplo de un jurado generado en el sistema, donde se visualizan los datos personales del jurado y el id único que lo identifica.

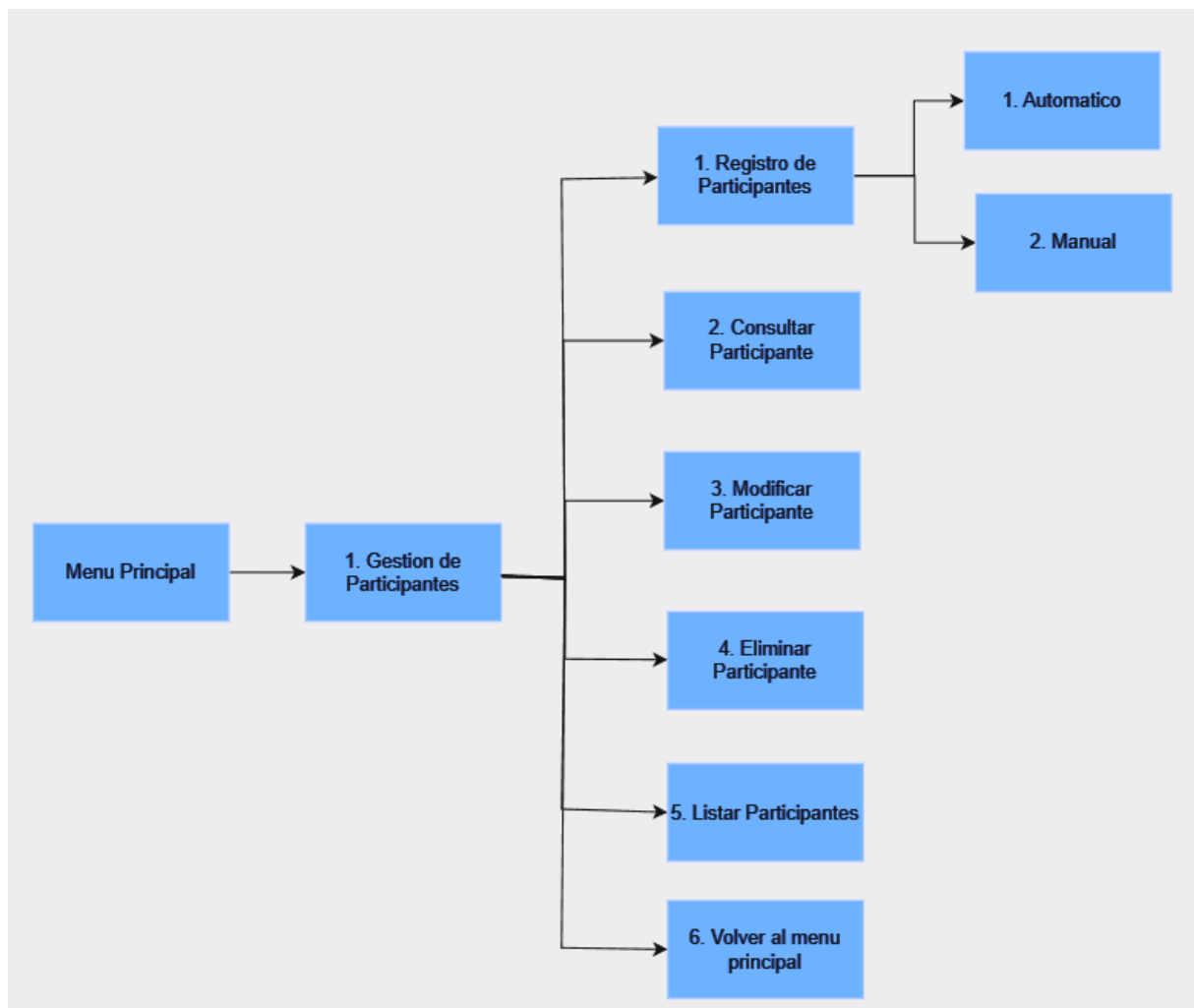
```
t_jurado = {  
    "ID_jurado": 8006,  
    "apellido": "Millar",  
    "nombre": "Morena",  
    "empresa": "Modas Monik",  
    "cargo": "Modelo",  
    "anios_indst": 10  
}
```

### Diseño TOP-DOWN del sistema

#### Menú Principal



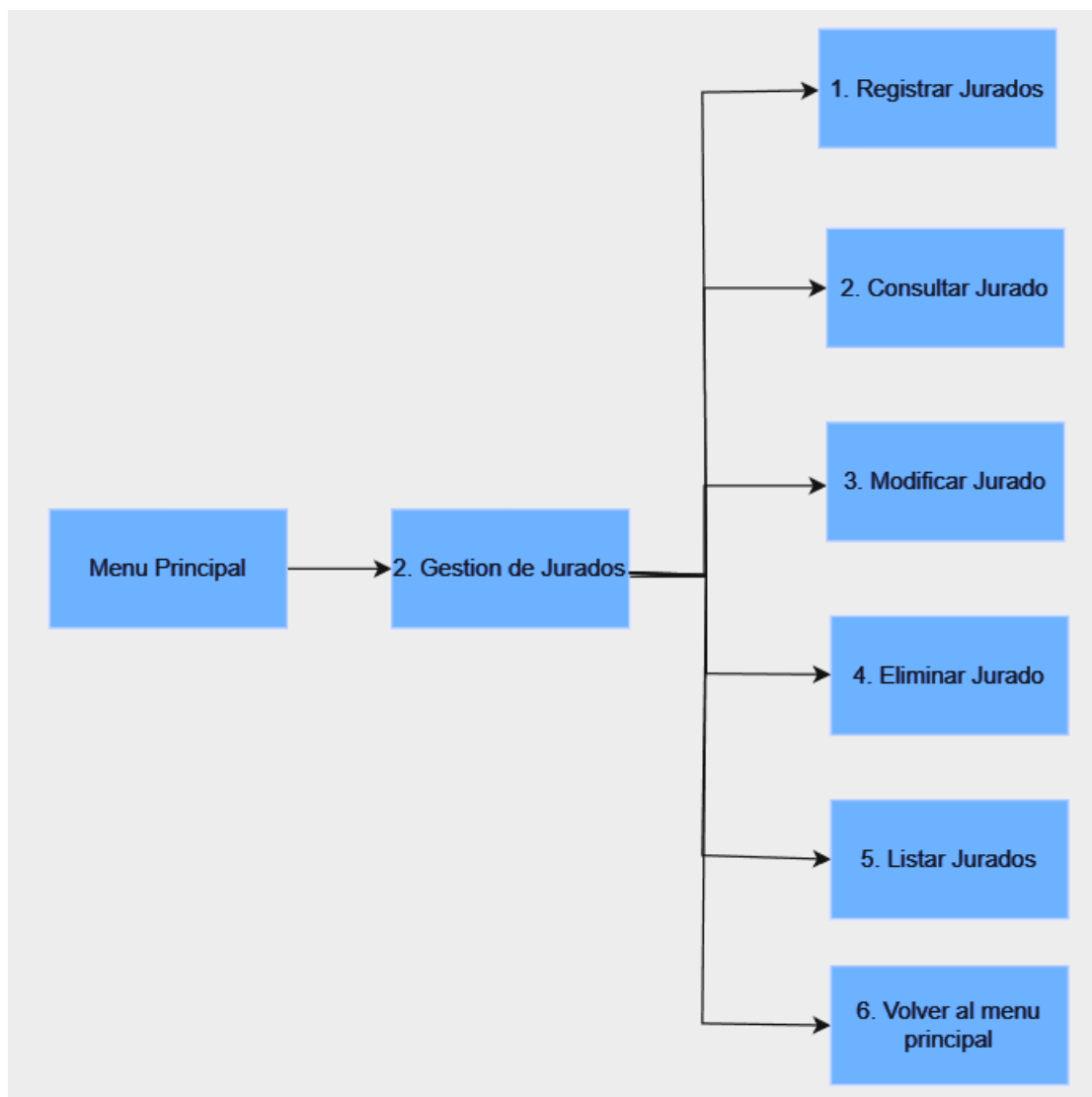
## Gestión de Participantes



- 1) Registro de participantes  
Propósito: Se utiliza para registrar los participantes en el sistema  
Entrada de datos: Vector de participantes y el ocupado  
Salida: Vector cargado con el participante
- 2) Consultar participante  
Propósito: Se utiliza para buscar un participante por id  
Entrada de datos: id del participante, Vector de participantes y el ocupado  
Salida: Datos del participante si está registrado, o mensaje de no existe.
- 3) Modificar Participante  
Propósito: Se utiliza para modificar los participantes en el sistema  
Entrada de datos: Se ingresa el id del participante a modificar, y el Vector de participantes y el ocupado  
Salida: Vector de participantes con el participante actualizado, o mensaje de no existe

- 4) Eliminar Participante  
Propósito: Eliminar un participante del Vector de participantes en el sistema  
Entrada de datos: Se ingresa el id del participante a eliminar, el Vector de participantes y el ocupado  
Salida: Vector de participantes restando el participante, o mensaje de no existe
- 5) Listar Participante  
Propósito: Listar los participantes en el sistema  
Entrada de datos: -  
Salida: Lista de participantes
- 6) Volver al menú Principal  
Propósito: Se utiliza para volver al menú principal

### Gestión de Jurados



## 1) Registro de jurados

Propósito: Se utiliza para registrar los jurados en el sistema

Entrada de datos: Vector de jurados y el ocupado

Salida: Vector cargado con el jurado

## 2) Consultar jurado

Propósito: Se utiliza para buscar un jurado por id

Entrada de datos: id del jurado, Vector de jurados y el ocupado

Salida: Datos del jurado si está registrado, o mensaje de no existe.

## 3) Modificar Jurado

Propósito: Se utiliza para modificar los jurados en el sistema

Entrada de datos: Se ingresa el id del jurado a modificar, y el Vector de jurados y el ocupado

Salida: Vector de jurados con el jurado actualizado, o mensaje de no existe

## 4) Eliminar Jurado

Propósito: Eliminar un jurado del vector de jurados en el sistema

Entrada de datos: Se ingresa el Apellido del jurado a eliminar, el Vector de jurados y el ocupado

Salida: Vector de jurados restando el jurado, o mensaje de no existe

## 5) Listar Jurados

Propósito: Listar los Jurados en el sistema.

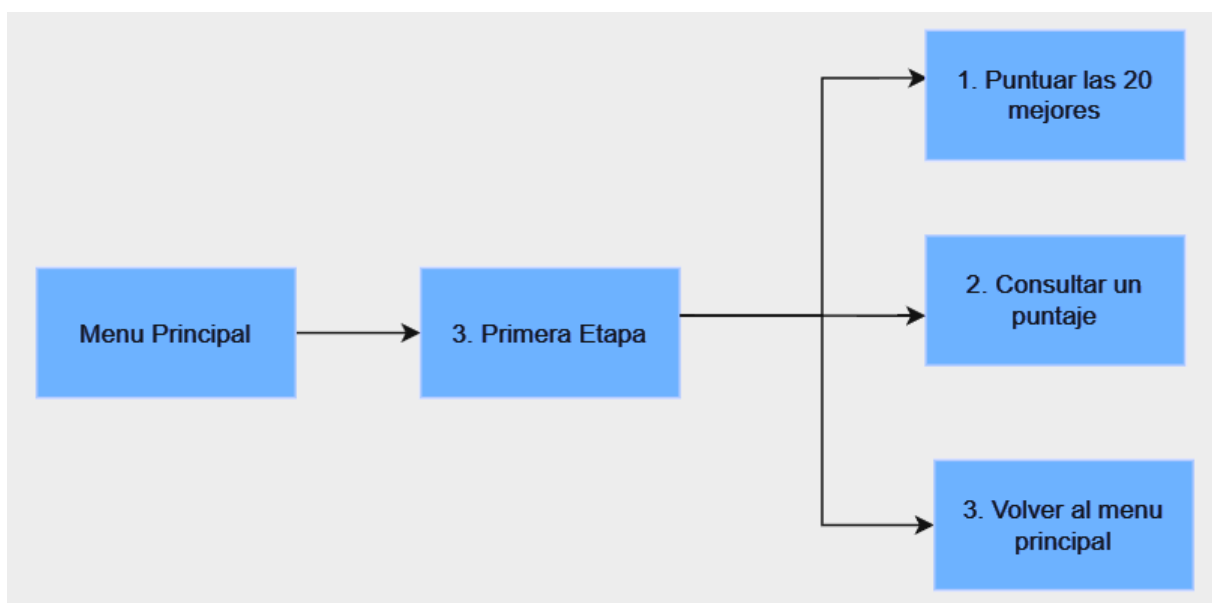
Entrada de datos: -

Salida: Lista de Jurados.

## 6) Volver al menú Principal

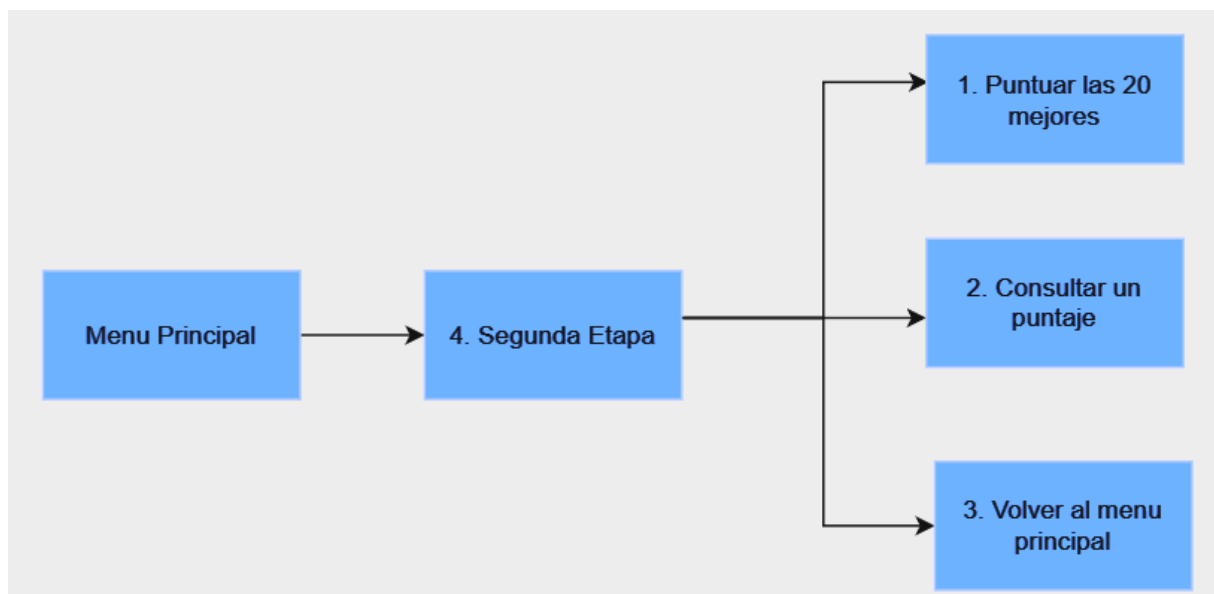
Propósito: Se utiliza para volver al menú principal.

## Primera Etapa



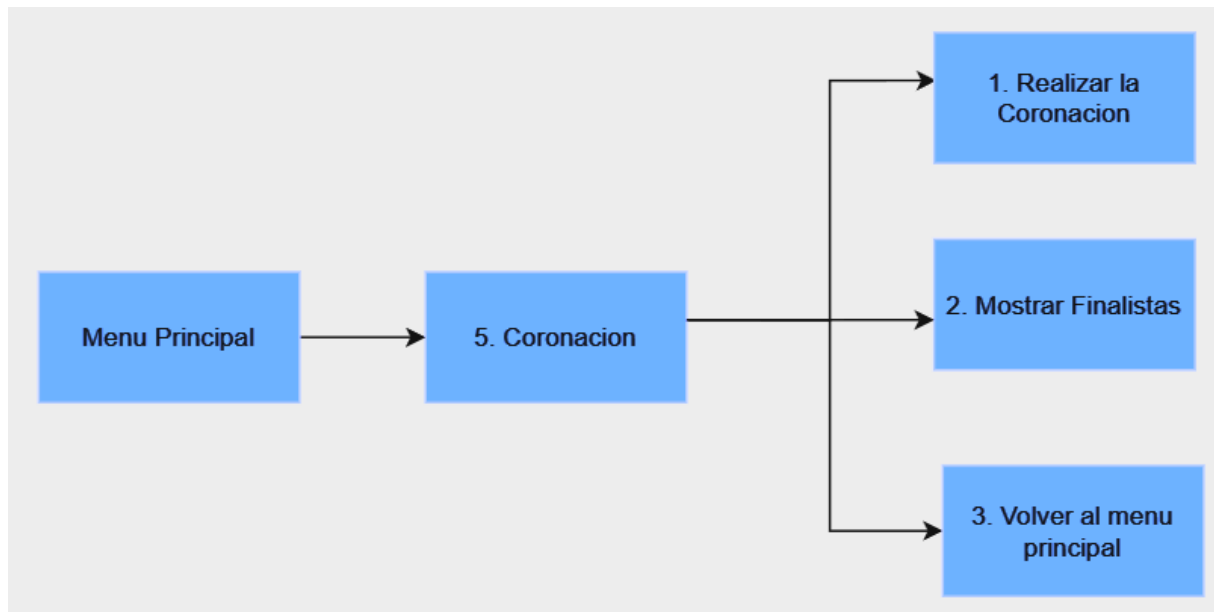
- 1) Puntuar las 20 mejores  
Propósito: Definir el puntaje de todas las participantes eligiendo las 20 con mejor puntaje y guardándolas en una lista de participantes que pasaron la etapa 1.  
Entrada de datos: Vector de participantes y el ocupado.  
Salida: Lista con 20 mejores participantes.
- 2) Consultar un puntaje  
Propósito: Permite consultar el puntaje de la etapa 1 de todas las participantes ingresando el id a consultar.  
Entrada de datos: Lista de participantes completa y el id a consultar.  
Salida: Resultado de la participante en la etapa 1, o mensaje de no existe.
- 3) Volver al menú principal  
Propósito: Se utiliza para volver al menú principal.

### Segunda Etapa



- 1) Puntuar las 20 mejores  
Propósito: Definir el puntaje de las 20 participantes seleccionadas siguiendo criterios evaluatorios de la etapa 2.  
Entrada de datos: Vector de participantes seleccionadas y el ocupado.  
Salida: Lista de participantes con el puntaje obtenido en la etapa 2.
- 2) Consultar un puntaje  
Propósito: Permite consultar el puntaje de la etapa 2 de las 20 participantes ingresando el id a consultar.  
Entrada de datos: Lista de participantes seleccionadas y el id a consultar.  
Salida: Resultado de la participante en la etapa 2, o mensaje de no existe.
- 3) Volver al menú principal

## Coronación



## 1) Realizar la coronación

Propósito: Suma los puntajes obtenidos de las 20 participantes en la primera y segunda etapa, y devuelve la lista de la ganadora y las 4 finalistas.

Entrada de datos: Vector de participantes que pasaron ambas etapas, su ocupado, Vector de finalistas y su ocupado.

Salida: Vector de finalistas cargado con las finalistas del certamen.

## 2) Mostrar Finalistas

Propósito: Mostrar la ganadora del certamen y las 4 finalistas con mejor puntaje obtenido en las etapas.

Entrada de datos: -

Salida: Lista de finalistas.

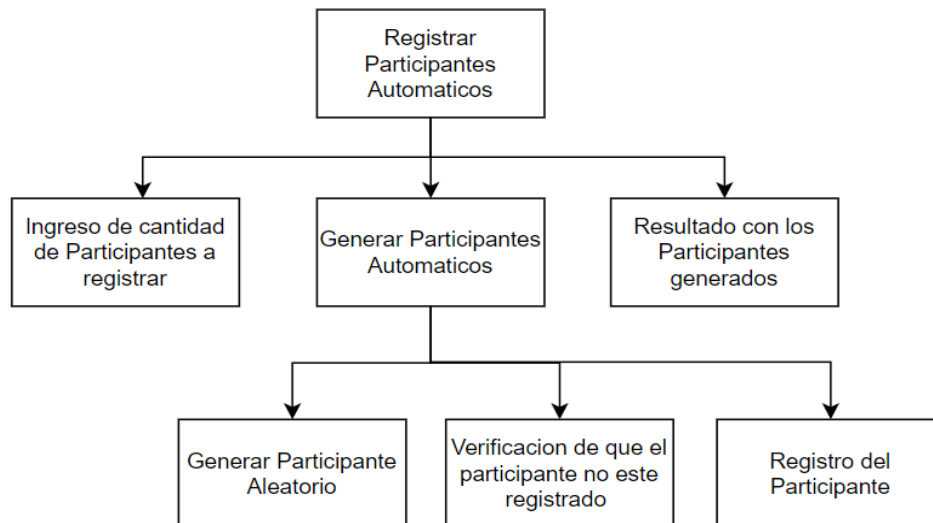
## 3) Volver al menú principal

Propósito: Volver al menú principal.

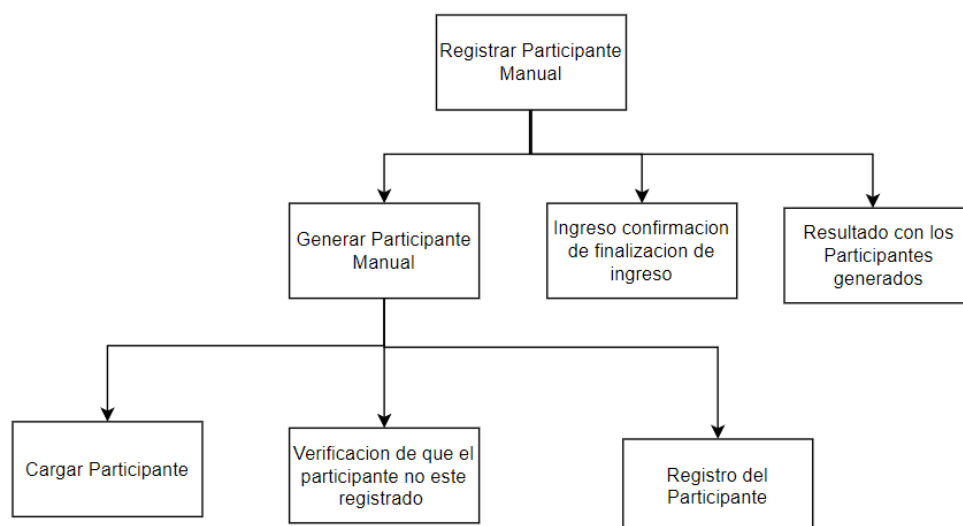
## Consideraciones y aportes que ayudarán a entender la lógica

**Creación Automática de Participantes**

A continuación, se planteará como se fue desglosando el módulo de creación automática de participantes

**Creación Manual de Participantes**

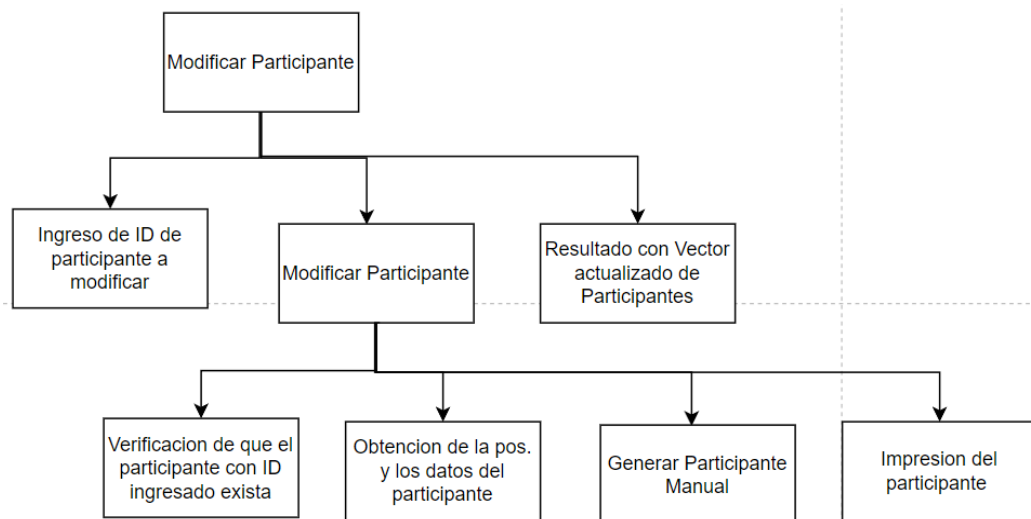
Se mostrará cómo se fue desglosando el módulo de creación manual de participantes



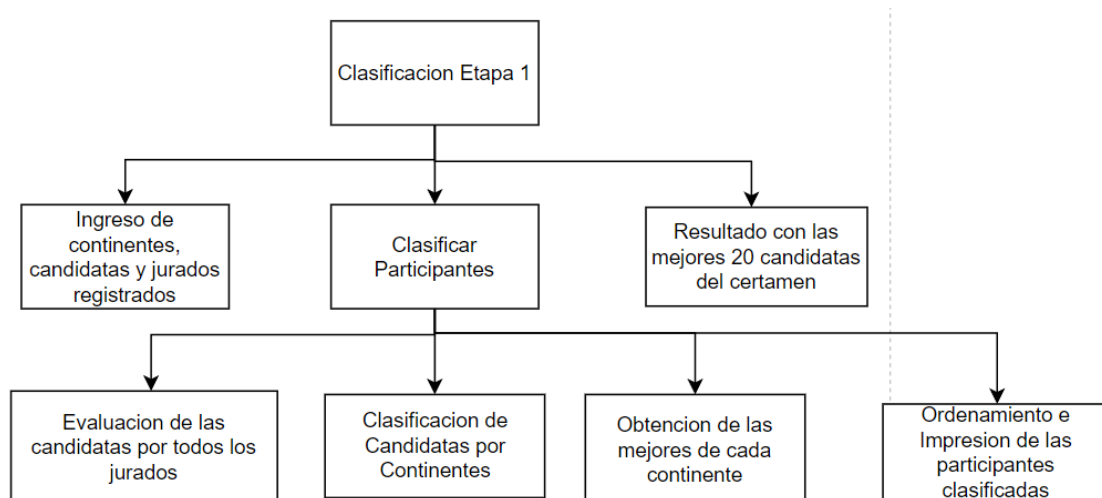


### Modificar Participante

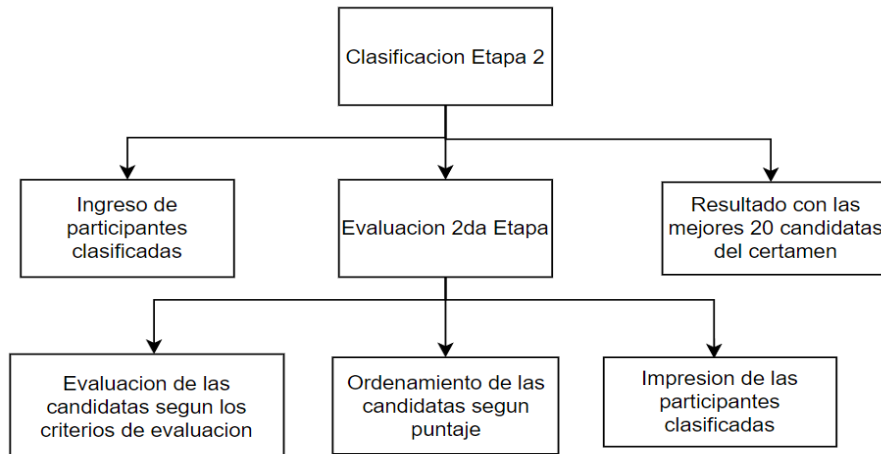
El modulo modificar participante cuenta con la lógica de llamar al módulo generar participante manual, desglosado en la Creación Manual de Participantes, para la carga del participante. Validando el campo primer, segundo nombre y apellido posterior se actualiza la posición y se imprime el participante.



### Primera Etapa



## Segunda Etapa



## Codificación

### Programa Principal:

El programa principal (**main**) define la estructura principal del sistema propuesto, tal como se observó en el diseño, contamos con distintos módulos de **administración** para cada fase del certamen donde se desarrollará toda la lógica que el sistema necesita para funcionar.

- Declaramos el `srand(time(NULL))` para generar una semilla de números aleatorios que usaremos a lo largo del sistema, declaramos un char llamado **op** para gestionar el menú, y las opciones ingresadas, y declaramos y llenamos los continentes, lo llenamos en el main por que se deberá usar por distintos **módulos** de esa forma ya queda cargado por única vez.
- Luego declaramos cada variable que servirá de contador, bandera, variables de control que utilizaremos para ir verificando en cada instancia que ocurra lo que el sistema espera.
- Declaramos los vectores principales para el funcionamiento, vector de **jurados** (`v_jurado`), **participantes**, **seleccionadas\_et1**, **finalistas** (`v_participante`), cada vector con su correspondiente ocupado.
- Luego de mostrar el menú, se procede a leer la opción elegida (**op**) y la sentencia **switch**, determinará el módulo de "**administración\_...**" que debe elegir o contrario a eso, indicará opción incorrecta
- Cada módulo de administración cuenta con su correspondiente menú y módulos internos para llevar la lógica del programa.

```

main()
{
    srand(time(NULL));
    char op;
    v_continentes continentes = {"AFRICA", "AMERICA", "ASIA", "EUROPA", "OCEANIA"};

    //PARTICIPANTES
    int vAfrica = 0, vAmerica = 0, vAsia = 0, vEuropa = 0, vOceania = 0;
    bool control_participante = false;
    v_participantes participantes;
    int ocup_participantes = -1;
    int id_participante_base = 1000;

    //JURADOS
    v_jurados jurados; //vector
    int ocupado_ju = -1;
    bool control_jurados = false;
    int id_jurado_base = 8000;

    //ETAPA 1
    v_participantes seleccionadas_et1;
    int ocup_seleccionadas_et1 = -1;
    bool control_etapa1 = false;

    //ETAPA 2
    bool control_etapa2 = false;

    //CORONACION
    v_finalistas finalistas;
    int ocup_finalistas = -1;

    do
    {
        system("cls");
        menu_principal(op);
        switch (op)
        {
            case '1':
                administracion_participantes(continentes, participantes, ocup_participantes,
                    id_participante_base, vAfrica, vAmerica, vAsia, vEuropa, vOceania, control_participante);
                break;
            case '2':
                administracion_jurados(jurados, ocupado_ju, control_jurados, id_jurado_base);
                break;
            case '3':
                administracion_etapa_1(continentes, participantes, ocup_participantes,
                    vAfrica, vAmerica, vAsia, vEuropa, vOceania, jurados, ocupado_ju, control_jurados,
                    control_participante, seleccionadas_et1, ocup_seleccionadas_et1, control_etapa1);
                break;
            case '4':
                administracion_etapa_2(seleccionadas_et1, ocup_seleccionadas_et1, control_etapa1, control_etapa2);
                break;
            case '5':
                administracion_coronacion(seleccionadas_et1, ocup_seleccionadas_et1, finalistas, ocup_finalistas, control_etapa2);
                break;
            case '6':
                cout << "\n- Fin del programa -" << endl;
                break;
            default:
                cout << "Opcion invalida." << endl;
        }
        system("pause");
    } while (op != '6');
}

```

### Menu Principal:

Se encuentra dentro del módulo principal (**main**) y detalla de manera concisa cada módulo de administración que posee el sistema, solicitando un char (**opcion**) al finalizar para dirigir a cada módulo de administración.

```

void menu_principal(char &opcion)
{
    cout << "*****\n";
    cout << "* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *\n";
    cout << "*****\n";
    cout << "*\n";
    cout << "* 1. Gestión de participantes *\n";
    cout << "* 2. Gestión de Jurados *\n";
    cout << "* 3. Primera Etapa *\n";
    cout << "* 4. Segunda Etapa *\n";
    cout << "* 5. Coronación *\n";
    cout << "* 6. Salir *\n";
    cout << "*****\n";
    cout << "Seleccione una opcion: ";
    cin >> opcion;
}

```

## Administración de participantes:

Uno de los módulos de administración más importantes que tiene el sistema es el módulo de administración de participantes donde se realiza toda la **gestión de participantes**, desde **registrar** manual, o automáticamente participantes, consultar por **id**, **eliminar**, etc. A continuación, describiremos el módulo de administración:

- Los parámetros que recibe son, vector de continentes (**continentes**), vector de participantes (**parts**), su ocupado (**ocup**), el **id** base que irá actualizándose por cada registro nuevo, contadores para contar cada participante de distinto continente y una variable de control (**control\_participante**) que controle el mínimo de **parts** necesarios para pasar a primera etapa.
- Inicializamos las distintas variables que usara el modulo, opcion (char para registro manual e int para el menú ppal), registro auxiliar de nuevo participante, variables de control y variables de tipo\_registro (para identificar manual y automatico)
- A continuación iniciamos un bucle **do{ }while**, que mostrara (**menú\_gestion\_participantes**) las distintas opciones del modulo y según el tipo de opcion ingresará a los distintos modulos que permiten gestionar participantes hasta que presione 6 (volver al menú ppal).
- Entre ellas tenemos el módulo de registro de participante, donde al ingresar, solicitará **1** para registro automatico (entre 30 y 1000 o dará error de fuera de rango) y registro manual (hasta presionar “n” ), el registro automatico contemplara que el valor ingresado pueda cargar los participantes caso contrario dira que el valor no puede ser cargado: (Ejemplo, cargamos 500 participantes, el vector mantendrá 500 lugares libres, al momento de cargar 600 más, la suma dará 1100, lo que es invalido ya que el limite es 1000 y el sistema indicara que solo puede cargar 500 más)
- Al finalizar la carga nos mostrará cuantos participantes se cargaron de distintos continentes.
- La opcion de consultar participante (2) permitirá ingresar un id a consultar (**valor**) y llamara al modulo “**consultar\_participante**” donde se indicará el tipo de consulta (0) dicho tipo de consulta mostrara los datos básicos del participante
- La opcion modificar permite modificar un participante existente en el sistema, indicando el **id** a modificar (**valor**) y llamando al modulo “**modificar\_participante**” (se explicará mas abajo), posteriormente se controla que los participantes cumplan criterios de la etapa 1, que hayan 4 por continentes y mayor a 30 (“**control\_continentes**”)
- Listar participantes, permite listar la cantidad de participantes totales que están cargados, de forma recursiva, esto crea una instancia del modulo (“**listar\_participantes\_rec**”) desde el final (**ocup**) hasta ejecutar el primero (0)

```

void administracion_participantes(v_continentes continentes, v_participantes &parts, int &ocup, int &id_participante, int &vAfrica, int &vAmerica,
int &vAsia, int &vEuropa, int &vOceania, bool &control_participante)
{
    int op, tipo_registro, valor;
    int ocup_continentes = MAX_Continentes - 1;
    t_participante nuevo;
    char opcion;
    bool bandCargados;

    do
    {
        bandCargados = false;
        system("cls");
        menu_gestion_participantes(op);
        switch (op)
        {
            case 1:
                do
                {
                    system("cls");
                    cout << "--Registrar Participante--" << endl;
                    cout << "- Seleccione tipo de registro -" << endl;
                    cout << "1) Automatico - 2) Manual: ";
                    cin >> tipo_registro;

                    if (tipo_registro != 1 && tipo_registro != 2)
                    {
                        cout << "Ingresó tipo registro inválido: " << tipo_registro << endl;
                        system("pause");
                    }
                }
                while (tipo_registro != 1 && tipo_registro != 2);

                switch (tipo_registro)
                {
                    case 1:
                        do
                        {
                            cout << "Ingrese cantidad de registros a generar (entre 30 y 1000): ";
                            cin >> valor;
                            if (!(valor >= 30 && valor <= 1000))
                                cout << "\nValor fuera del rango..." << endl;
                        }
                        while (!(valor >= 30 && valor <= 1000));

                        if ((MAX_Participante - ocup) > valor)
                        {
                            cout << "\nSe generaran: " << valor << " participantes automaticamente...\n" << endl;
                            generar_participantes_automaticos(id_participante, valor, parts, continentes, ocup_continentes, ocup, nuevo);
                            bandCargados = true;
                        }
                        else
                        {
                            cout << "\nSolo puede agregar: " << (MAX_Participante - 1) - ocup << " participantes.\n" << endl;
                        }
                        break;

                    case 2:
                        do
                        {
                            bandCargados = true;
                            generar_participantes_manual(id_participante, parts, continentes, ocup_continentes, ocup, nuevo);
                            cout << "Desea seguir ingresando participantes? s/n: " << endl;
                            cin >> opcion;
                        }
                        while (opcion != 'n' && opcion != 'N');
                        break;
                }

                control_continentes(parts, continentes, ocup, vAfrica, vAmerica, vAsia, vEuropa, vOceania, control_participante);
                if (bandCargados == true)
                {
                    cout << "\nSe cargaron: " << vAfrica << " participantes de AFRICA" << endl;
                    cout << "\nSe cargaron: " << vAmerica << " participantes de AMERICA" << endl;
                    cout << "\nSe cargaron: " << vAsia << " participantes de ASIA" << endl;
                    cout << "\nSe cargaron: " << vEuropa << " participantes de EUROPA" << endl;
                    cout << "\nSe cargaron: " << vOceania << " participantes de OCEANIA\n" << endl;
                }

                system("pause");
                break;

            case 2:
                system("cls");
                cout << "--Consultar Participante--" << endl;
                cout << "Ingrese ID de participante a consultar: ";
                cin >> valor;
                consultar_participante(parts, ocup, valor, 0);
                system("pause");
                break;

            case 3:
                system("cls");
                cout << "--Modificar Participante--" << endl;
                cout << "Ingrese ID de participante a modificar: ";
                cin >> valor;
                modificar_participante(parts, ocup, valor, continentes, ocup_continentes);
                control_continentes(parts, continentes, ocup, vAfrica, vAmerica, vAsia, vEuropa, vOceania, control_participante);
                system("pause");
                break;

            case 4:
                system("cls");
                cout << "--Eliminar Participante--" << endl;
                cout << "Ingrese ID de participante a eliminar: ";
                cin >> valor;
                eliminar_participante(parts, ocup, valor);
                control_continentes(parts, continentes, ocup, vAfrica, vAmerica, vAsia, vEuropa, vOceania, control_participante);
                system("pause");
                break;

            case 5:
                system("cls");
                cout << "--Lista de Participantes--" << endl;
                listar_participantes_rec(parts, ocup);
                system("pause");
                break;

            case 6:
                break;

            default:
                cout << "Opción inválida" << endl;
                system("pause");
                break;
        }
    }
    while (op != 6);
}

```

## Gestión de Participantes:

Como habíamos mencionado anteriormente, la opción **Registrar Participantes**, cuenta con dos métodos de carga (manual o automática).

### Carga automática

#### **Modulo “genera\_participantes\_automáticos”**

Va a llamar al módulo “**cargar\_participante\_auto**” de acuerdo a cuantas participantes (**total\_cargados**) quiere registrar el usuario.

- El bucle for se ejecuta desde 1 hasta **total\_cargados**, generando así el número total de participantes especificado.
- Se llama al módulo “**cargar\_participante\_auto**” para cargar automáticamente los datos del nuevo participante, asignándole un continente y otros detalles necesarios.
- Se hace una copia de los participantes actuales en “**aux\_participantes**” usando la función “**copiar\_participantes**” y se verifica si el nuevo participante ya existe en “**aux\_participantes**” usando la función “**buscar\_participante\_by\_nombre\_apellido**” como condición dentro de un bucle **while**. Si el participante ya existe, se genera un nuevo participante y se repite la verificación.
- Una vez que se verifica que el nuevo participante no está duplicado, se inserta en el vector **parts** usando el procedimiento “**insertar\_participante**” y por último se incrementa el contador “**id\_participante**” para el próximo participante.

```
void generar_participantes_automaticos(int &id_participante, int total_cargados, v_participantes &parts,
                                     v_continentes continentes, int ocup_continentes, int &ocup, t_participante &nuevo){
    int i;
    v_participantes aux_participantes;
    for (i = 1; i <= total_cargados; i++)
    {
        nuevo.id = id_participante;
        cargar_participante_auto(nuevo, continentes, ocup_continentes);

        copiar_participantes(parts, aux_participantes, ocup);

        while(buscar_participante_by_nombre_apellido(aux_participantes, ocup, nuevo))
        {
            cargar_participante_auto(nuevo, continentes, ocup_continentes);
        }
        insertar_participante(parts, ocup, nuevo);
        id_participante++;
    }
}
```

#### **Modulo “cargar\_participante\_auto”**

Como podemos observar cada campo se carga aleatoriamente.

La variable **p** es un registro de tipo **t\_participante**, la variable **continentes** es de tipo **v\_continentes** que ingresa al módulo con datos ya definidos.

Los proyectos finalizados (0) y liderados (2) dependen de la cantidad de **años de experiencia** de la participante, si la participante tiene cero años de experiencia no podrá no tener finalizados ni liderados.

```

void cargar_participante_auto(t_participante &p, v_continentes continentes, int ocup_continente)
{
    v_nombres nombres1 = {"Maria", "Soledad", "Merrye", "Cristal", "Alexis", "Abigail", "Any", "Lizbeth", "Yesica", "Abril"};
    v_nombres nombres2 = {"Aba", "Trinidad", "Carla", "Defina", "Antonella", "Leonarda", "Amy", "Mirtha", "Josefa", "Isabel"};
    v_nombres apellidos = {"Willson", "Vince", "Jhonson", "Santillan", "Velazques", "Birge", "Calvo", "Renteria", "Torres", "Reinaga"};

    strcpy (p.primer_nombre, nombres1[aleatorio(0, MAX_Nombre - 1)]);
    strcpy (p.segundo_nombre, nombres2[aleatorio(0, MAX_Nombre - 1)]);
    strcpy (p.apellido, apellidos[aleatorio(0, MAX_Nombre - 1)]);
    strcpy(p.continente, continentes[aleatorio(0, ocup_continente)]);

    p.edad = aleatorio(21, 40);
    p.altura = aleatorio(150, 190) / 100.0;

    p.cant_idiomas = aleatorio(1, 6);
    p.cant_lenguajes_prog = aleatorio(1, 10);
    p.anios_experiencia = aleatorio(0, 20);

    p.proyectos[1] = aleatorio(0, 4);

    if(p.anios_experiencia > 0)
    {
        p.proyectos[0] = aleatorio(0, 25);
        p.proyectos[2] = aleatorio(0, p.proyectos[0] + p.proyectos[1]);
    }
    else
    {
        p.proyectos[0] = 0;
        p.proyectos[2] = 0;
    }
}

```

### Carga Manual:

El modulo “**generar\_participantes\_manual**” está diseñado para registrar manualmente nuevos participantes en un vector de registros de **t\_participante**.

- Este módulo llama a “**cargar\_participante\_manual**” y carga la información del nuevo participante de manera manual (por teclado) validando cada campo cumpla con cada criterio especificado.
- El modulo “**copiar\_participantes**” copia la lista de participantes actuales a un vector auxiliar para realizar verificaciones sin alterar la lista de participantes original.
- Se utiliza un bucle **while** para verificar si el nuevo participante ya está registrado en la lista de participantes utilizando la función “**buscar\_participante\_by\_nombre\_apellido**”, si el participante ya está registrado, se muestra una advertencia y se solicita cargar nuevamente la información del participante.
- Por último el modulo “**insertar\_participante**”; inserta el nuevo participante en la lista de participantes y incrementa el contador “**id\_participante**” para el próximo registro

```

void generar_participantes_manual(int &id_participante, v_participantes &parts, v_continentes continentes,
    int ocup_continentes, int &ocup, t_participante &nuevo){

    v_participantes aux_participantes;
    nuevo.id = id_participante;

    cargar_participante_manual(nuevo, continentes, ocup_continentes);

    copiar_participantes(parts, aux_participantes, ocup);

    while(buscar_participante_by_nombre_apellido(aux_participantes, ocup, nuevo))
    {
        cout << "(Advertencia) El participante: " << nuevo.apellido << ", " << nuevo.primer_nombre << ", " << nuevo.segundo_nombre << " Ya esta registrado." << endl;
        system("pause");
        cargar_participante_manual(nuevo, continentes, ocup_continentes);
    }

    insertar_participante(parts, ocup, nuevo);
    id_participante++;
}

```

**Búsqueda de participante por nombres y apellidos**

El modulo “**buscar\_participante\_by\_nombre\_apellido**” fue diseñado para devolvernos un valor lógico true o false en caso encuentre al participante buscado por primer,segundo nombre y apellido

- El modulo recibe la lista de participantes donde se buscara el registro, el ocupado y el registro a buscar, para cada valor de comparación creamos distintas variables que servirán para comparar las cadenas (comparo\_apellido, comparo\_primer\_nombre, comparo\_segundo\_nombre)
- Se declara un contador i=0, y las variables auxiliares para el primer, segundo nombre y apellido del participante buscado
- Los datos del participante se guardan en las variables auxiliares y se llevan a mayúsculas. La lista de participantes se convierte a mayúsculas y se ordena por apellido, primer nombre y segundo nombre usando el método de ordenación **Shell** por estos 3 **criterios**.
- Se declara la variable **buscar** como **true** e inicia la búsqueda con un **while**, comparando los campos. Si un campo comparado es mayor, la búsqueda se detiene de lo contrario, continúa hasta encontrar una coincidencia.
- Si i llega a ocup sin encontrar coincidencia, se devuelve **false**. Si se encuentra la participante la variable **buscar** cambia a **false** y **encontrado** a **true**.
- Finalmente, se devuelve el valor de **encontrado**.

```
bool buscar_participante_by_nombre_apellido(v_participantes &parts, int ocup, t_participante buscado)
{
    int comparo_apellido, comparo_primer_nombre, comparo_segundo_nombre;

    bool encontrado = false;
    int i=0;

    tcad aux_buscado_1er_nombre, aux_buscado_2do_nombre, aux_buscado_apellido;

    shell_apell_nombres(parts, ocup);

    strcpy(aux_buscado_1er_nombre, buscado.primer_nombre);
    strcpy(aux_buscado_2do_nombre, buscado.segundo_nombre);
    strcpy(aux_buscado_apellido, buscado.apellido);

    mayuscula(aux_buscado_1er_nombre);
    mayuscula(aux_buscado_2do_nombre);
    mayuscula(aux_buscado_apellido);

    bool buscar = true;

    while (i <= ocup && buscar) {
        comparo_apellido = strcmp(parts[i].apellido, aux_buscado_apellido);
        comparo_primer_nombre = strcmp(parts[i].primer_nombre, aux_buscado_1er_nombre);
        comparo_segundo_nombre = strcmp(parts[i].segundo_nombre, aux_buscado_2do_nombre);

        if (comparo_apellido > 0 ||
            (comparo_apellido == 0 && comparo_primer_nombre > 0) ||
            (comparo_apellido == 0 && comparo_primer_nombre == 0 && comparo_segundo_nombre > 0)) {
            buscar = false;
        } else if (comparo_apellido == 0 && comparo_primer_nombre == 0 && comparo_segundo_nombre == 0)
            encontrado = true;
        buscar = false;
    }
    i++;
    return encontrado;
}
```

**Aclaración:**

Se utilizó una búsqueda secuencial, en lugar de una búsqueda recursiva por temas de memoria, al realizar la búsqueda en vectores de hasta 1000 registros, la búsqueda recursiva llegaba a consumir toda la memoria del sistema y este se caía o no llegaba a responder en el tiempo esperado devolviendo error.



**Modificar Participante**

El modulo “**modificar\_participante**” fue creado para modificar un participante registrado, Este módulo recibe la lista de participantes (**parts**) de tipo **v\_participante**, el ocupado de participantes (**ocup\_parts**), el **id** a buscar de tipo **int**, la lista de continentes (**conts**) y el ocup de continentes (**ocup\_conts**)

- Se declaran las variables tanto el contador (**i**), como las variables para buscar la posición del registro mediante búsqueda binaria (alto, bajo), Declaramos una bandera (**encontrado**) de tipo **bool**, y un auxiliar de participante (**aux\_participante**)
- Al verificar que el vector tiene registros inicializamos las variables mencionadas y realizamos la búsqueda binaria del **id** en cuestión, dicha búsqueda devolverá el valor de la posición con la cual modificaremos **encontrado**, en caso de que se encuentre la posición en el **rango 0** y ocup
- De ser así, el participante existe y procedemos a escribir **aux\_participante** con los datos del registro y llamamos a “**cargar\_participante\_manual**” si **encontrado** es **false** **no existe** y devolvemos el mensaje de **error**.
- Al terminar la carga **imprime** los datos del participante actualizado

```
void modificar_participante(v_participantes &vec, int ocup_parts, int id, v_continentes conts, int ocup_conts)
{
    int i, alto, bajo;
    bool encontrado = false;
    t_participante aux_participante;

    if(ocup_parts == -1)
        cout << "\nNo existe ningún participante.\n" << endl;
    else
    {
        i = -1;
        bajo = 0;
        alto = ocup_parts;

        buscar_participante_bin_by_id(vec, bajo, alto, id, i);
        encontrado = (i > -1 && i <= ocup_parts);

        if(encontrado == true)
        {
            aux_participante = vec[i];

            cargar_participante_manual(aux_participante, conts, ocup_conts);

            vec[i] = aux_participante;

            cout << "\nSe modificó el participante: \n" << endl;

            imprimirParticipante(vec[i]);
        }
        else
            cout << "\nEl participante con ID: " << id << " no existe...\n" << endl;
    }
}
```

## Gestión de Jurados:

Eliminar Jurado:

EL módulo (“**Borrar**”) que recibe una lista de **jurados** (**v\_jurados**), el ocupado (**ocup**), y el apellido a borrar (**borrar**) está diseñado para eliminar un jurado del vector de jurados basado en el apellido. Estos son sus principales objetivos:

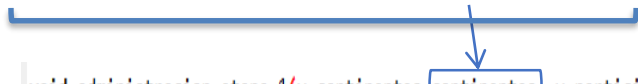
- **Validación de Existencia:** Verifica si hay jurados en el vector. Si no hay jurados, muestra un mensaje indicando que no existe ningún jurado.
- **Conversión de Apellidos a Mayúsculas:** Convierte el apellido ingresado a mayúsculas para asegurar que la comparación no sea sensible a mayúsculas/minúsculas.
- **Búsqueda del Jurado:** El bucle **while** recorre el vector de jurados buscando una coincidencia con el apellido ingresado. Si encuentra una coincidencia, establece una bandera (**enco**) indicando que el jurado a sido encontrado.
- **Eliminación del Jurado:**
  - Si el jurado es encontrado, desplaza todos los jurados posteriores una posición hacia atrás para llenar el espacio dejado por el jurado eliminado.
  - Decremento el contador de jurados (**ocup**) para reflejar la eliminación.
  - Si el jurado es encontrado y eliminado, muestra un mensaje indicando el jurado que ha sido eliminado.
  - Si no se encuentra el jurado, muestra un mensaje indicando que no existe un jurado con el apellido especificado.

```
void Borrar(v_jurados &vec, int &ocup, tcad borrar) //borrar un jurado por su apellido
{
    int i;
    bool enco;
    tcad borrado; //creamos variables auxiliares para realizar la comparación de apellidos de los jurados
    tcad auxApellido;
    if(ocup == -1)
    {
        cout << "\nNo existe ningún jurado.\n" << endl; //validamos que exista al menos un jurado en el vector
    }
    else
    {
        enco = false;
        i = 0;
        strcpy(borrado, borrar); //Llevamos a mayusculas el apellido ingresado
        mayuscula(borrado);
        while((i <= ocup) && !enco)
        {
            strcpy(auxApellido, vec[i].apellido); //aquí llevamos a mayusc. el apellido del jurado del vector y procedemos a comparar
            mayuscula(auxApellido);
            if(strcmp(borrado, auxApellido) == 0)
            {
                cout << "Se eliminó al jurado: " << endl;
                Mostrar_Jurados(vec, i); //si el jurado existe mostramos el jurado que se eliminará en la posición de (i)
                enco = true;
            }
            else
            {
                i++;
            }
        }
        if(enco == true)
        {
            while(i < ocup)
            {
                vec[i] = vec[i + 1];
                i++;
            }
            ocup--;
        }
        else
        {
            cout << "\nNo existe jurado con ID: " << borrar << endl;
        }
    }
    system("pause");
}
```

## Gestión de la 1era Etapa:

A continuación, describiremos lo más importante del módulo “**administracion\_etapa\_1**”. Este módulo se encarga de gestionar y validar el correcto funcionamiento de la ETAPA\_1. Empezaremos describiendo cada parte del código para un mejor entendimiento.

`continentes = {"AFRICA", "AMERICA", "ASIA", "EUROPA", "OCEANIA"};` → viene cargado con estos valores desde el main



```
void administracion_etapa_1(v_continentes continentes, v_participantes &participantes, int ocup_participantes, int vAfrica, int vAmerica,
    int vAsia, int vEuropa, int vOceania, v_jurados jurados, int ocup_jurados, bool control_jurados, bool control_participante,
    v_participantes &seleccionadas, int &ocup_seleccionadas, bool &ctr_1erEtapa);
```

- La variable participante es de tipo **v\_participantes**, es un vector de registro. La variable **ocup\_participantes** representa la cantidad de participantes registradas,
- Son contadores que vienen desde el main con valores cargados (**vAfrica, vAmerica, vAsia, vEuropa, vOceania**); que se encargaran de validar que existan 4 participantes como mínimo por cada continente.
- **jurados es de tipo v\_jurados**, es un vector de registro, este contiene la info de los jurados, **ocup\_jurados** representa la cantidad de jurados registrados.
- **control\_jurados, control\_participante y control\_etapa1** son banderas necesarias para el correcto funcionamiento de la etapa\_1.  
 “**control\_participante**” si es false significa que no se registraron 30 participantes como mínimo, de lo contrario si.  
 “**control\_jurados**” si es false significa que no se registraron 3 jurados como mínimo, de lo contrario sí. Después el control de los contadores de los continentes.

Si las banderas (**control\_participante, control\_jurados**) son verdadera y los contadores **vAfrica, vAmerica, vAsia, vEuropa, vOceania**; son **>=4** entonces **control\_etapa1** cambia a verdadero,

```
else if (vAfrica >= 4 && vAmerica >= 4 && vAsia >= 4 && vEuropa >= 4 && vOceania >= 4){ //validar q existan minimo 4 participante de cada continentes

    control_etapa1 = true;
    cout << "--Clasificacion 1er Etapa--" << endl;
    clasificar_participantes(continentes, participantes, ocup_participantes, jurados, ocup_jurados, seleccionadas, ocup_seleccionadas);
}
```

De esta forma garantizamos que todo está correcto y llamamos al módulo “**clasificar\_participantes**” para iniciar con evaluación de cada participante.

**Modulo “clasificar participantes”**

- El propósito de esta parte del código es recorrer toda la lista de **participantes** y reiniciar los acumuladores de ciertas notas a cero antes de calcular nuevas sumas. Esta acción es necesaria para asegurarse de que los acumuladores no contengan residuos de cálculos anteriores y se pueda empezar desde un valor limpio.
 

```
for(i = 0; i <= ocup_participantes; i++) //Recorrido de toda la lista de par
{
    //antes de calcular las sumas debemos iniciar los acumuladores en 0
    participantes[i].etapa1.suma_nota_vestuario = 0;
    participantes[i].etapa1.suma_nota_elegancia = 0;
    participantes[i].etapa1.suma_nota_elocuencia = 0;
    participantes[i].etapa1.suma_nota_idiomas = 0;
}
```
- El propósito de esta parte del código es recorrer la lista de jurados para cada participante y realizar varias operaciones, como copiar los datos del jurado, generar puntajes aleatorios para cada criterio de evaluación y sumar estos puntajes en los acumuladores correspondientes de la participante.
 

```
for(j = 0; j <= ocup_jurados; j++) //Recorrido de la lista de jurados
{
    //copiar datos del jurado
    participantes[i].etapa1.notas_jurado[j].ID_jurado = jurados[j].ID_jurado; //copiamos el id del jurado
    strcpy(participantes[i].etapa1.notas_jurado[j].apellido_jurado, jurados[j].apellido); //copiamos el apell
    strcpy(participantes[i].etapa1.notas_jurado[j].nombre_jurado, jurados[j].nombre); //copiamos el nombre

    //generamos valores aleatorios como puntajes individuales
    participantes[i].etapa1.notas_jurado[j].nota_vestuario = aleatorio(0, 100);
    participantes[i].etapa1.notas_jurado[j].nota_elegancia = aleatorio(0, 100);
    participantes[i].etapa1.notas_jurado[j].nota_elocuencia = aleatorio(0, 100);
    participantes[i].etapa1.notas_jurado[j].nota_idiomas = (participantes[i].cant_idiomas / 6.0 * 100); //La

    //sumar las notas de cada categoría
    participantes[i].etapa1.suma_nota_vestuario += participantes[i].etapa1.notas_jurado[j].nota_vestuario;
    participantes[i].etapa1.suma_nota_elegancia += participantes[i].etapa1.notas_jurado[j].nota_elegancia;
    participantes[i].etapa1.suma_nota_elocuencia += participantes[i].etapa1.notas_jurado[j].nota_elocuencia;
    participantes[i].etapa1.suma_nota_idiomas += participantes[i].etapa1.notas_jurado[j].nota_idiomas;
}

// promediamos las notas parciales
participantes[i].etapa1.suma_nota_vestuario = participantes[i].etapa1.suma_nota_vestuario / cantidad_jurados;
participantes[i].etapa1.suma_nota_elegancia = participantes[i].etapa1.suma_nota_elegancia / cantidad_jurados;
participantes[i].etapa1.suma_nota_elocuencia = participantes[i].etapa1.suma_nota_elocuencia / cantidad_jurados;
participantes[i].etapa1.suma_nota_idiomas = participantes[i].etapa1.suma_nota_idiomas / cantidad_jurados;

//al finalizar sumamos las notas promediadas para obtener un puntaje total de la participante
participantes[i].etapa1.puntaje_total =
participantes[i].etapa1.suma_nota_vestuario +
participantes[i].etapa1.suma_nota_elegancia +
participantes[i].etapa1.suma_nota_elocuencia +
participantes[i].etapa1.suma_nota_idiomas;
```
- El propósito de esta parte del código es calcular el promedio de cada participante para las diferentes categorías de evaluación y luego sumar estas notas promediadas para obtener un puntaje total para cada participante.

Filtra y cuenta los participantes por continente.

Ordena y agrega a la lista de seleccionadas

Muestra las 20 participantes con mejor puntaje

```
for (i=0;i<=4;i++){ //5 continentes
    pertenece_continente(participantes, ocup_participantes, aux_continente, ocup_aux, continentes[i]);
    Ordenacion_Continentes(aux_continente, ocup_aux, seleccionadas, ocup_seleccionadas);
}

cout << "\n---* Se guardaron las: " << ocup_seleccionadas + 1 << " mejores del certamen!... *---" << endl;

Insercion_primera(seleccionadas, ocup_seleccionadas); //ordenamos las 20 mejores

for (i = 0; i <= ocup_seleccionadas; i++)
{
    mostrar_Participante_etapa1(seleccionadas[i]);
}
```

Por ejemplo la variable “c” ingresa con la cadena “AMERICA”, si el continente existe en la lista de continentes definida entonces “ocup\_conti” incrementa y lo asigna a la variable auxiliar “aux\_continente”

```
void pertenece_continente(v_participantes list_part, int ocup_part, v_participantes &vconti, int &ocup_conti, tcad c)
{
    ocup_conti = -1;
    int i;
    for(i = 0; i <= ocup_part; i++) //recorre todo el vec de participantes
    {
        if(strcmp(list_part[i].continente, c) == 0)
        {
            ocup_conti++;
            vconti[ocup_conti] = list_part[i]; //asigono una participante a su respectivo continente
        }
    }
}
```

que volverá, en este caso con todas las participantes de “America”. Por ultimo esta lista auxiliar “aux\_continente” ya cargadas con todas las participantes de “America” es enviada al módulo “Ordenacion\_Continentes”.

### Modulo “Ordenacion\_Continentes”:

Diagram illustrating the flow of data into the `Ordenacion_Continentes` module:

- aux\_continente** (Auxiliary list for a continent) is passed as `lista_auxiliar`.
- Seleccionadas va acumular las 4 mejores de cada continente** (Selected participants accumulate the 4 best of each continent) is passed as `seleccionadas`.

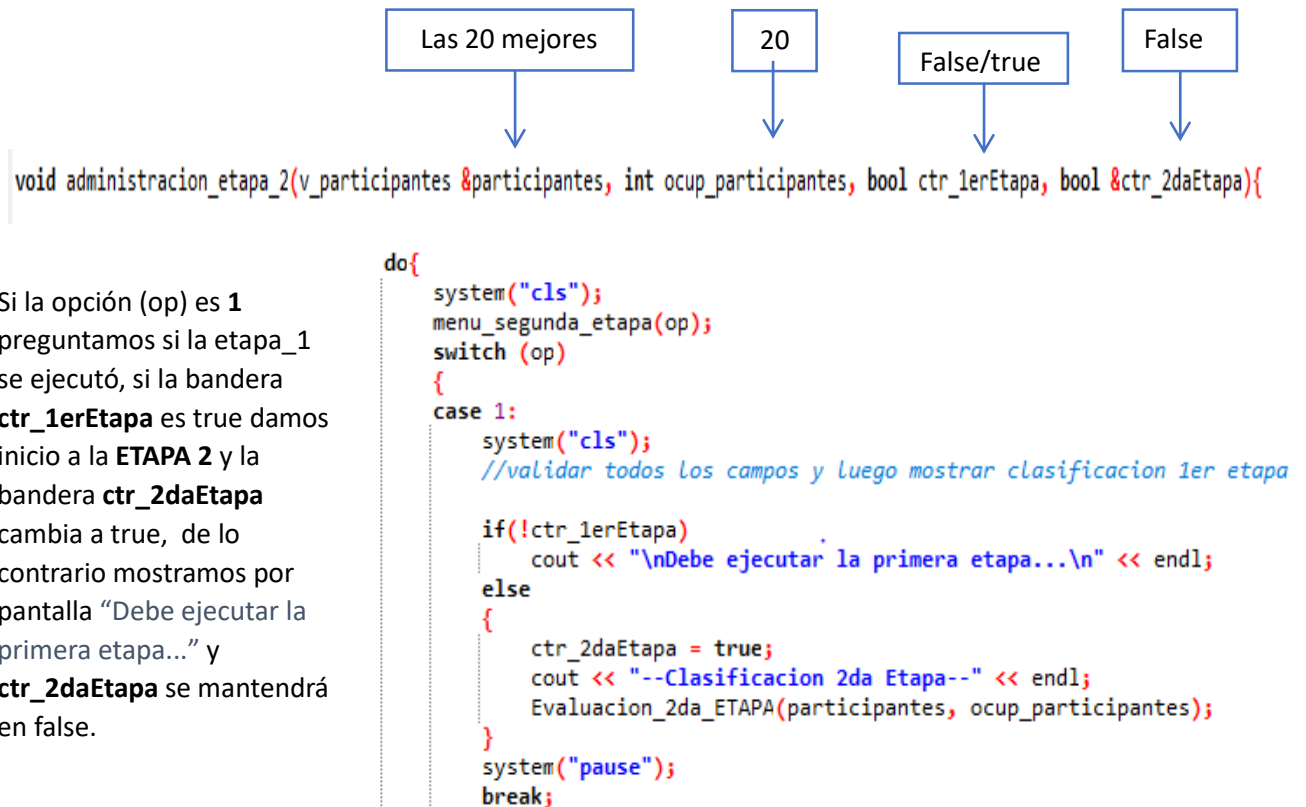
```
void Ordenacion_Continentes(v_participantes &lista_auxiliar, int ocup_auxiliar, v_participantes &seleccionadas, int &ocup_seleccionadas)
{
    int i;
    //ordena los participantes por puntaje total
    Insercion_primera(lista_auxiliar, ocup_auxiliar);
    //me quedo con las primeras cuatro participantes
    for(i = 0; i <= 3; i++)
    {
        ocup_seleccionadas++;
        seleccionadas[ocup_seleccionadas] = lista_auxiliar[i]; //EJ: africa, Seleccionadas se va quedando con las 4 de africa.
    }
}
```

Siguiendo con el ejemplo de las participantes de **áfrica** el modulo “Ordenacion\_Continentes” invoca al módulo “Insercion\_primera” que se encarga de ordenar la lista auxiliar “aux\_continente” (contiene a las participantes de **africa**) por puntaje total de forma decreciente. Por ultimo una vez ya ordenada lista auxiliar “aux\_continente” ingresara al bucle FOR y la variable “seleccionadas” se quedara con las 4 primeras participantes con mayor puntaje de **africa**.

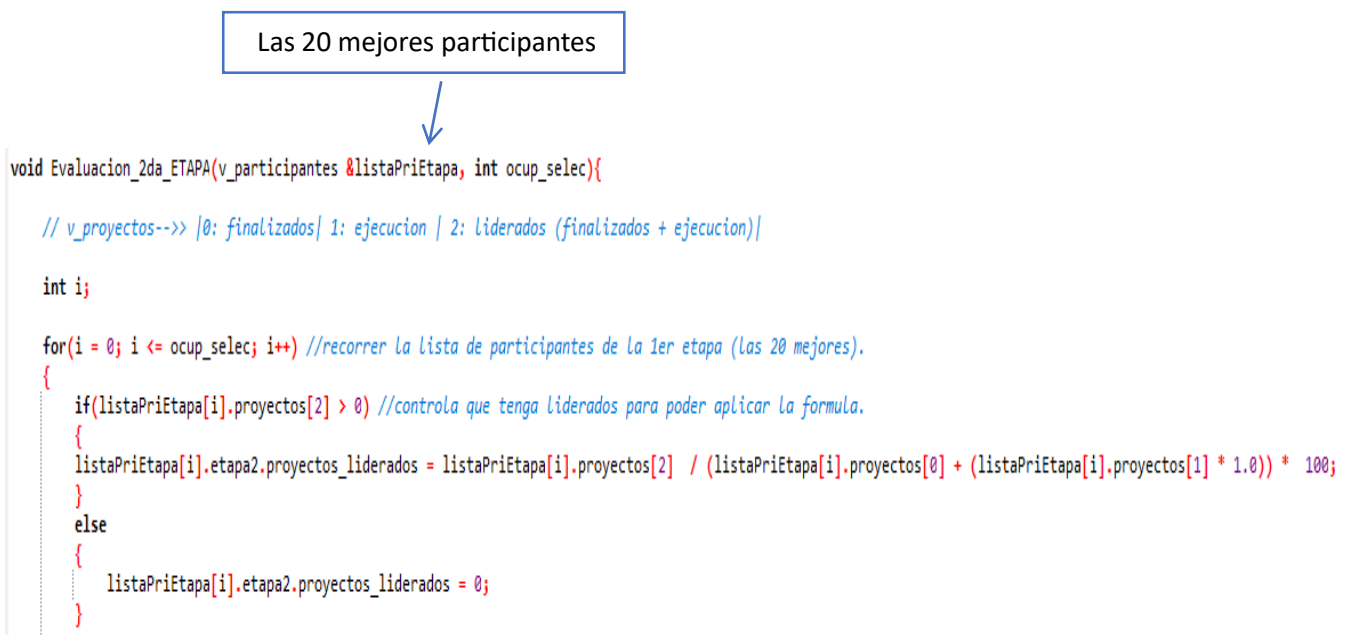
Este mecanismo se repite para los 4 continentes.

## Gestión de la 2da Etapa:

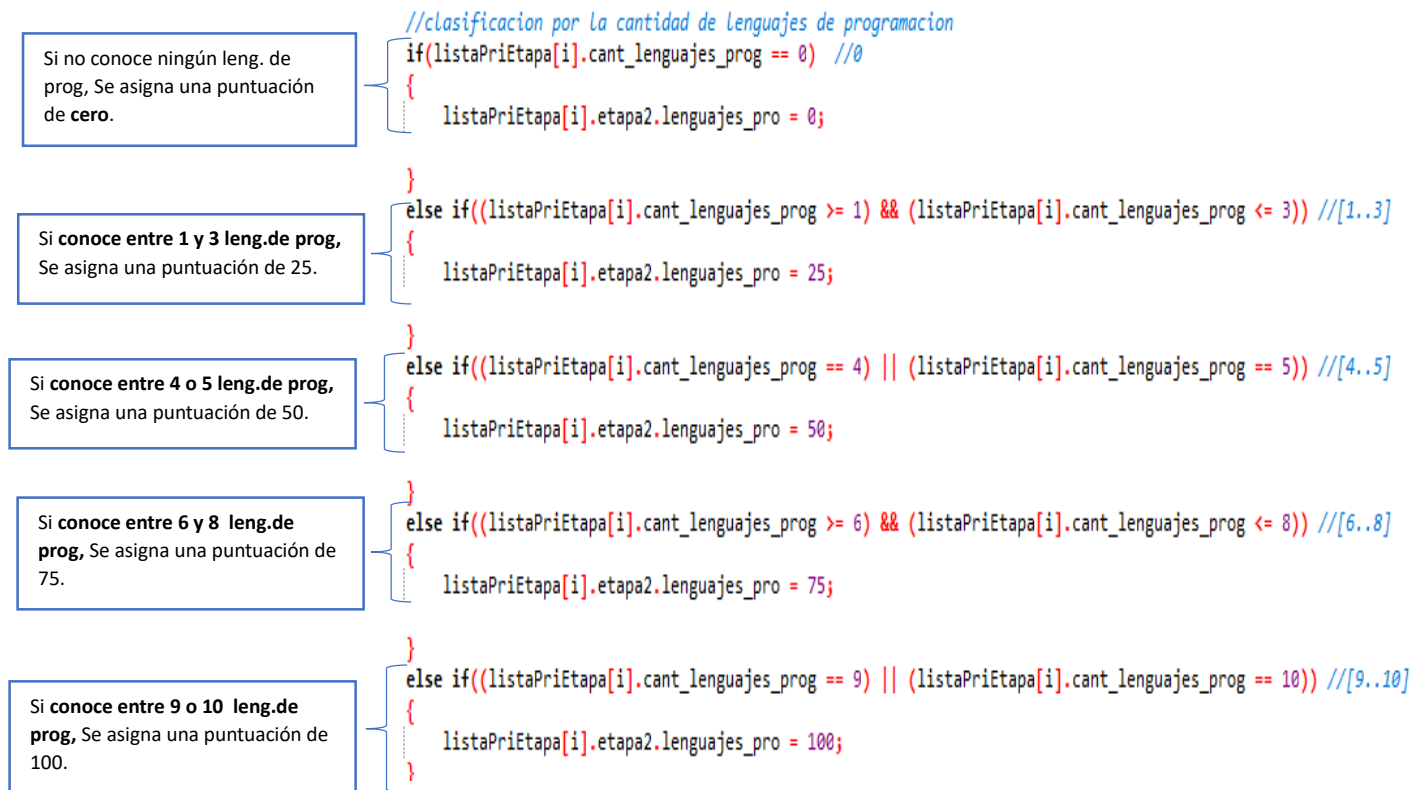
A continuación, describiremos lo más importante del módulo “**administracion\_etapa\_2**”. Este módulo se encarga de gestionar y validar el correcto funcionamiento de la ETAPA\_2. Empezaremos describiendo cada parte del código para un mejor entendimiento.



Ahora pasaremos a explicar el procedimiento “**Evaluacion\_2da\_ETAPA**”: Este módulo se encarga de evaluar a las 20 mejores participantes.



- El propósito de esta parte del código es **clasificar a los participantes en función de la cantidad de lenguajes de programación que conocen** y asignarles una puntuación correspondiente en la segunda etapa. A continuación, se detalla cómo funciona:



- El propósito de esta parte del código es **clasificar a los participantes según su experiencia laboral** (medida en años) y asignarles una puntuación correspondiente en la segunda etapa. Aquí se detalla cómo funciona:

```

//clasificacion por años de experiencia
if(listaPriEtapa[i].anios_experiencia == 0) //0
{
    listaPriEtapa[i].etapa2.experiencia = 0;
}
else if(listaPriEtapa[i].anios_experiencia >= 1 && listaPriEtapa[i].anios_experiencia <= 5) //[1..5]
{
    listaPriEtapa[i].etapa2.experiencia = 25;
}
else if(listaPriEtapa[i].anios_experiencia >= 6 && listaPriEtapa[i].anios_experiencia <= 10) //[6..10]
{
    listaPriEtapa[i].etapa2.experiencia = 50;
}
else if(listaPriEtapa[i].anios_experiencia >= 11 && listaPriEtapa[i].anios_experiencia <= 15) //[11..15]
{
    listaPriEtapa[i].etapa2.experiencia = 75;
}
else if(listaPriEtapa[i].anios_experiencia >= 16 && listaPriEtapa[i].anios_experiencia <= 20) //[16..20]
{
    listaPriEtapa[i].etapa2.experiencia = 100;
}

```



- Esta parte de código clasifica a los participantes y les asigna una puntuación basada en la cantidad de proyectos que han finalizado. Cada rango de proyectos finalizados tiene una puntuación específica, lo cual ayuda a evaluar de manera estructurada y justa a los participantes en la competencia.

```
//clasificación por cantidad de Proyectos finalizados
if(listaPriEtapa[i].proyectos[0] == 0) //0
{
    listaPriEtapa[i].etapa2.proyectos_finalizados = 0;
}
else if(listaPriEtapa[i].proyectos[0] >= 1 && listaPriEtapa[i].proyectos[0] <= 6) //[1..6]
{
    listaPriEtapa[i].etapa2.proyectos_finalizados = 25;
}
else if(listaPriEtapa[i].proyectos[0] >= 7 && listaPriEtapa[i].proyectos[0] <= 12) //[7..12]
{
    listaPriEtapa[i].etapa2.proyectos_finalizados = 50;
}
else if(listaPriEtapa[i].proyectos[0] >= 13 && listaPriEtapa[i].proyectos[0] <= 20) //[13..20]
{
    listaPriEtapa[i].etapa2.proyectos_finalizados = 75;
}
else if(listaPriEtapa[i].proyectos[0] >= 21 && listaPriEtapa[i].proyectos[0] <= 25) //[21..25]
{
    listaPriEtapa[i].etapa2.proyectos_finalizados = 100;
}
```

- Esta parte de código clasifica a los participantes y les asigna una puntuación basada en la cantidad de proyectos que tienen actualmente en ejecución. Cada cantidad específica de proyectos en ejecución tiene una puntuación asignada, lo que ayuda a evaluar de manera estructurada y justa a los participantes en la competencia.

```
//clasificación por cantidad de Proyectos en ejecución
if(listaPriEtapa[i].proyectos[1] == 0) //0
{
    listaPriEtapa[i].etapa2.proyectos_en_ejecucion = 0;
}
else if(listaPriEtapa[i].proyectos[1] == 1) //1
{
    listaPriEtapa[i].etapa2.proyectos_en_ejecucion = 25;
}
else if(listaPriEtapa[i].proyectos[1] == 2) //2
{
    listaPriEtapa[i].etapa2.proyectos_en_ejecucion = 50;
}
else if(listaPriEtapa[i].proyectos[1] == 3) //3
{
    listaPriEtapa[i].etapa2.proyectos_en_ejecucion = 75;
}
else if(listaPriEtapa[i].proyectos[1] == 4) //4
{
    listaPriEtapa[i].etapa2.proyectos_en_ejecucion = 100;
}
```

Se suman todos los puntajes obtenidos anteriormente y se obtiene la **suma\_total**.

```
//Al finalizar se realiza la suma total del puntaje de cada categoria.
listaPriEtapa[i].etapa2.suma_total =
listaPriEtapa[i].etapa2.proyectos_liderados +
listaPriEtapa[i].etapa2.lenguajes_pro +
listaPriEtapa[i].etapa2.experiencia +
listaPriEtapa[i].etapa2.proyectos_finalizados +
listaPriEtapa[i].etapa2.proyectos_en_ejecucion;
```

```
Insertion_segunda(listaPriEtapa, ocup_selec); //ordenación por suma total.
```

**Al finalizar el bucle FOR** se realiza la ordenación que es de acuerdo a la **suma\_total** obtenida.

```
for (i = 0; i <= ocup_selec; i++)
{
    mostrar_Participante_etapa2(listaPriEtapa[i]); //muestra las 20 mejores con orden decreciente.
}
```



### Coronacion:

La fase de coronación se maneja desde un módulo de “**administración\_coronacion**” donde recibe los vectores de 20 finalistas (“**participantes**”), su ocupado, el vector de finalistas (“**finalistas**”) y bool “**ctrCoronacion**” este módulo realiza la suma de los puntajes obtenidos en la primera y segunda etapa realizando el calculo en el modulo “**evaluacion\_Coronacion**” siempre y cuando se realicen ambas etapas (se valida con la variable **ctrCoronacion**) al finalizar, se habilita el modulo de mostrar finalistas

- Se crea una variable bandera (“**coronar**”) en false, que valida que primero se realice la evaluación para poder mostrar los finalistas en la opcion 2 del menú
- El modulo inicia con un **do{ } while**, que muestra el menú de la gestión de coronación y finaliza con la opcion 3 (Volver al menú ppal)
- Si cumple con la clasificación de primera y segunda etapa, podemos realizar la evaluación (opcion 1), caso contrario nos indicará que deben realizar la segunda, y anteriormente primera etapa
- Luego de realizar la evaluacion se puede mostrar la ganadora del certamen y las 4 finalistas (opcion 2)
- Cualquier otra opcion distinta de 3 dará error e indicará el mensaje “**opcion invalida**”

```
void administracion_coronacion(v_participantes &participantes, int ocup_participantes, v_finalistas &finalistas, int &ocup_finalistas, bool ctrCoronacion)
{
    bool coronar = false;
    int op;
    do
    {
        system("cls");
        menu_coronacion(op);
        switch (op)
        {
            case 1:
                system("cls");
                if(!ctrCoronacion)
                    cout << "\nDebe ejecutar la segunda etapa...\n" << endl;
                else
                {
                    evaluacion_Coronacion(participantes, ocup_participantes, finalistas, ocup_finalistas);
                    coronar = true;
                }
                system("pause");
                break;
            case 2:
                system("cls");
                if(coronar){
                    mostrar_finalistas(finalistas);
                }else{
                    cout << "\nDebe realizar coronacion..\n" << endl;
                }
                system("pause");
                break;
            case 3:
                break;
            default:
                cout << "Opción inválida" << endl;
                system("pause");
                break;
        }
    } while (op != 3);
}
```

**Evaluación de coronación:**

El módulo de **evaluacion\_coronacion** se encarga de realizar la evaluación de las 20 finalistas (**listaPriEtapa**) que se clasificaron en ambas etapas, y guardarlas en una **listaFinalistas** con la ganadora y las finalistas

- Se declara un contador **i**, posterior se recorren las 20 participantes (indicadas por el **ocup\_selec**) y por cada participante vamos sumando los puntajes con la formula:  $(\text{puntajeEt1} * 0,33 + \text{PuntajeEt2} * 0,67)$
- Al finalizar llamamos al módulo **guardar\_finalistas** (para guardar las finalistas)
- Terminando de evaluar las 5 candidatas finales (ganadora y finalistas), se llama al módulo ("**mostrar\_finalistas**") que se encarga de mostrar los resultados de todo el certamen.

```
void evaluacion_Coronacion(v_participantes &listaPriEtapa, int ocup_selec, v_finalistas &listaFinalistas, int &ocup_Finalistas)
{
    int i;
    for(i = 0; i <= ocup_selec; i++)
    {
        listaPriEtapa[i].puntaje_Coronacion = 0;
        listaPriEtapa[i].puntaje_Coronacion = (listaPriEtapa[i].etapa1.puntaje_total * 0.33) + (listaPriEtapa[i].etapa2.suma_total * 0.67);
    }
    guardar_finalistas(listaPriEtapa, ocup_selec, listaFinalistas, ocup_Finalistas); //se obtiene las 5 mejores y se guarda en la lista de
    mostrar_finalistas(listaFinalistas);
}
```

**Guardamos las finalistas:**

El módulo **guardar\_finalistas** se encarga de ordenar las 20 participantes por puntaje de coronación (utilizando un modulo de **inserción\_Coronacion** que ordenará decreciente por **puntajeCoronacion**), y luego recorrer dicha lista ordenada obteniendo las 5 mejores y llevándolas a la **listaFinalistas** y actualizando la variable del ocupado de finalistas (**ocup\_finalistas**)

```
void guardar_finalistas(v_participantes &lista_2da_etapa, int ocup_2da_etapa, v_finalistas &finalistas, int &ocup_finalistas)
{
    int i;
    //ordena los participantes por resultado final
    Insercion_coronacion(lista_2da_etapa, ocup_2da_etapa);
    for(i = 0; i <= 5; i++)
    {
        ocup_finalistas++;
        finalistas[ocup_finalistas] = lista_2da_etapa[i];
    }
}
```

**Mostramos las finalistas**

El modulo **mostrar\_finalistas**, se encarga de mostrar los resultados de la ganadora final del certamen (la finalista en la **posición 0**) y las 4 finalistas recorriendo el vector de finalistas con un **for** que va desde el índice **i =1 hasta i<= 4**, En cada recorrido vamos mostrando el lugar de la finalista en la posición de **i**.

```
void mostrar_finalistas(v_finalistas finalistas, int &ocup_finalistas)
{
    int i;

    cout << "\n*****\n";
    cout << "* LA GANADORA ES... *" << endl;;
    mostrar_Participante_Coronacion(finalistas[0]);
    cout << "\n*****\n" << endl;

    cout << "\n*****\n";
    cout << "* LAS FINALISTAS SON... *" << endl;;
    for(i = 1; i <= 4; i++)
    {
        cout << i << " ) LUGAR" << endl;
        mostrar_Participante_Coronacion(finalistas[i]);
        cout << "\n*****\n" << endl;
    }
}
```

## Pruebas

Las pruebas permiten detectar la presencia de errores tanto en la estructura del programa como en la lógica de solución del problema. Para ello, pueden definirse 3 tipos de pruebas:

### Pruebas unitarias

<b>ID Caso de Prueba: 001</b>			
<b>Requisitos de la prueba:</b> El vector de continentes debe tener continentes cargados, el ocupado de continentes debe ser mayor a -1			
<b>Propósito de la prueba:</b> Realizar generación de datos aleatorios para crear un participante			
<b>Descripción de las acciones y/o condiciones para las Pruebas</b>			
Nº	Acciones (entradas)	Salida Esperada	Salida Obtenida
01	cargar_participante_auto()  <i>Se ingresa el registro de tipo participante por referencia, se ingresa el vector de continentes, se ingresa el ocupado de continentes apuntando al último continente.</i>	El sistema debe generar el participante con los campos primer nombre, segundo nombre, edad, altura, cant. De idiomas, años de experiencia, proyectos finalizados, ejecución y liderados con valores aleatorios.	El sistema realiza la carga del registro de tipo participante con los campos primer nombre, segundo nombre, edad, altura, cant. De idiomas, años de experiencia, proyectos finalizados, ejecución y liderados con valores aleatorios.
<b>Resultados obtenidos</b>			
<b>Resultado (Ap/Desap): Aprobado</b>			
<b>Observaciones:</b> -			

### ID Caso de Prueba: 001

01 (Se visualizan los datos aleatorios cargados para el registro de tipo participante):

```

-----
ID: 1001
Apellido: Velazques
Primer Nombre: Alexis
Segundo Nombre: Leonarda
Edad: 35 años
Altura: 1.55 mts.
Cantidad de Idiomas: 3
Continente: EUROPA
Cantidad de Lenguajes de Programación: 4
Años de Experiencia: 18 años
Proyectos Finalizados: 20
Proyectos En ejecución: 0
Proyectos Liderados: 8
-----
  
```

**ID Caso de Prueba: 002**

**Requisitos de la prueba:** El vector de continentes debe tener continentes cargados, el ocupado de continentes debe ser mayor a -1

**Propósito de la prueba:** Solicitar al usuario el ingreso de los datos para generar un participante

**Descripción de las acciones y/o condiciones para las Pruebas**

Nº	Acciones (entradas)	Salida Esperada	Salida Obtenida
01	cargar_participante_manual()  <i>Se tipea una cadena vacia en el campo apellido del participante</i>	El sistema debe indicar que el apellido ingresado es invalido y solicitar nuevamente el apellido	El sistema indica que el apellido ingresado es invalido y solicita nuevamente el apellido
02	cargar_participante_manual()  <i>Se ingresa el registro de tipo participante por referencia, se ingresa el vector de continentes, se ingresa el ocupado de continentes apuntando al último continente.</i>	El sistema debe solicitar el ingreso de datos del participante para los distintos campos: primer nombre, segundo nombre, edad, altura, cant. De idiomas, años de experiencia, proyectos finalizados, ejecución y liderados.	El sistema solicita el ingreso de datos del participante para los distintos campos, y genera un nuevo participante con los datos cargados.
03	cargar_participante_manual()  <i>Se tipea el nombre de la participante en el campo nombre del participante</i>	El sistema debe validar que el nombre tiene una longitud mayor a 3 y guardar el nombre en el campo nombre.	El sistema valida que el nombre cumple con la cantidad mínima de caracteres y guarda el nombre en el campo nombre.

**Resultados obtenidos**

**Resultado (Ap/Desap):** Aprobado

**Observaciones:** -

**ID Caso de Prueba: 002**

01:

```
C:\Users\MI PC\Desktop\todc x + v
Ingrese Apellido:
Ingresó un apellido inválido
Presione una tecla para continuar . . .
Ingrese Apellido: Alfaro
```

Ingresamos  
apellido invalido,  
vacío o con  
números

02, 03

```
C:\Users\MI PC\Desktop\todc x + v
Ingrese Apellido:
Ingresó un apellido inválido
Presione una tecla para continuar . . .
Ingrese Apellido: Alfaro
Ingrese Primer Nombre: Josue
Ingrese Segundo Nombre: Dario
Ingrese Edad (años): 26
Ingrese Altura (mts): 1.70
Ingrese Cant. Idiomas que maneja: 2
Ingrese Continente: America
Ingrese Cant. de Lenguajes de programacion que conoce: 1
Ingrese Años de Experiencia que posee: 2
Ingrese Cant. de Proyectos en ejecución: 1
Ingrese Cant. de Proyectos finalizados: 2
Ingrese Cant. de Proyectos liderados: 1
Desea seguir ingresando participantes? s/n:
n

Se cargaron: 9 participantes de AFRICA
Se cargaron: 6 participantes de AMERICA
Se cargaron: 0 participantes de ASIA
Se cargaron: 8 participantes de EUROPA
Se cargaron: 8 participantes de OCEANIA
```

Ingresamos  
Nombre valido, lo  
cual permite llenar  
los otros campos

**ID Caso de Prueba: 003**

**Requisitos de la prueba:** El vector de Participantes debe tener al menos una participante.

**Propósito de la prueba:** Realizar el listado recursivo de los participantes cargados

**Descripción de las acciones y/o condiciones para las Pruebas**

Nº	Acciones (entradas)	Salida Esperada	Salida Obtenida
01	listar_participantes_rec()  <i>Se ingresa un vector de participantes cargados, Se ingresa el ocupado apuntando al ultimo participante</i>	El sistema debe listar recursivamente los participantes hasta llegar al primero.	El sistema lista recursivamente los participantes hasta llegar al primero.
02	listar_participantes_rec()  <i>Se ingresa un vector de participantes vacío, Se ingresa el ocupado en -1</i>	El sistema debe indicar que la lista de participantes está vacía	El sistema indica "Lista de participantes vacía."

**Resultados obtenidos**

**Resultado (Ap/Desap):** Aprobado

**Observaciones:** Al listar recursivamente demasiados registros, el sistema puede no llegar a responder.

**Caso de Prueba: 003**

01:

Listamos un vector  
con registros, lo  
cual es valido

```
C:\Users\MI PC\Desktop\todc X + v
---Lista de Participantes---
-----
ID: 1000
Apellido: Velazques
Primer Nombre: Anya
Segundo Nombre: Aba
Edad: 31 años
Altura: 1.72 mts.
Cantidad de Idiomas: 2
Continente: AFRICA
Cantidad de Lenguajes de Programación: 1
Años de Experiencia: 14 años
Proyectos Finalizados: 25
Proyectos En ejecución: 4
Proyectos Liderados: 10
-----
ID: 1001
Apellido: Velazques
Primer Nombre: Alexis
Segundo Nombre: Leonarda
Edad: 35 años
Altura: 1.55 mts.
Cantidad de Idiomas: 3
Continente: EUROPA
Cantidad de Lenguajes de Programación: 4
Años de Experiencia: 18 años
Proyectos Finalizados: 20
Proyectos En ejecución: 0
Proyectos Liderados: 8
-----
ID: 1002
Apellido: Renteria
Primer Nombre: Alexis
Segundo Nombre: Josefa
Edad: 40 años
Altura: 1.8 mts.
Cantidad de Idiomas: 3
Continente: AFRICA
Cantidad de Lenguajes de Programación: 2
Años de Experiencia: 13 años
Proyectos Finalizados: 13
Proyectos En ejecución: 3
Proyectos Liderados: 5
-----
```

02:

Queremos listar  
recursivamente el  
vector vacío

```
C:\Users\MI PC\Desktop\todc X + v
---Lista de Participantes---
Lista de participantes vacía.
-----
Presione una tecla para continuar . . . |
```

## Pruebas de integración

**ID Caso de Prueba: IT-01**

**Requisitos de la prueba:** El vector de Participantes debe tener espacio disponible para agregar participantes, los continentes deben estar cargados.

**Propósito de la prueba:** Realizar prueba de integración de todos los módulos que permiten cargar participantes, ingresando desde el menú principal hasta la gestión de participantes a registro, tanto carga automática, como manual, Allí, demostrar que ambos módulos pueden convivir en el sistema, al finalizar listaremos recursivamente los participantes creados y eliminaremos uno de ellos.

**Descripción de las acciones y/o condiciones para las Pruebas**

Nº	Acciones (entradas)	Salida Esperada	Salida Obtenida
01	main()  <i>Se ingresa la opcion 9</i>	El sistema debe indicar que la opción es inválida	El sistema indica que la opcion es invalida.
02	main()  <i>Se ingresa la opcion 1: administracion_participantes() con lista de participantes, continentes, el ocupado de participantes, id base, contadores de participantes por continente, y bandera de participantes válidos.</i>	El sistema debe mostrar un menú con las distintas opciones para gestionar participantes	El sistema muestra un menú con las distintas opciones para gestionar participantes
03	menu_gestion_participantes()  <i>Se ingresa opcion: 7</i>	El modulo debe indicar que es opcion invalida	El modulo indica que la opcion es invalida
04	menu_gestion_participantes()  <i>Se ingresa opcion: 1 "Registro de participantes"</i>	El sistema debe solicitar que tipo de registro quiere usar el usuario (1 Auto, 2 Manual)	El sistema Muestra los tipos de registros (1 Auto, 2 Manual) y espera la eleccion
05	--Registrar Participante--  <i>Se ingresa tipo de registro: 3</i>	El sistema debe indicar que el tipo de registro es invalido	El sistema indica que el tipo de registro es invalido esperando tipo 1, o 2.
06	--Registrar Participante--  <i>Se ingresa tipo de registro 1, y cantidad de participantes a crear 20</i>	El sistema debe indicar que el valor 20 está fuera del rango (30 a 1000)	El sistema le indica al usuario que el valor está fuera del rango y que debe ingresar 30 a 1000 participantes
07	--Registrar Participante--  <i>Se ingresa cantidad de participantes a crear 200</i>	El sistema debe validar que hay espacio suficiente para crear 200 participantes y llamar al módulo de generar participantes	El sistema valida que existe espacio para los participantes, muestra el mensaje de que están por crearse los participantes y llama al módulo encargado de la tarea.
08	generar_participantes_automaticos()	El modulo debe generar los "N" nuevos participantes con	El modulo genera N participantes con aleatorios y



	<i>Se ingresa el id base, cantidad de N participantes generar, vector de parts y continentes con sus ocupados, y el registro nuevo a rellenar.</i>	valores aleatorios, verificar que no esté repetido y guardarlos en el vector, al finalizar actualizar la id base.	luego de verificar que no exista los inserta e incrementa la id
09	--Registrar Participante— <i>Se devuelve el vector cargado con los participantes aleatorios</i>	El sistema debe incrementar los contadores de participante por continente y mostrar cuantos se crearon por continente	El sistema muestra por pantalla cuantos participantes se crearon por continente
10	menu_gestion_participantes() <i>Se ingresa opcion: 1 “Registro de participantes”</i>	El sistema debe solicitar que tipo de registro quiere usar el usuario (1 Auto, 2 Manual)	El sistema Muestra los tipos de registros (1 Auto, 2 Manual) y espera la elección
11	--Registrar Participante— <i>Se ingresa tipo de registro: 2</i>	El sistema debe llamar al módulo encargado de la creación manual de participantes	El sistema llama al módulo encargado de crear participantes manual: generar_participantes_manual
12	generar_participantes_manual() <i>Se ingresa el id base, vector de participantes y continentes con sus ocupados, y el registro nuevo a rellenar.</i>	El modulo debe solicitar al usuario los datos del nuevo participante, apellido, primer y segundo nombre, etc, al finalizar, validar que el usuario no exista, de ser así, insertarlo en la lista y actualizar la id base	El modulo le solicita al usuario los datos del nuevo participante, apellido, primer y segundo nombre, etc, al finalizar, valida que el usuario no exista, de ser así, insertarlo en la lista y actualiza la id base
13	--Registrar Participante— <i>Se devuelve el vector cargado con los participantes</i>	El sistema debe incrementar los contadores de participante por continente y mostrar cuantos se crearon por continente	El sistema muestra por pantalla cuantos participantes se crearon por continente
14	menu_gestion_participantes() <i>Se ingresa opcion: 5 “Listar participantes”</i>	El sistema debe llamar al modulo encargado de la tarea listar_participantes_rec	El sistema debe llama al modulo encargado de la tarea listar_participantes_rec
15	listar_participantes_rec() <i>Se ingresa la lista de participantes cargada, y el ocupado</i>	El sistema debe listar recursivamente los participantes	El sistema lista recursivamente los participantes mostrando para cada uno, el id, nombres, apellido, edad, altura, continente, proyectos, etc.
16	menu_gestion_participantes() <i>Se ingresa opcion: 4 “Eliminar Participante” Se ingresa un Id que <b>no existe</b></i>	El modulo debe verificar que exista el id, caso contrario mostrar un mensaje de que no existe participante con ese id	El modulo verifica que existe el id, y muestra un mensaje de que no existe participante con ese id
17	menu_gestion_participantes()	El modulo debe verificar que exista el id, y eliminarlo de la lista, al finalizar imprimir el	El modulo verifica que exista el id, imprime al participante

	<i>Se ingresa opción: 4 “Eliminar Participante” Se ingresa un Id que <b>existe</b></i>	participante eliminado y actualizar el ocupado	que se eliminará, lo elimina y actualiza el ocupado.
<b>Resultados obtenidos</b>			
<b>Resultado (Ap/Desap): Aprobado</b>			
<b>Observaciones: -</b>			

**Caso de Prueba: IT-01**

01:

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 9
Opcion invalida.
```

Ingresamos una opción que no contempla el menú

Ingresamos opción 1, lo cual ingresa al módulo de gestión de Participantes

02:

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 1
```

```
-----
* 1. Gestión de Participantes *
-----
*
* 1. Registro de participantes
* 2. Consultar participante
* 3. Modificar participante
* 4. Eliminar participante
* 5. Listar participantes
* 6. Volver a Menú Principal
*
-----
Seleccione una opcion: |
```

03:

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 7
Opcion invalida.
Presione una tecla para continuar . . .
```

Ingresamos una opción que no contempla el menú

04:

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 1
```

```
C:\Program Files (x86)\Zinjal\  X  +  v
--Registrar Participante--
- Seleccione tipo de registro -
1) Automatico - 2) Manual:
```

Ingresamos opción 1, lo cual ingresa a la sección de Registrar Participante Manual o Auto.

05:

```
--Registrar Participante--
- Seleccione tipo de registro -
1) Automatico - 2) Manual: 3
Ingresó tipo registro inválido: 3
Presione una tecla para continuar . . .
```

Ingresamos tipo de registro inválido

06:

```
--Registrar Participante--
- Seleccione tipo de registro -
1) Automatico - 2) Manual: 1
Ingrese cantidad de registros a generar (entre 30 y 1000): 20
Valor fuera del rango...
Ingrese cantidad de registros a generar (entre 30 y 1000):
```

Ingresamos op. 1, y 20 registros, lo cual es invalido

07-08-09:

```
--Registrar Participante--
- Seleccione tipo de registro -
1) Automatico - 2) Manual: 1
Ingrese cantidad de registros a generar (entre 30 y 1000): 200
Se generaran: 200 participantes automaticamente...

Se cargaron: 30 participantes de AFRICA
Se cargaron: 53 participantes de AMERICA
Se cargaron: 45 participantes de ASIA
Se cargaron: 39 participantes de EUROPA
Se cargaron: 33 participantes de OCEANIA
```

Ingresamos 200 registros, se validan y se cargan

10-11

Ahora ingresamos a registro Manual

```
-----
* 1. Gestión de Participantes *
-----
* 1. Registro de participantes *
* 2. Consultar participante *
* 3. Modificar participante *
* 4. Eliminar participante *
* 5. Listar participantes *
* 6. Volver a Menú Principal *
*
-----
Seleccione una opción: 1|
```

```
C:\Users\MI PC\Desktop\toda X + v
--Registrar Participante--
- Seleccione tipo de registro -
1) Automatico - 2) Manual: 2|
```

12:

```
Ingrese Apellido: Alfaro
Ingrese Primer Nombre: Josue
Ingrese Segundo Nombre: Dario
Ingrese Edad (años): 26
Ingrese Altura (mts): 1.70
Ingrese Cant. Idiomas que maneja: 2
Ingrese Continente: 1
El continente no es válido: 1

--Continentes válidos: --
AFRICA , AMERICA , ASIA , EUROPA , OCEANIA ,
-----

Presione una tecla para continuar . . .
Ingrese Continente: ASIA
Ingrese Cant. de Lenguajes de programacion que conoce: 1
Ingrese Años de Experiencia que posee: 2
Ingrese Cant. de Proyectos en ejecución: 1
Ingrese Cant. de Proyectos finalizados: 2
Ingrese Cant. de Proyectos liderados: 1
Desea seguir ingresando participantes? s/n:
```

Procedemos a llenar todos los campos con datos válidos, caso contrario el sistema nos indicará que hubo error, al finalizar nos pregunta si queremos seguir agregando (s/n)

13:

```
Se cargaron: 30 participantes de AFRICA
Se cargaron: 53 participantes de AMERICA
Se cargaron: 46 participantes de ASIA
Se cargaron: 39 participantes de EUROPA
Se cargaron: 33 participantes de OCEANIA

Presione una tecla para continuar . . .
```

Se incrementaron los participantes generados automáticamente con el participante manual (ASIA suma 1)

14 – 15:

```

-----
*   1. Gestión de Participantes   *
-----
*                               *
*  1. Registro de participantes   *
*  2. Consultar participante     *
*  3. Modificar participante     *
*  4. Eliminar participante      *
*  5. Listar participantes       *
*  6. Volver a Menú Principal    *
*                               *
-----
Seleccione una opcion: 5

```

Listamos los 201  
participantes, el creado  
manual y los 200 creados  
automáticos

```

-----
ID: 1199
Apellido: Reinaga
Primer Nombre: Soledad
Segundo Nombre: Leonarda
Edad: 22 años
Altura: 1.86 mts.
Cantidad de Idiomas: 3
Continente: OCEANIA
Cantidad de Lenguajes de Programación: 10
Años de Experiencia: 1 años
Proyectos Finalizados: 21
Proyectos En ejecución: 2
Proyectos Liderados: 12
-----
ID: 1200
Apellido: Alfaro
Primer Nombre: Josue
Segundo Nombre: Dario
Edad: 26 años
Altura: 1.7 mts.
Cantidad de Idiomas: 2
Continente: ASIA
Cantidad de Lenguajes de Programación: 1
Años de Experiencia: 2 años
Proyectos Finalizados: 2
Proyectos En ejecución: 1

```

16:

Ingresamos a la opción de  
eliminar: 4, e intentamos  
eliminar un participante  
que no existe

```

-----
*   1. Gestión de Participantes   *
-----
*                               *
*  1. Registro de participantes   *
*  2. Consultar participante     *
*  3. Modificar participante     *
*  4. Eliminar participante      *
*  5. Listar participantes       *
*  6. Volver a Menú Principal    *
*                               *
-----
Seleccione una opcion: 4

```

```

--Eliminar Participante--
Ingrese ID de participante a eliminar: 1244

El participante con ID: 1244 no existe....

Presione una tecla para continuar . . .

```

17:

```
--Eliminar Participante--
Ingrese ID de participante a eliminar: 1200

Se eliminó el participante:

-----
ID: 1200
Apellido: Alfaro
Primer Nombre: Josue
Segundo Nombre: Dario
Edad: 26 años
Altura: 1.7 mts.
Cantidad de Idiomas: 2
Continente: ASIA
Cantidad de Lenguajes de Programación: 1
Años de Experiencia: 2 años
Proyectos Finalizados: 2
Proyectos En ejecución: 1
Proyectos Liderados: 3
-----
Presione una tecla para continuar . . .
```

Eliminamos el  
participante id 1200, se  
muestra el que se eliminó

Nuevamente listamos, y  
como se observa, el  
eliminado ya no aparece.

```
-----
ID: 1198
Apellido: Willson
Primer Nombre: Cristal
Segundo Nombre: Isabel
Edad: 29 años
Altura: 1.61 mts.
Cantidad de Idiomas: 2
Continente: AMERICA
Cantidad de Lenguajes de Programación: 4
Años de Experiencia: 20 años
Proyectos Finalizados: 9
Proyectos En ejecución: 0
Proyectos Liderados: 2
-----
ID: 1199
Apellido: Reinaga
Primer Nombre: Soledad
Segundo Nombre: Leonarda
Edad: 22 años
Altura: 1.86 mts.
Cantidad de Idiomas: 3
Continente: OCEANIA
Cantidad de Lenguajes de Programación: 10
Años de Experiencia: 1 años
Proyectos Finalizados: 21
Proyectos En ejecución: 2
Proyectos Liderados: 12
-----
Presione una tecla para continuar . . .
```

## Pruebas de aceptación

**ID Caso de Prueba: AC-01**

**Requisitos de la prueba:** El vector de continentes debe estar cargado previamente, el ocup de continentes debe estar en valor mayor en -1.

**Propósito de la prueba:** Realizar una prueba general del “Sistema De Concurso” desde el registro de los participantes hasta la coronación, indicando los mensajes que validan que se cumpla lo mínimo requerido en cada etapa (mínimo de integrantes, jurados, ejecución de las etapas, etc.)

**Descripción de las acciones y/o condiciones para las Pruebas**

Nº	Acciones (entradas)	Salida Esperada	Salida Obtenida
01	main() administracion_participantes() generar_participantes_automaticos() control_continentes()	El sistema debe generar 200 participantes automaticos y al finalizar indicar cuantos participantes se generaron por continente	El sistema genera 200 participantes automaticos y al finalizar indica cuantos participantes se generaron por continente
02	main() administracion_jurados() Agregar_Jurados() Lista_recursoivo()	El Sistema debe permitir cargar jurados y validar que el jurado no exista (no esté duplicado), al finalizar listar los jurados generados	El sistema agregó los jurados al vector de jurados y permite visualizar los datos de los jurados cargados
03	main() administracion_etapa_1() clasificar_participantes() pertenece_continente() Ordenacion_Continentes() Insercion_primera() mostrar_Participante_etapa1()	El sistema en esta face debe permitir puntuar a todos los participantes registrados, obtener los mejores de cada continente y al finalizar guardar 20 mejores totales y mostrarlos	El sistema clasifica los participantes por sus habilidades, y al finalizar obtiene los 20 mejores y los muestra
04	main() administracion_etapa_2() Evaluacion_2da_ETAPA() Insercion_segunda() mostrar_Participante_etapa2()	El sistema debe clasificar los 20 obtenidos en la primera, los ordena y muestra en forma decreciente	El sistema puntua las participantes de la primera por distintos criterios, los ordena y muestra en forma decreciente
05	main() administracion_coronacion() evaluacion_Coronacion() guardar_finalistas() mostrar_finalistas()	Una vez ejecutado etapas 1 y 2, el sistema permite coronar a la ganadora, sumando las puntuaciones obtenidas y eligiendo a la ganadora, al finalizar muestra los resultados	El sistema muestra el resultado del certamen, indicando la ganadora por el puntaje más alto, y las 4 finalistas.

**Resultados obtenidos**

**Resultado (Ap/Desap):** Aprobado

**Observaciones:** -

**Caso de Prueba: AC-01****01 PARTICIPANTES:**

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 1|
```

```
-----
* 1. Gestión de Participantes *
-----
*
* 1. Registro de participantes
* 2. Consultar participante
* 3. Modificar participante
* 4. Eliminar participante
* 5. Listar participantes
* 6. Volver a Menú Principal
*
-----
Seleccione una opcion: 1
```

```
--Registrar Participante--
- Seleccione tipo de registro -
1) Automatico - 2) Manual: 1
Ingrese cantidad de registros a generar (entre 30 y 1000): 200

Se generaran: 200 participantes automaticamente...

Se cargaron: 32 participantes de AFRICA
Se cargaron: 44 participantes de AMERICA
Se cargaron: 30 participantes de ASIA
Se cargaron: 47 participantes de EUROPA
Se cargaron: 47 participantes de OCEANIA

Presione una tecla para continuar . . .
```

Se probó la gestión  
de participantes el  
modulo completo



## 02 JURADOS:

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes          *
* 2. Gestión de Jurados              *
* 3. Primera Etapa                   *
* 4. Segunda Etapa                   *
* 5. Coronación                      *
* 6. Salir                           *
*
*****
Seleccione una opcion: 2
```

```
--Agregar Jurado--
Ingrese apellido: JORGE
Ingrese nombre: ROJAS
Ingrese empresa: VMODAS
Ingrese cargo: GG
Ingresó cargo inválido
Presione una tecla para continuar . . .
Ingrese cargo: GERENTE
Ingrese años en la industria: 1
```

```
---Lista de Jurados---
-----
ID: 8000
Apellido y nombre: JORGE, ROJAS
Empresa: VMODAS
Cargo: GERENTE
Años en la industria: 1
-----
ID: 8001
Apellido y nombre: JORGE, MATOS
Empresa: COLLETI
Cargo: PPTT
Años en la industria: 3
-----
ID: 8002
Apellido y nombre: JUAN, PEREZ
Empresa: GSEERF
Cargo: PTE
Años en la industria: 3
-----
Presione una tecla para continuar . . .
```

Se probó la gestión  
de jurados el  
modulo completo

## 03 PRIMERA ETAPA:

```
*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 3|
```

```
-----
* 3. Primera Etapa *
-----
*
* 1. Puntuar las 20 mejores
* 2. Consultar un puntaje
* 3. Volver a Menú Principal
*
-----
Seleccione una opcion: 1
```

```
--Clasificacion 1er Etapa--

---* Se guardaron las: 20 mejores del certamen!... *---
---
Continente: EUROPA
ID Participante: 1196
Participante: Velazques, Abril, Aba
Suma de notas de vestuario: 78.3333
Suma de notas de elegancia: 81.3333
Suma de notas de elocuencia: 56
Suma de notas de idiomas: 83.3333

Puntaje total en Etapa 1: 299
---
---
Continente: AMERICA
ID Participante: 1125
Participante: Velazques, Soledad, Amy
Suma de notas de vestuario: 67.6667
Suma de notas de elegancia: 64.3333
Suma de notas de elocuencia: 80.3333
Suma de notas de idiomas: 83.3333

Puntaje total en Etapa 1: 295.667
---
---
Continente: AMERICA
ID Participante: 1036
Participante: Torres, Lizbeth, Defina
Suma de notas de vestuario: 45
Suma de notas de elegancia: 82.6667
Suma de notas de elocuencia: 59.6667
Suma de notas de idiomas: 100

Puntaje total en Etapa 1: 287.333
---
```

## 04 SEGUNDA ETAPA

```

*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes *
* 2. Gestión de Jurados *
* 3. Primera Etapa *
* 4. Segunda Etapa *
* 5. Coronación *
* 6. Salir *
*
*****
Seleccione una opcion: 4

```

```

-----
* 4. Segunda Etapa *
-----
*
* 1. Puntuar las 20 mejores *
* 2. Consultar un puntaje *
* 3. Volver a Menú Principal *
*
-----
Seleccione una opcion: 1

```

```

--Clasificacion 2da Etapa--
---
Continente: AMERICA
ID Participante: 1125
Nombres: Velazques, Soledad, Amy
Suma de lenguajes programacion: 50
Suma de experiencia: 100
Suma de proyectos finalizados: 75
Suma de proyectos en ejecucion: 100
Suma de proyectos liderados: 72.2222

Puntaje Total en Etapa 2: 397.222
---
---
Continente: AMERICA
ID Participante: 1035
Nombres: Torres, Alexis, Mirtha
Suma de lenguajes programacion: 100
Suma de experiencia: 100
Suma de proyectos finalizados: 100
Suma de proyectos en ejecucion: 50
Suma de proyectos liderados: 40.7407

Puntaje Total en Etapa 2: 390.741
---
---
Continente: OCEANIA
ID Participante: 1183
Nombres: Calvo, Merrye, Amy
Suma de lenguajes programacion: 100
Suma de experiencia: 100
Suma de proyectos finalizados: 75
Suma de proyectos en ejecucion: 50
Suma de proyectos liderados: 55.5556

Puntaje Total en Etapa 2: 380.556
---

```

## 05 CORONACION

```

*****
* SISTEMA DE CONCURSO MISS PROGRAMACION 2024 *
*****
*
* 1. Gestión de participantes
* 2. Gestión de Jurados
* 3. Primera Etapa
* 4. Segunda Etapa
* 5. Coronación
* 6. Salir
*
*****
Seleccione una opcion: 5

```

```

-----
* 5. ¡ CORONACION !
-----
*
* 1. Realizar la coronacion
* 2. Mostrar Finalistas
* 3. Volver a Menú Principal
*
*
-----
Seleccione una opcion: 1

```

```

*****
* LA GANADORA ES... *
Continente: AMERICA
ID Participante: 1125
Nombres: Velazques, Soledad, Amy
Puntaje total primer etapa: 295.667
Puntaje total segunda etapa: 397.222

Puntaje Total Coronacion: 363.709
*****

*****
* LAS FINALISTAS SON... *
1) LUGAR
Continente: EUROPA
ID Participante: 1196
Nombres: Velazques, Abril, Aba
Puntaje total primer etapa: 299
Puntaje total segunda etapa: 375

Puntaje Total Coronacion: 349.92
*****

2) LUGAR
Continente: AMERICA
ID Participante: 1035
Nombres: Torres, Alexis, Mirtha
Puntaje total primer etapa: 261.333
Puntaje total segunda etapa: 390.741

Puntaje Total Coronacion: 348.036
*****

```

¡Se obtuvo la ganadora del Certamen!! Y las finalistas

## Conclusiones

Este informe logra englobar todos los contenidos vistos en la materia satisfactoriamente, se presentaron las características de los módulos más importantes y relevantes. Más allá del uso de todos los temas de la programación estructurada nos permite resolver problemas desde el más chico hasta el más grande.

El uso de vectores y registros nos permiten guardar alta cantidad de información valiosa para luego modificarla o usarla para lo que sea necesario, la implementación de la modularización para lograr un programa más eficiente en cuestiones de cálculos y validaciones.

- **Mejoras que podrían incluirse en el programa considerando todo lo aprendido.**

El sistema se en el ingreso de apellido de participantes, lo que sucede cuando el apellido ingresado ya existe es que 1ero se cargan todos los campos del registro y después sale por pantalla que ya existe, lo que se puede hacer es si el apellido ya existe que no deje seguir cargando los demás campos hasta que el apellido ingresado sea distinto a los ya existentes, en nuestro programa si bien cumple que no se repita el apellido repetido, es algo que se puede mejorar a futuro.

Otro tema a agregar en las fases evaluativas es poder consultar el nombre del jurado que voto a tal participante.

- **Dificultades que presentó el desarrollo y cómo se resolvieron estas situaciones.**

Unas de las dificultades que tuvimos fueron como sacar las mejores 4 de cada continente para obtener las 20 mejores. Lo que hicimos fue 1ero crear un módulo que filtre por continente y agregar a un vector de continentes, entonces creamos un vector para cada continente. Una vez que las participantes estén en su respectivo continente, lo que planteamos fue ordenar de acuerdo al puntaje a esos vectores de forma decreciente para así poder obtener las 4 mejores de cada continente y de esta forma logramos obtener las 4 mejores.

Otra dificultad encontrada fue al momento de plantear la gestión automática de participantes donde encontramos muchos desafíos a tener en cuenta, la cantidad de participantes existentes, la cantidad que ingresa no debía ser mayor a la capacidad máxima del vector, y la búsqueda de apellido y nombres no lograba funcionar correctamente por que el vector debía estar ordenado por 3 criterios, entonces basándose en los algoritmos de la materia se planteó un método de ordenación por estos 3 criterios (primer, segundo nombres y apellidos), para hacer una búsqueda secuencial eficiente que termine si el participante no se encuentra o al momento de superar el apellido buscado con el comparado.