

# Estrutura de Dados 1

## Alunos:

Gabriel Soares de Barros Villarinho

Marcelo Lima Bromonschenkel

## TUTORIAL

- 1 - linux/MAC:
  - 1.1 - instale o gcc com:

```
$ sudo apt update -y
$ sudo apt upgrade -y
$ sudo apt install gcc
```
  - 1.2 - Entre na pasta do arquivo main.cpp pelo terminal
  - 1.3 - Execute o seguinte comando para compilar o arquivo

```
$ g++ -o main main.cpp
```
  - 1.4 - Para executar o programa escreva o seguinte comando:

```
$ ./main
```
- 2 - Windows
  - 2.1 Instale o mingw na sua máquina, pode seguir esse tutorial:  
<https://www.youtube.com/watch?v=1Uw9EV4Te8M&t=153s>
  - 2.2 Com a pasta aberta no terminal execute o seguinte comando para compilar o arquivo

```
g++ main.cpp
```
  - 2.3 Para executar o programa digite o seguinte comando no terminal

```
a.exe
```

**IPC:** Verifique se a última linha do csv está vazia, é necessário que esteja

## 1- Print do menu inicial e seleção de opções

```
AX 100
TAM 100
Adc 10 Agência Espacial Brasileira
Alt =====
Rem [1] Adicionar um novo planeta
Bus [2] Alterar dados de um planeta
Mos [3] Remover um planeta
Mos [4] Buscar um planeta "planeta",
Sai [5] Mostrar dados de um planeta
Sai [6] Mostrar planetas
    [7] Sair do sistema
> 
```

```
=====
Agência Espacial Brasileira " << endl;
===== " << endl;

()

for (int i = 0; i < TAM_OPTIONS; i++)
    << options[i] << endl;

return (int *value)
```

## 2 - Funções responsáveis para adicionar planetas

### 2.1 - Para adicionar na lista duplamente encadeada

```
1  template <typename T>
2  void Append(list<T> &lst, T newData)
3  {
4      node<T> *newNode = new node<T>;
5      newNode->data = newData;
6
7      if (lst.count == 0)
8          lst.begin = lst.end = newNode;
9      else
10     {
11         newNode->previous = lst.end;
12         lst.end->next = newNode;
13         lst.end = newNode;
14     }
15     lst.count++;
16 }
```

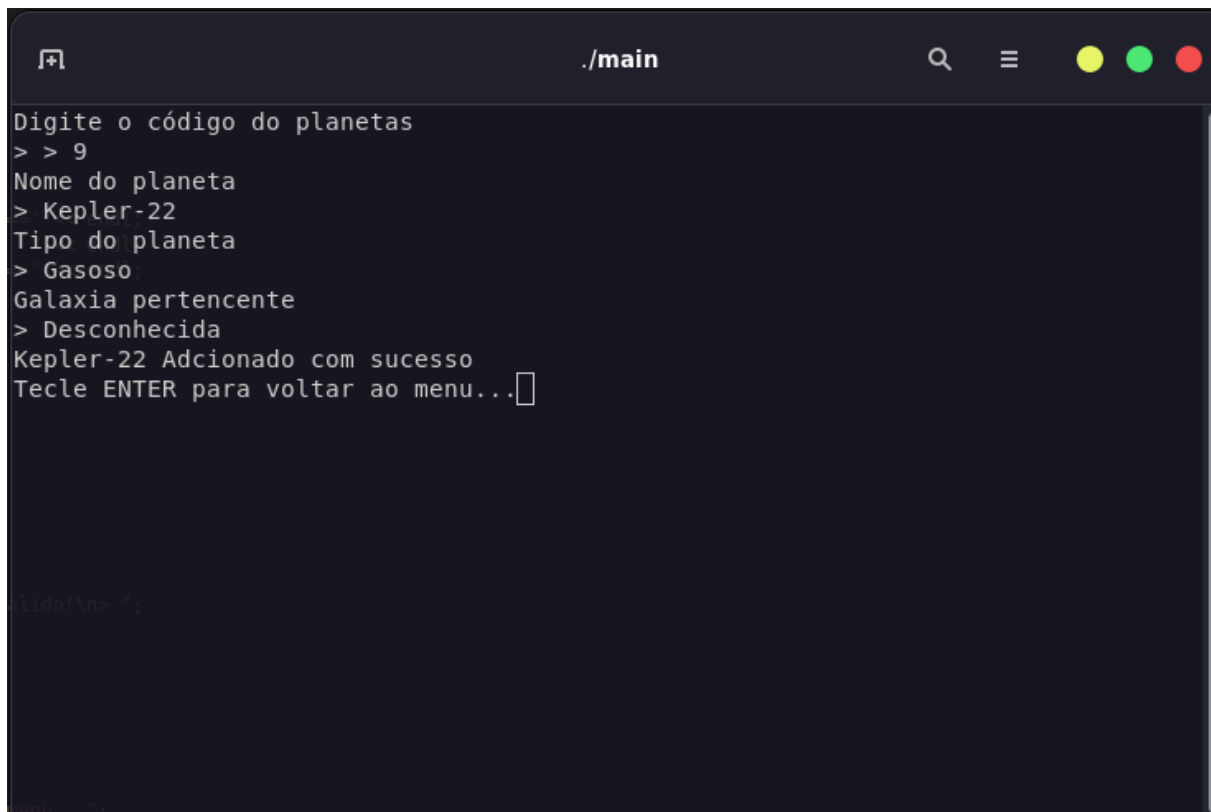
## 2.2 - Para ler dados do arquivo csv

```
1 void readPlanets(list<Planet> &lst)
2 {
3     FILE *fptr;
4
5     fptr = fopen(LOCAL, "r");
6
7     if (fptr == NULL)
8     {
9         cout << "Não foi possível abrir o arquivo" << endl;
10        fclose(fptr);
11        return;
12    }
13
14    char str[TAM];
15    while (fgets(str, TAM, fptr))
16    {
17        Planet atual;
18        char temp[TAM];
19
20        // Copiando o id
21        memset(temp, 0, TAM);
22        int pos = searchChar(';', 0, str);
23        strncpy(temp, str, pos);
24        atual.Code = atoi(temp);
25
26        // Copiando o nome
27        memset(temp, 0, TAM);
28        int pos_old = ++pos;
29        pos = searchChar(';', pos, str);
30        strncpy(temp, str + pos_old, pos - pos_old);
31        strcpy(atual.Name, temp);
32
33        // Copiando o tipo
34        memset(temp, 0, TAM);
35        pos_old = ++pos;
36        pos = searchChar(';', pos, str);
37        strncpy(temp, str + pos_old, pos - pos_old);
38        strcpy(atual.Type, temp);
39
40        // Copiando a galaxia
41        memset(temp, 0, TAM);
42        strcpy(temp, str + pos + 1);
43        strcpy(atual.Galaxy, temp);
44
45        Append(lst, atual);
46    }
47
48    fclose(fptr);
49 }
```

## 2.3 - Para adicionar um planeta novo durante a execução

```
1 // Para o arquivo main.cpp
2 void readNewPlanet(Planet *newPlanet)
3 {
4     char temp[TAM];
5     cout << "Digite o código do planetas\n> ";
6     _readInteger(&newPlanet->Code);
7     cout << "Nome do planeta\n> ";
8     cin.ignore();
9     fgets(temp, TAM, stdin);
10    temp[strcspn(temp, "\n")] = 0;
11    strcpy(newPlanet->Name, temp);
12    cout << "Tipo do planeta\n> ";
13    cin >> temp;
14    strcpy(newPlanet->Type, temp);
15    cout << "Galaxia pertencente\n> ";
16    cin.ignore();
17    fgets(temp, TAM, stdin);
18    strcpy(newPlanet->Galaxy, temp);
19 }
```

## 2.4 execução para adicionar um novo planeta



```
./main
Digite o código do planetas
> > 9
Nome do planeta
> Kepler-22
Tipo do planeta
> Gasoso
Galaxia pertencente
> Desconhecida
Kepler-22 Adicionado com sucesso
Tecle ENTER para voltar ao menu...

```

### 3 - Funções para alteração de dados de um planeta

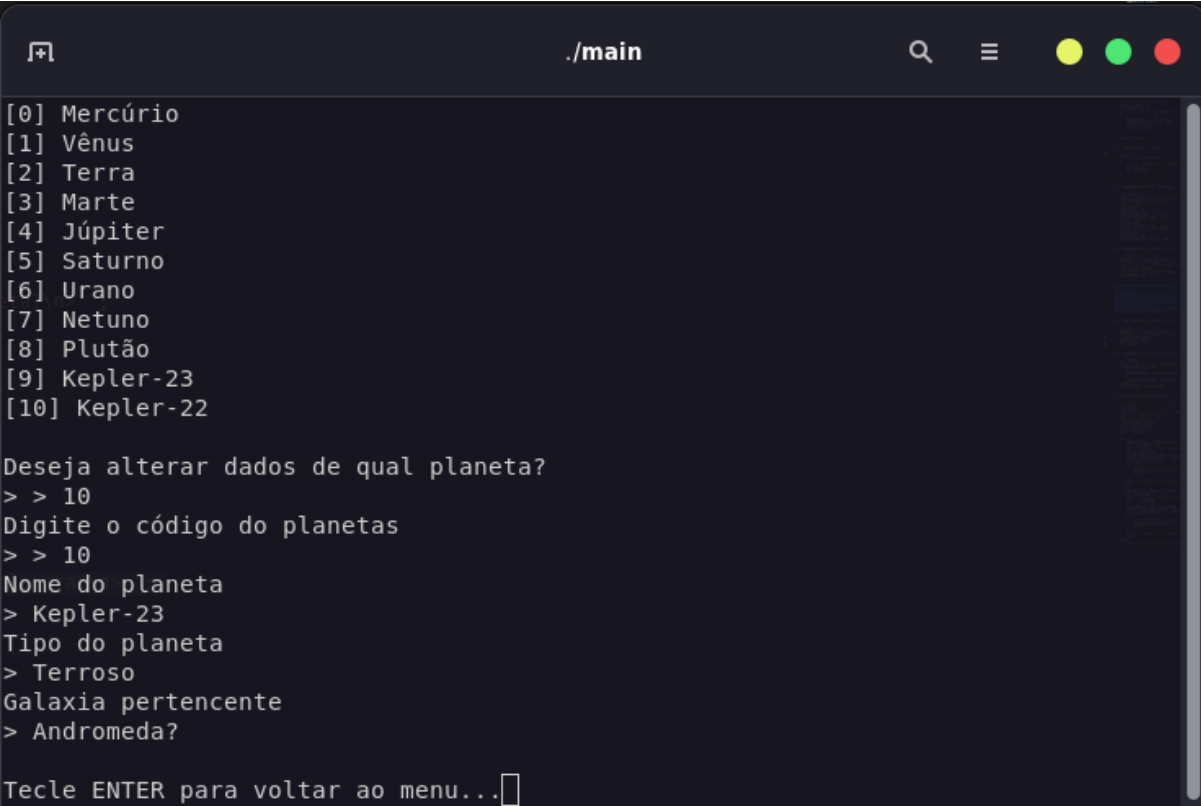
#### 3.1 pegar um planeta na lista duplamente encadeada

```
1  template <typename T>
2  node<T> *get(list<T> lst, u_int32_t position)
3  {
4      try
5      {
6          if (position > lst.count - 1)
7              throw "Posição inválida";
8      }
9      catch (string e)
10     {
11         cout << e << endl;
12     }
13
14     node<T> *aux = lst.begin();
15     for (int i = 0; i < position; i++)
16         aux = aux->next;
17
18     return aux;
19 }
20
```

#### 3.2 Interação com o usuário

```
1  void changePlanet(list<Planet> &lst)
2  {
3      int temp;
4      quickSort(lst.begin, lst.end);
5      PrintList(lst, PrintRemove, true);
6      cout << "Deseja alterar dados de qual planeta?\n> ";
7      _readInteger(&temp);
8      node<Planet> *aux = get(lst, temp);
9      readNewPlanet(&(aux->data));
10 }
```

### 3.3 Execução da função



```
./main
[0] Mercúrio
[1] Vênus
[2] Terra
[3] Marte
[4] Júpiter
[5] Saturno
[6] Urano
[7] Netuno
[8] Plutão
[9] Kepler-23
[10] Kepler-22

Deseja alterar dados de qual planeta?
> > 10
Digite o código do planetas
> > 10
Nome do planeta
> Kepler-23
Tipo do planeta
> Terroso
Galaxia pertencente
> Andromeda?

Tecle ENTER para voltar ao menu...
```

## 4 - Funções responsáveis para a remoção de uma planeta

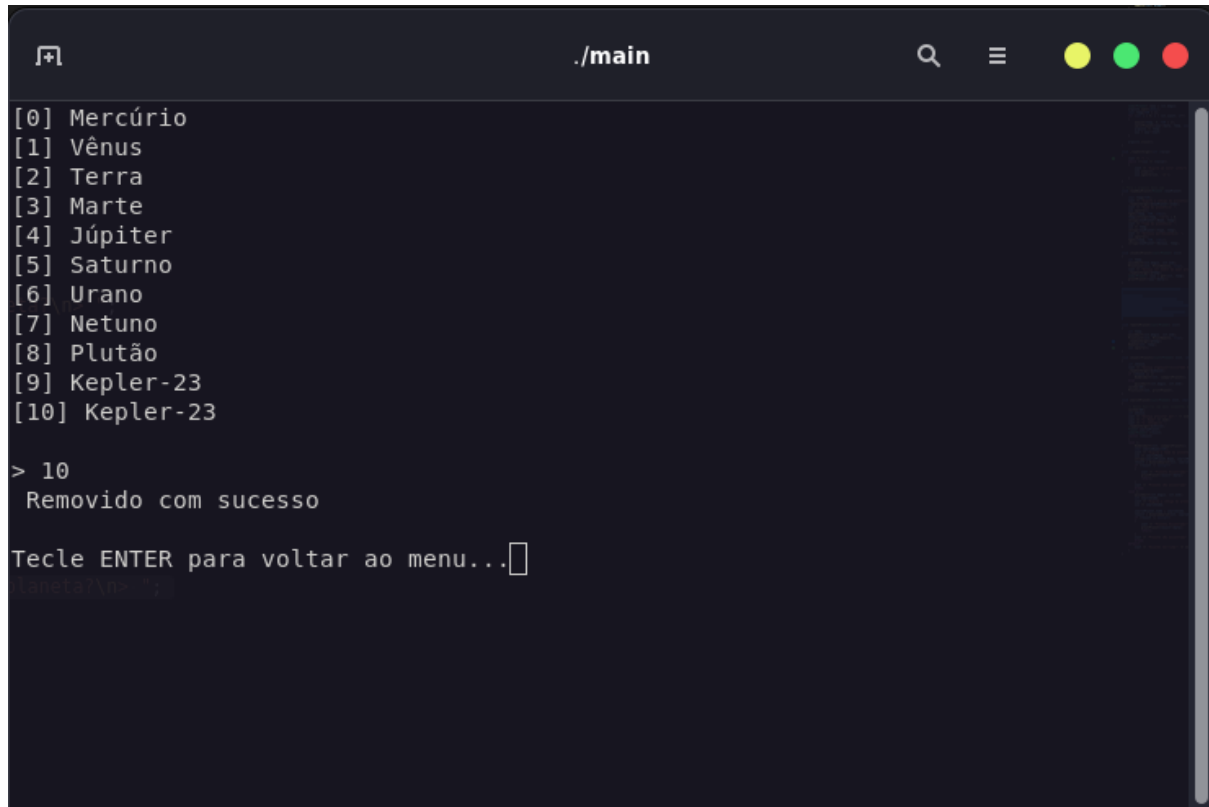
### 4.1 para remover da lista duplamente encadeada

```
1  template <typename T>
2  node<T> *get(list<T> lst, u_int32_t position)
3  {
4      try
5      {
6          if (position > lst.count - 1)
7              throw "Posição inválida";
8      }
9      catch (string e)
10     {
11         cout << e << endl;
12     }
13
14     node<T> *aux = lst.begin;
15     for (int i = 0; i < position; i++)
16         aux = aux->next;
17
18     return aux;
19 }
20
21 template <typename T>
22 void Remove(list<T> &lst, u_int32_t position)
23 {
24
25     if (position >= lst.count)
26     {
27         cout << "Posição inválida" << endl;
28         return;
29     }
30
31     if (lst.count == 0)
32     {
33         cout << "A lista está vazia" << endl;
34         return;
35     }
36
37     node<T> *aux = get(lst, position);
38     (aux->next != nullptr) ? aux->next->previous = aux->previous : lst.end = aux->previous;
39     (aux->previous != nullptr) ? aux->previous->next = aux->next : lst.begin = aux->next;
40     delete aux;
41     lst.count--;
42 }
```

### 4.2 Interação com o usuário

```
1  void removePlanet(list<Planet> &lst)
2  {
3      int temp;
4      quickSort(lst.begin, lst.end);
5      PrintList(lst, PrintRemove, true);
6      _readInteger(&temp);
7      Remove(lst, temp);
8      cin.ignore();
9  }
```

### 4.3 Execução da função



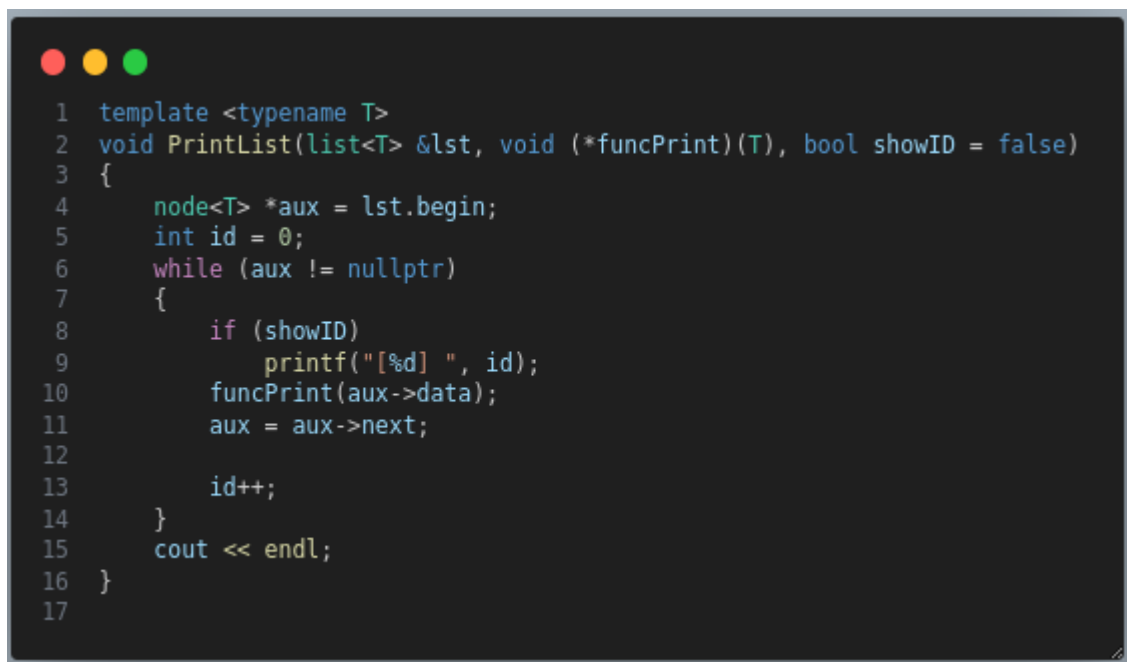
```
./main
[0] Mercúrio
[1] Vênus
[2] Terra
[3] Marte
[4] Júpiter
[5] Saturno
[6] Urano
[7] Netuno
[8] Plutão
[9] Kepler-23
[10] Kepler-23

> 10
Removido com sucesso

Tecle ENTER para voltar ao menu...
[anexo2\c:\c++\bin\g++.exe]
```

## 5 - Funções Responsáveis por mostrar todas as entidades contidas na lista duplamente encadeada

### 5.1 Função para mostrar



```
1 template <typename T>
2 void PrintList(list<T> &lst, void (*funcPrint)(T), bool showID = false)
3 {
4     node<T> *aux = lst.begin();
5     int id = 0;
6     while (aux != nullptr)
7     {
8         if (showID)
9             printf("[%d] ", id);
10        funcPrint(aux->data);
11        aux = aux->next;
12
13        id++;
14    }
15    cout << endl;
16 }
17
```



```

1 void printPlanet(const Planet planet)
2 {
3     cout << "
4     cout << " |
5     cout << " Código: " << planet.Code << endl;
6     cout << " Nome: " << planet.Name << endl;
7     cout << " Tipo: " << planet.Type << endl;
8     cout << " Galaxia pertencente: " << planet.Galaxy;
9     cout << " |
10 }

```

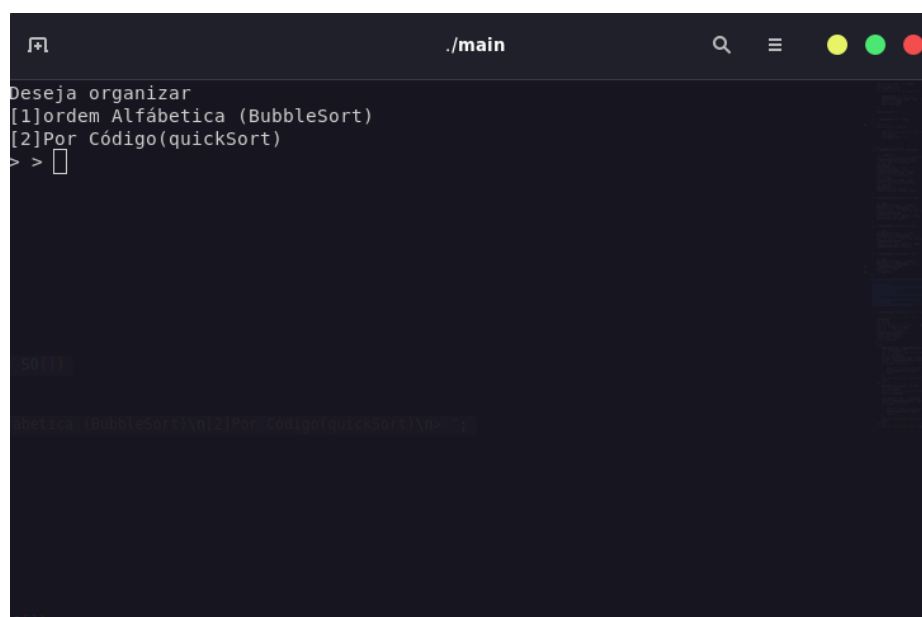
## 5.2 - Interação com o usuário

```

1 void showAllPlanets(list<Planet> &lst, char S0[])
2 {
3     int choose;
4     cout << "Deseja organizar\n[1]ordem Alfabética (BubbleSort)\n[2]Por Código(quickSort)\n";
5     _readInteger(&choose);
6     if (choose == 1)
7         BubbleSort(lst, comparePlanets);
8     else
9         quickSort(lst.begin, lst.end);
10    system(S0);
11    PrintList(lst, printPlanet);
12    cin.ignore();
13 }

```

## 5.3 execução da função



```

./main
Deseja organizar
[1]ordem Alfabética (BubbleSort)
[2]Por Código(quickSort)
> >

```

```

Código: 5
Nome: Júpiter
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 10
Nome: Kepler-23
Tipo: Gasoso
Galaxia pertencente: Desconhecida

Código: 4
Nome: Marte
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 1
Nome: Mercúrio
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 8
Nome: Netuno
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 9
Nome: Plutão
Tipo: Anão
Galaxia pertencente: Via Láctea

Código: 6
Nome: Saturno
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 3
Nome: Terra
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 7
Nome: Urano
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 2
Nome: Vênus
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Tecle ENTER para voltar ao menu...

```

```

Código: 1
Nome: Mercúrio
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 2
Nome: Vênus
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 3
Nome: Terra
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 4
Nome: Marte
Tipo: Terrestre
Galaxia pertencente: Via Láctea

Código: 5
Nome: Júpiter
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 6
Nome: Saturno
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 7
Nome: Urano
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 8
Nome: Netuno
Tipo: Gasoso
Galaxia pertencente: Via Láctea

Código: 9
Nome: Plutão
Tipo: Anão
Galaxia pertencente: Via Láctea

Código: 10
Nome: Kepler-23
Tipo: Gasoso
Galaxia pertencente: Desconhecida

Tecle ENTER para voltar ao menu...

```

## 6 - Mostrar uma entidade

```

1 void showOnePlanet(list<Planet> &lst)
2 {
3     int temp;
4     quickSort(lst.begin, lst.end);
5     PrintList(lst, PrintRemove, true);
6     cout << "Deseja ver dados de qual planeta?\n> ";
7     _readInteger(&temp);
8     node<Planet> *aux = get(lst, temp);
9     printPlanet((aux->data));
10    cin.ignore();
11 }
12

```

## 6.1 Execução

```
./main
[0] Mercúrio
[1] Vênus
[2] Terra
[3] Marte
[4] Júpiter
[5] Saturno
[6] Urano
[7] Netuno
[8] Plutão
[9] Kepler-23

Deseja ver dados de qual planeta?
> > 9

|
| Código: 10
| Nome: Kepler-23
| Tipo: Gasoso
| Galaxia pertencente: Desconhecida
|

Tecle ENTER para voltar ao menu...|
```

## 7 - Ordenação Por bubbleSort (alfabética)

```
1 // Funções para implementar o bubble sort
2 template <typename T>
3 void swap(list<T> &lst, int positionA, int positionB)
4 {
5     node<T> *node1 = get(lst, positionA);
6     node<T> *node2 = get(lst, positionB);
7     T aux = node1->data;
8     node1->data = node2->data;
9     node2->data = aux;
10 }
11
12 template <typename T>
13 void BubbleSort(list<T> &lst, int (*funcComp)(T, T))
14 {
15     bool isOrder = false;
16     while (!isOrder)
17     {
18         isOrder = true;
19         for (int i = 1; i < lst.count; i++)
20         {
21             node<T> *nodeA = get(lst, i - 1);
22             node<T> *nodeB = get(lst, i);
23             if (funcComp(nodeA->data, nodeB->data) > 0)
24             {
25                 isOrder = false;
26                 swap(lst, i, i - 1);
27             }
28         }
29     }
30 }
```

```

1 int comparePlanets(const Planet p1, const Planet p2)
2 {
3     return strcasecmp(p1.Name, p2.Name);
4 }

```

## 8- Ordenação por QuickSort (Por Código)

```

1 // Funções responsáveis pelo quick sort
2 template <typename T>
3 void swapNode(node<T> *nodeA, node<T> *nodeB)
4 {
5     T aux = nodeA->data;
6     nodeA->data = nodeB->data;
7     nodeB->data = aux;
8 }
9
10 template <typename T>
11 node<T> *partition(node<T> *start, node<T> *end)
12 {
13     /*
14     Função partition é um complemento para a função quickSort, a qual tem a função
15     de escolher um elemento como um pivô, que no meu caso eu fiz escolhendo o último,
16     e durante a execução a função vai particionando a lista para que as entidades menores
17     que o pivô escolhido fiquem a esquerda e os maiores fiquem a direita do mesmo.
18     Esse homem indiano me salvou
19     https://www.youtube.com/watch?v=ms_rjPaUNqs&t=870s
20     */
21     int pivot = end->data.Code;
22     node<T> *aux1 = start->previous;
23
24     for (node<T> *aux2 = start; aux2 != end; aux2 = aux2->next)
25     {
26         if (aux2->data.Code <= pivot)
27         {
28             aux1 = (aux1 == nullptr) ? start : aux1->next;
29             swapNode(aux1, aux2);
30         }
31     }
32     aux1 = (aux1 == nullptr) ? start : aux1->next;
33     swapNode(aux1, end);
34
35     return aux1;
36 }
37
38 template <typename T>
39 void quickSort(node<T> *start, node<T> *end)
40 {
41     if (end != nullptr && start != end && start != end->next)
42     {
43         node<T> *pivot = partition(start, end);
44
45         quickSort(start, pivot->previous);
46         quickSort(pivot->next, end);
47     }
48 }

```

**OBS:** A execução de ambas as funções podem ser visualizadas no item “5” mostrar todas as entidades

## 9 Busca binária

### 9.1 Funções Responsáveis

```
1 // Funções Responsáveis Pela busca binária
2 // OBS Se for pesquisar por nome use o bubble sort para organizar
3 template <typename T>
4 node<T> *binarySearchRecursive(node<T> *start, node<T> *end, const T &searchData, bool byName)
5 {
6     if (start == nullptr || end == nullptr || start == end->next)
7         return nullptr;
8
9     node<T> *mid = start;
10    int cmp;
11
12    if (byName)
13    {
14        cmp = strcasecmp(mid->data.Name, searchData.Name);
15    }
16    else
17        cmp = (mid->data.Code == searchData.Code) ? 0 : (mid->data.Code < searchData.Code) ? -1
18                                                    : 1;
19
20    if (cmp == 0)
21        return mid;
22    else if (cmp < 0)
23        return binarySearchRecursive(mid->next, end, searchData, byName);
24    else
25        return binarySearchRecursive(start, mid->previous, searchData, byName);
26 }
27
28 template <typename T>
29 node<T> *binarySearch(list<T> &lst, const T &searchData, bool byName)
30 {
31     node<T> *start = lst.begin;
32     node<T> *end = lst.end;
33
34     return binarySearchRecursive(start, end, searchData, byName);
35 }
```

```

1 void searchPlanet(list<Planet> &lst, char S0[])
2 {
3     // Busca Binária com dois elementos diferentes
4     system(S0);
5     int choice;
6     cout << "Deseja procurar por:" << endl;
7     cout << "1. Nome" << endl;
8     cout << "2. Código\n> ";
9     _readInterger(&choice);
10    Planet searchPlanet;
11    node<Planet> *result;
12    switch (choice)
13    {
14    case 1:
15        BubbleSort(lst, comparePlanets);
16        char searchName[TAM];
17        cout << "Digite o nome do planeta que deseja procurar no banco\n> ";
18        cin >> searchName;
19        strcpy(searchPlanet.Name, searchName);
20        result = binarySearch(lst, searchPlanet, true);
21        if (result != nullptr)
22        {
23            cout << "Planeta Encontrado!" << endl;
24            printPlanet(result->data);
25            return;
26        }
27        cout << "Planeta não encontrado" << endl;
28        break;
29    case 2:
30        quickSort(lst.begin, lst.end);
31        int searchCode;
32        cout << "Digite o código do planeta: ";
33        cin >> searchCode;
34
35        searchPlanet.Code = searchCode;
36        result = binarySearch(lst, searchPlanet, false);
37        if (result != nullptr)
38        {
39            cout << "Planeta Encontrado!" << endl;
40            printPlanet(result->data);
41            return;
42        }
43        cout << "Planeta não encontrado" << endl;
44        break;
45    default:
46        cout << "Escolha Inválida!" << endl;
47    }
48 }

```

## 9.2 Execução

```
./main
Deseja procurar por:
1. Nome
2. Código
> > 1
Digite o nome do planeta que deseja procurar no banco
> terra
Planeta Encontrado!

| Código: 3 |
| Nome: Terra |
| Tipo: Terrestre |
| Galaxia pertencente: Via Láctea |

Tecle ENTER para voltar ao menu...|
```

```
./main
Deseja procurar por:
1. Nome
2. Código
> > 2
Digite o código do planeta: 5
Planeta Encontrado!

| Código: 5 |
| Nome: Júpiter |
| Tipo: Gasoso |
| Galaxia pertencente: Via Láctea |

Tecle ENTER para voltar ao menu...|
```

## 10 - salvar no arquivo csv

```
1 void writePlanets(list<Planet> &lst)
2 {
3     ofstream arquivo;
4
5     node<Planet> *aux = lst.begin();
6     arquivo.open(LOCAL);
7     char temp[TAM * 2];
8     for (int i = 0; i < lst.count; i++)
9     {
10         memset(temp, 0, TAM * 2);
11         splitPlanets(aux->data, temp, sizeof(temp));
12         arquivo << temp;
13         aux = aux->next;
14     }
15
16     arquivo.close();
17 }
```

```
1 void splitPlanets(Planet &planet, char *result, int resultSize)
2 {
3     snprintf(result, resultSize, "%d;%s;%s;%s", planet.Code, planet.Name, planet.Type, planet.Galaxy);
4 }
```

```
1 1;Mercúrio;Terrestre;Via Láctea
2 2;Vênus;Terrestre;Via Láctea
3 3;Terra;Terrestre;Via Láctea
4 4;Marte;Terrestre;Via Láctea
5 5;Júpiter;Gasoso;Via Láctea
6 6;Saturno;Gasoso;Via Láctea
7 7;Urano;Gasoso;Via Láctea
8 8;Netuno;Gasoso;Via Láctea
9 9;Plutão;Anão;Via Láctea
10 10;Kepler-23;Gasoso;Desconhecida
```