**Problem analysis**

**Problem specification**

|  |  |
| --- | --- |
| **Client:** | * Cali’s mayor’s office |
| **Users:** | * Public officials |
| **Functional requirements:** | * Req#1: Register a community. * Req#2: Register a place. * Req#3: Add a product to a community. * Req#4: Delete a product from a community. * Req#5: Add a species in a place. * Req#6: Modify a species data in a place. * Req#7: Access the information from a place. * Req#8: Access the information of the communities in a department. * Req#9: Access the information of communities based on their biggest problematic. * Req#10: Display the name of the place with the most species diversity. * Req#11: Display the three places with the biggest area. |
| **Context of the problem:** | * The 16th edition of the UN’s Climate Change Conference of the Parties (COP) is an important event to address the conservation of biodiversity and ecosystems. This year, the event is being held in Cali this year and for the reason, the Mayor’s Office is looking to make a program that will let them register, organize, and keep track of the biodiverse places, the communities that take care of those places, and the products the communities sell. * The public officials should be able to utilize the program to save information of these biodiverse places (and their species), communities and products. Additionally, the program should allow them to consult, modify or eliminate information depending on the type of data. Lastly, the program should allow the consultation of the most diverse place and the top three with the biggest area. * The program should be able to work with two separate menus (administrative and consultation). It should be able to be manipulated by multiple users at a time, be organized and be able to handle a big amount of information. |
| **Nonfunctional requirements:** | * Interoperability: The program should be able to handle multiple users at a time. * High usability: The program should use a menu that makes it easy to understand while entering, editing, or consulting information. * High availability: The program should be working 24/7 and outage risks mitigated. |

**Requirement specifications**

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#1: Register a community. | | |
| **Summary:** | The system allows the user to register a new community by adding a name, type, representant’s name, representant’s phone number, population, and greatest challenges. The greatest challenges will be asked one by one, and the user must answer each one with yes or no. At the start, the product inventory is an empty array of 20 elements that will be filled as the program progresses so it’s not an input. The community type options will be displayed so the user can pick one of each. If the community is registered correctly, a message stating that the process was successful will print. On the other hand, if the community that is trying to be registered already exists, there is any error from unknown cause, there is invalid information or incomplete information the appropriate message will be displayed. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| Name | String | * Can’t be repeated from the other communities already registered. * Mandatory |
| intType | int | * Must be one of the following options:   1. Afro Colombian   2. Indigenous   3. Raizal * Mandatory |
| Representant’s Name | String | * Mandatory |
| Representant’s Phone Number | String | * Mandatory |
| Population | Int | * Mandatory |
| Greatest challenges | String | * A yes or no question will be printed for every possible challenge and the user must answer each one that applies in order to register 0 or more challenges to the community. This are the possible option of challenges:   1. Absence of hospitals   2. Lack of schools   3. Inadequate access to clean water   4. Insufficient access to food |
| **Results or postcondition** | The program allows the user to register a new community using the name, type, representants name, representants phone number, population, and greatest challenges. The greatest challenges are selected by answering “yes” or “no” to the 4 possible options. The system will create an array of 20 empty spaces to later fill with the community’s product inventory (data type product[]). In order to register the enumeration like community type the possible options will be printed for the user to choose from. The greatest challenges questions will then be turned into their respective literal position and then into the literals themselves. The system will also check if the name of the entered community already exists inside the product array. Afterwards, a message of success will appear. In case the community is not registered a message of error, incomplete information or invalid information will print depending on the situation. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Success | String | A message stating that the community was registered successfully will be printed. |
| Duplicate Community | String | A message saying that the community that is trying to be added already exists will pop up. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the community to be registered will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#2: Register a place. | | |
| **Summary:** | A user can register a new biodiverse place after entering the name, department, area, type, inauguration date, photo, caregiving community’s name, and economic resources. The present species in the place at the beginning is an empty array with 50 elements that will be filled as the program runs. In order to select the caregiving community, a small menu displaying all the registered communities will be shown so the user enters one option. To enter the department and place type two small menus showing the available options will be printed for the user to choose from. Before registering a place, the system will check if there are duplicates (displaying a “duplicates” message in case there are) and if there is space to register a new place (displaying a “no space” in case there isn’t). If the place is registered correctly, a message stating that the process was successful will pop up. On the other hand, a message of error, incomplete information or invalid information will be printed depending on the cause of the error. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| Name | String | * Mandatory |
| intDepartment | int | * Must be one of the following options:   1. Choco   2. Valle   3. Cauca   4. Nariño * Mandatory |
| Area | Double | * Mandatory |
| intType | int | * Must be one of the following options:   1. Protected area   2. National Park   3. Private area |
| Inauguration date | Date | * dd-mm-aaaa * Mandatory |
| Photo | String | * URL of the selected image * Mandatory |
| intCommunty | int | * Can only be one of the preregistered communities (They will be displayed with numbers by its side). * Mandatory |
| Economic Resources | Double | * Mandatory |
| **Results or postcondition** | The place is registered with all the mandatory information: Name, department, area, type, inauguration date, photo, caregiving community’s name, economic resources. First, the system will search for duplicates, if there are any, a “duplicate message” will be thrown. Then it will look for an available space to save a new place, if there is no space, a no space message will be shown. Afterwards, it will create an empty array with 50 spaces to fill with the species that are present in the area (data type Species[]). The name of the possible registered caregiving communities will be displayed for the user to choose one from in the form of an int (if there are no preregistered communities a message will be printed). Next, two other small menus will be printed showing the available options for the department and place type for the user to enter their choice as an int. If done correctly, a message of success will be printed. If there are any errors, the appropriate messages of error, incomplete or invalid information will de be printed. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Success | String | A message stating that the place was registered successfully will be printed. |
| No preregistered communities | String | A message stating that there are no preregistered communities will be printed |
| Duplicate place | String | A message saying that the place that is trying to be added already exists will pop up. |
| No space for new place | String | A message stating there is no more space to store a new place will pop up. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the place to be registered will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#3: Add a product to a community. | | |
| **Summary:** | A product can be registered by a user to a community if there are registered communities beforehand (throwing an empty communities message if there are none), an int representing a community, product name, natural materials percentage, type (selected from a menu) and handcraft condition are entered completely and correctly. The system will also check that there is space to add a new product and if the product name already exists. To enter a product type, a small menu showing the options for both states will be printed for the user to choose from (Food or craft). After a product is created and associated to a community, it will show a message stating that the product was registered successfully. If any error were to appear, the error message, incomplete information or invalid information will be printed depending on the root of the error. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| intCommunity | int | * The name is extracted from one of the preexisting communities already created (they will be printed with numbers by their side). * Mandatory |
| Product name | String | * Mandatory |
| Natural materials percentage | Double | * Mandatory |
| intType | int | * Must contain one of the following options:   1. Food   2. Craft * Mandatory |
| Handcraft condition | String | * Only options are:   + Yes   + No * Mandatory |
| **Results or postcondition** | To register a new product, it needs to be associated to a community, so if there are no registered communities a “empty communities” message will be printed. The int entered will be changed to a community which then will be associated with the new product. The system will also check if there is no space to save a new product and if the name of the entered new product already exists, printing either a “no space” or “duplicate” message are printed if any is true. To add a product, the user must the name of the product, natural materials percentage, the type, and the handcraft condition. After, If the community exists, there is space in the array, and no product duplicates are present, the user enters all the necessary information, a printed message will say a new product has been added to the community. Otherwise, a message of error, incomplete information, or invalid information will be printed depending on the cause. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Added New Product | String | A message stating that the product was created and then registered successfully to a community will be printed. |
| Empty Communities | String | A message will show that there are no registered communities. |
| No space for new product | String | A message stating that the entire array has been filled and there is no more space to register any other product will be printed. |
| Duplicate product | String | A message saying that the product that is trying to be added already exists will pop up. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the product to be registered will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#4: Delete a product from a community. | | |
| **Summary:** | For a user to eliminate a product from a community he will enter an int from a menu displaying the existing communities, and then select from the next menu a product that was preregistered to the community. If there is no preregistered community or product a “no preregistered community” or “no preregistered product” message will be printed. After the product is eliminated, a success message will pop up. Alternatively, an error, incomplete information or invalid information message will print in case the product couldn’t be eliminated. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| IntCommunity | int | * The int represents a preregistered community in the system and it is selected from a menu. * Mandatory |
| intProduct | int | * The int represents a preregistered product in a community and it is selected from a menu. * Mandatory |
| **Results or postcondition** | After the user inputs the community’s name and the product from two separate menus represented by ints, the system will allow the product to be deleted. If there are no preregistered communities or products within a community two different no preregistered messages will appear. If there is at least one community and one product, then it will allow the user to erase the product from the inventory of the community and a success message will be printed. If there are any errors, an error message, an incomplete information, or an invalid information will be printed. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Success | String | A message stating that the product was successfully eliminated form a community will be printed. |
| No preregistered community | String | A message saying that there are no communities registered will be printed. |
| No preregistered product | String | A message showing that there are no preregistered products in that community will be printed. |
| Error | String | A message stating there was an error of unknown cause that didn't allow product to be deleted from the community will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#5: Add a species in a place. | | |
| **Summary:** | A user can register a species as long as he enters a valid species’ name, type (which will be entered using a small menu), photo, and local population count. If there is not at least one preregistered place, the system will skip the process. The system will also provide a small menu showing the names of the places that have already been registered so the user must choose a number that represents the place where the new species is going to be added to. Additionally, it will check if there is space for a new species and that there are not any duplicates of the species name saved. After the validations a message stating that a new species was added will be printed. On the other hand, if any of the validations go wrong, a message showing that the place doesn’t exist, that there is no more space for a new species or that the species already exists will be printed. If there are any other type of errors a message displaying that an error of unknown cause, incomplete information or invalid information didn’t allow the process to continue normally. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| Place’s name | int | * The number is chosen from a provided list of pre-registered places so the use must select a number that represents the place name he wants to add a species to. * Mandatory |
| Species’ name | String | * Mandatory |
| intType | int | * Must select one of the following options from the menu that is represented using a numbered list:   1. Flora   2. Fauna * Mandatory |
| Photo | String | * URL of the selected image * Mandatory |
| Local population count | Int | * Mandatory |
| **Results or postcondition** | The system will allow the user to add a species to a place if the place exists (selected from a list of preregistered places) and if at least on place has been registered before selecting this option. Otherwise, a “no preregistered place” message will be printed). The system will also check that the species name is not duplicated and there is space to enter a new species (throwing an error message if any of the las prerequisites are not met). The other information to associate a new species is a photo, and the local population count must be added. In the case of the species type, a small menu showing the two possible options will be printed for the user to select from. After the user enters all the necessary information, a message that a new species has been added to a Place. If there was any other source of error, the appropriate message of error, incomplete information or invalid information will be printed depending on the situation. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Added New species | String | A message stating that the species was created and then registered successfully to a place will be printed. |
| No preregistered places | String | A message showing stating that there is not at least one place to register a species to will be shown. |
| No space for new Species | String | A message stating that the entire array has been filled and there is no more space to register any other species will be printed. |
| Duplicate species | String | A message saying that the species that is trying to be added already exists will pop up. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the species to be registered to a place will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#6: Modify a species data in a place. | | |
| **Summary:** | The user will be able to modify the species’ information inside a community by first entering the place’s name and the species name to check if those even exist. The system will first check if there is at least on registered place and a species to modify the information to (showing a “at least one place” or a “at least one species” message if any of those isn’t true). The system will need to check if the new species name is not duplicated with any other registered species’ name before changing the name of a species. If the validation step passes, the information that wants to be changed should be entered as an int(Species name, type, photo, or local population count). To change the species, type a small menu with the two options that will be used to enter the information. If the information is modified, a success message will be printed showing the information that was changed. If the information couldn’t be modified, the system will show a message of error, incomplete information or invalid information depending on what caused the error. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| intPlace | int | * The int represents a place from the list of preexisting places already registered in the system. * Mandatory |
| intSpecies | int | * The int represents a species from the list of preexisting species already registered within the place. * Mandatory |
| Int modification | int | * Should represent one of the following options:   1. Name   2. Photo   3. Type   4. Local population Count * Mandatory |
| New species name | String | * Optional |
| Type | SpeciesType | * Must select one of the following options represented by a number from the list:   1. Flora   2. Fauna * Optional |
| Photo | String | * URL of the selected image * Optional |
| Local population count | Int | * Optional |
| **Results or postcondition** | The system will check if there is at least one place and then if there is at least one species inside the entered place to be able to modify the information of a species. Both the place and the species will be selected via two menus displaying all the preregistered places and species within the selected place. Afterwards, the user should enter what piece of information he wasn’t to modify from a species using the small, printed menu. In case the user wants to change the name of a species, it will be checked if that name isn’t already registered (it will throw another duplicate message). Afterwards, the user must enter the values of the variables he wishes to change (Type, Photo, and/or local population count). The type of species will be selected via a small menu that will show the two available options. Next, the program shows a message of success showing that the information of the species inside the place was modified (stating which part specifically). In other cases, the message of error, incomplete information or invalid information will be printed. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Success | String | A message stating that the information of a species inside a place was modified correctly will be printed (it shows which part of the species was modified). |
| At least one place | String | A message showing that there is no place registered already so there is no species to modify the information to. |
| At least one species | String | A message showing that there is not any species registered to the place to modify the information to. |
| duplicate species | String | A message saying that the new species name that is trying to be modified already exists will pop up. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the information of a species inside a place be modified will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#7: Access the information from a place. | | |
| **Summary:** | For the system to print all the information from a place, the user must do is enter the number from the list (that is printed) that matches the place’s name. There must be registered at least one place to access the function, otherwise, an “at least on place” message will be shown. If the information is not printed, an error message or invalid information message will be printed depending on the source of the error. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| intPlace | int | * The int represents a place from the list of preexisting places already registered in the system. * Mandatory |
| **Results or postcondition** | The system will first make sure that there is at least one preregistered place to make the function work (throwing an error message if there is none). After the user enters the int that represents the name of the place, the system will display all the information that is stored about that place. For the caregiving place and the species it has saved, it will only print the names. If the information is not printed, a message of error or invalid information will be printed instead (depending on the situation). | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Place information | String | The entire information of a place will be printed. |
| At least one place | String | A message showing that there is no place registered already so there is nothing to extract the information from. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the information to be printed |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#8: Access the information of the communities in a department. | | |
| **Summary:** | First, the system will make sure that there is at least on registered place (if there isn’t any, an “at least one place” message will be printed. For the user to retrieve all the information of all the communities in one department, he will need to provide an int that is selected from the list of departments that is printed. If there are no communities inside that place, a “no registered community in the department” message will be printed. If there is at least one community, then it will print the name of the communities that there are inside that department (without repeating their names). The user then must enter the int from the list of communities to return all the information from that community. Alternatively, if the information was not printed, then an error, incomplete information or invalid information will be printed. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| IntDepartment | int | * Must be one of the following options:  1. Choco 2. Valle 3. Cauca 4. Nariño  * Mandatory |
|  | IntCommunity | int | * Must select one of the numbers that represents one of the community names from the list of printed communities. * Mandatory |
| **Results or postcondition** | First, the system will ensure there's at least one registered place. If none exist, it'll print a message indicating the need for at least one place. To retrieve information about communities within a specific department, the user must select a department from a displayed list by providing its corresponding integer. If no communities exist within the selected department, a message stating "no registered community in the department" will appear. If there's at least one community, the system will display the names of the communities within that department, avoiding repetition. The user then inputs the integer corresponding to the desired community from the list displayed to retrieve its information. In case the information isn't printed, an error message indicating incomplete or invalid information will be displayed. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Community information | String | The entire information of the selected community in the department will be printed. |
| At least one place | String | A message showing that there is no place registered already so there is nothing to extract the information from. |
| No Community in the department | String | A message stating that the entered department doesn’t contain any communities will be printed. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the information to be printed will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#9: Access the information of communities based on their biggest problematic. | | |
| **Summary:** | Firstly, it's designed to print out all information grouped by community, particularly highlighting areas facing challenges “absence of hospitals” and “lack of schools”. Before proceeding with the printing process, it verifies if both challenges do have any community confronting this challenge (if not, an empty message will be printed under the empty challenge). Should any errors arise during the printing process, the system provides appropriate feedback. This includes messages indicating an error, incomplete information, or invalid data, depending on the specific issue encountered. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| Main problematic | Challenge | * Must select one of the following options:   + Absence of hospitals   + Lack of schools |
| **Results or postcondition** | The system will print out all the information (grouped by community) about all the communities that show either of those main problematics (absence of hospitals or lack of schools). If both of the mentioned challenges don’t make part of the communities registered in the program, an empty challenge message will be printed. If the information can’t be printed, it will show a message of error, incomplete information, or invalid information depending on the cause of the error. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Community information | String | All the information stored about the communities that had shown either of the two challenges (absence of hospitals or lack of schools) will be printed. |
| Empty challenge | String | A message showing that there is no community that is going through “lack of schools” and/or “absence of hospitals” within the program will be printed. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the information to be printed will be printed. |
| Incomplete information | String | A message stating that not all the mandatory fields where filled will be printed. |
| Invalid information | String | A message stating that there are mistakes in the inputted information (don’t follow the conditions) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#10: Display the name of the place with the most species diversity. | | |
| **Summary:** | When the user selects the option of consulting the name of the place with the most species, the system will extract and print the name of the place with the most diversity (highest number of species registered). If two or more places were to have the same number of registered species, any of the names could be printed. If the name is not printed, an error message will be printed instead. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| N/A | N/A | N/A |
| **Results or postcondition** | The system will use the information stored of every place and then compare which has the most registered species. If two or more places had the same number of registered species the system will choose one depending on the order the places are extracted and compared. In general, it will calculate the number of registered species, compare the numbers, and print the name of the most diverse place. If the name is not printed, a message of error will appear. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Most diverse place | String | A message stating that the place was registered successfully will be printed. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the place to be registered will be printed. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier and Name:** | Req#11: Display the three places with the biggest area. | | |
| **Summary:** | The system will allow the user to see which are the top three places with the biggest area (in km2) by printing it. In the case that there is not enough information (at least 3 registered places) then a “not enough data” message will be printed. If any error occurred during the process of printing the information, the error message would be printed. | | |
| **Inputs** | **Input name** | **Data Type** | **Conditions for valid values** |
| N/A | N/A | N/A |
| **Results or postcondition** | The system will compare the area information of the registered places, organize it, and print the names of the top three places with the biggest area. Before starting the extraction process, the system will make sure there are at least 3 registered places (if the condition is not met, a “not enough data” message will be printed). If all the places had the same area (or any population count number is repeated), it would choose the first 3 that come in order as the system reads the program. If the names are not printed, a message of error will appear. | | |
| **Outputs** | **Output Name** | **Data type** | **Format** |
| Success | String | A message stating that the place was registered successfully will be printed. |
| Not enough data | String | A message stating that there are not at least three places registered to do the necessary comparisons to extract the top 3 biggest areas. |
| Error | String | A message stating there was an error of unknown cause that didn't allow the place to be registered will be printed. |