it008472

Generated by Doxygen 1.8.17

Chapter 1

Coursework 2 Project Planning exercise

Planning is the keystone of successful programming, but it need not all be dry paperwork. The idea of experimenting with code in a series of feasibility studies can really help identify which approach to take.

In this exercise, you have some thinking to do, and you should get together with others in your peer group (other students) to discuss ideas.

First, though, grab a copy of this project by going to it on CSGitLab and using the "Fork" button near the top right corner.

Please set your version of the project to "private" after forking.

1.1 Goals

Identify the key goals you need to complete for the coursework

1. Being able to write functions in different files

1. The different functions should be able to call on each other when needed

1. All the 'musts' should be completed for the Project

1. Having good practices throughout my code - eg. using seperate files instead of writing my code in one large file

1. Using Plantuml - I would need to use plantuml in order to get high marks so learning about that would be very helpful

1.2 Requirements to fulfill goals

What do you need to be able to meet those goals? This can include clearer specifications, tests (what needs testing, how to test it?), knowledge etc.

1. Testing code that i have written - week7 tests, giving in test data and reviewing the outcome

1. Following the specifications - I would need to follow the specification and create the project as intended

1. More knowlege about linking files together - This project works with different files and i would need to be able to link them together

1. Learning about new software - Plantuml is a piece of software that i haven't used before so learning about it would be very helpful

1. Learning about good practices - Learning about good practices would make my code easier to write,read and modify

1.3 Dependencies (mapping goals/requirements etc)

Which requirements relate to which goals - think about drawing out a Product Breakdown for the coursework

1. moving tags depends on finding tags

1. creating a git repository depends on creating a project

1. the wizard depends on every other function that will be created

1. creating a plantuml file will depend on finding and locating folders/directories

1. renaming files will depend on creating files/projects

Hint: This might look something like this:

```
graph LR

    Project --> 1_must_haves
    subgraph "Must have"
    1_must_haves --> 1_create_folder_structure
    1_must_haves --> 2_abort_if_exists
    1_must_haves --> 3_initialise_git
    1_must_haves --> 4_feature_management
    end

    4_feature_management --> manage_features
    subgraph "manage features"
        manage_features(Manage Features)
        manage_features --> 1_shorthand_codes
        manage_features --> 2_shorthand_lookup
        manage_features --> 3_git_branch_per_feature
    end

    Project --> 2_should_haves
    subgraph "Should have"
    2_should_haves --> 5_renaming_features
    2_should_haves --> 6_moving_features
    2_should_haves --> 7_tree_diagram
    2_should_haves --> 8_time_workload_data
    2_should_haves --> 9_time_workload_on_diagram
    end

    Project --> 3_could_haves
    Project --> 4_wont_haves
    Project --> readme.md
```

1.4 Plan:

1.4.1 What experimental code can I write now? (feasibility studies)

1. Creating directory - creating feature

1. git directory - creating git repository

1. the main file - pm.c

1. renaming files - renaming features

1. Trying to recursivly look through a directory

Hint: try modifying the make file created in week4, consider the code in the videos for ideas about automating your code build, and the video on restucturing your project folder, for instance.

1.4.2 What were the results of trying things out?

1. Using the system function is the easiest way of creating directories

1. using the system funciton and then passing in the bash command for git produced a repository. However, i would need to create a .md file and then push it in order to push other items to the remote repository

1. the bash 'mv' function was useful for renaming files

1. you could look into directories through recursion or itteration

1. i can link files through header files which is useful for pm.c

Chapter 2

Coursework 2 - a project manager tool

Using the file system as the main datastore, this tool helps manage work breakdown methodology, milestones and time management.

The core elements will be strongly guided, and completion of them equates to approximately 40% of the marks. Beyond that, students should problem solve, in a graded hierarchy of difficulty and completion to achieve higher marks.

It is unrealistic to expect to meet all the feature criteria listed

Part of the task therefore involves deciding which elements to implement in order to maximise your mark.

Some of the features may be best implemented using shell (bash or zsh) scripting, or could be done directly in C.

Tests will be provided for 1...10, which will also define the command line arguments required (including the sub-command name for the features, where appropriate e.g. pm feature feature_name or pm feature move new_↩ feature_name).

Coursework submission will include creating project documentation (using doxygen) so code ***must*** be well commented. Any shell scripts or make files included in the solution ***must*** be included in the documentation. The submission itself ***must*** be the URL of the CSGitLab repository used.

2.1 Core functionality:

2.1.1 Must (up to 10% for each top level feature)

1. Be able to create basic file structure for project
2. Abort if requested project/feature name already exists under 'root' folder. Here 'root' does not mean the / root of the file system, but the folder from which the program is run.
   (a) Requires checking existing file system for matching name
   (b) Requires using branching (*if*) to exit program if necessary
3. Initialise git repository
   (a) ***Should*** set up CSGitLab project etc.
4. Feature management
   (a) ***Must*** implement a method of having a shorthand code for feature e.g. F1, F2.1..., stored in a file.
   (b) ***Must*** implement lookup to facilitate getting path from shorthand code
   (c) ***Should*** include setting up git branch as appropriate

2.1.2 Should (up to 10% for each top level feature)

1. Include mechanism for renaming features (subtrees)
2. Include mechanism for moving feature to new location in tree (folder hierarchy)
3. Output tree diagram - PBS or WBS (svg, using plantuml)
   (a) Requires tree walk (iterative or recursive)
   (b) ***Must*** exclude folders that start with a '.'
   (c) ***Should*** use the plantuml tool
   (d) ***Could*** implement from scratch (much harder, more marks)
4. Time/workload estimate information stored in files in subfolders
   (a) ***Should*** have mechanisms for adding these from the program not just editing the files
   (b) ***Should*** include subtrees costs in parent tree total
5. Time/workload added to output diagram
   (a) ***Could*** also produce Gantt chart (using plantuml)

2.1.3 Could (up to 10% for each top level feature)

1. Output diagram includes links (when used in browser, for example)
   (a) ***Should*** use plantuml to do this
2. Dependencies information across tree branches
   (a) ***Must*** identify relevant other paths in tree to do this

2.1.4 Elite challenges ("Won't do" in MoSCoW terms) (up to 20% for each top level feature)

1. Guided work breakdown wizard (Slightly advanced, would require interactive questions/answer handling)
   (a) Needs a number of sub-features, such as minimum time allocation threshold, user input parsing
2. Multi-user (Advanced, would require some form of permissions model)
   (a) may be best done using a traditional SQL database, but can use flat files. Complex task.
3. Available as web application (Advanced, probably easiest creating a simple embedded server)
   (a) sample code for simple communications between applications will be covered
4. GOAP recommendation of suitable pathway (Advanced, can use existing GOAP library, however)
   (a) GOAP uses a 'heap' data structure

Chapter 3

File Index

3.1 File List

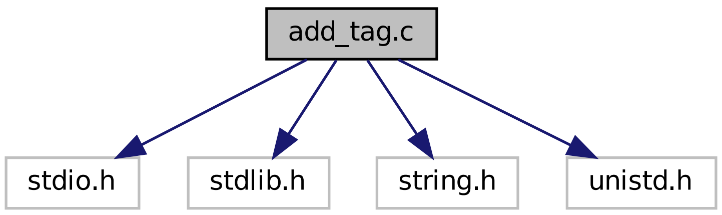
Here is a list of all files with brief descriptions:

Chapter 4

File Documentation

4.1 add_tag.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```
Include dependency graph for add_tag.c:



Functions

- int add_tag (char ∗name)

Variables

- char ∗ filename1 = ".pm_tag"

4.1.1 Function Documentation

4.1.1.1 add_tag()

```
int add_tag (
            char * name )
```

4.1.2 Variable Documentation

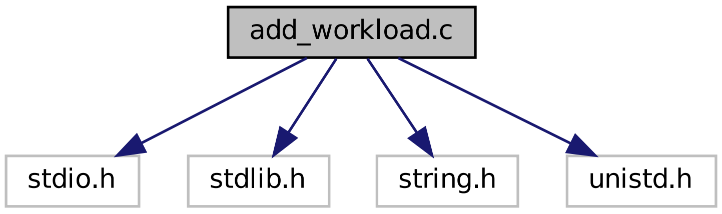
4.1.2.1 filename1

```
char* filename1 = ".pm_tag"
```

4.2 add_workload.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for add_workload.c:



Functions

- int add_workload (char ∗number)

Variables

- char ∗ filename = ".workload"

4.2.1 Function Documentation

4.2.1.1 add_workload()

```
int add_workload (
            char * number )
```

4.2.2 Variable Documentation

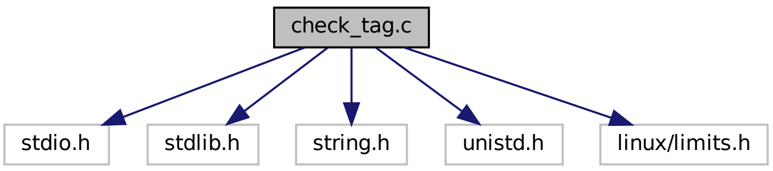
4.2.2.1 filename

```
char* filename = ".workload"
```

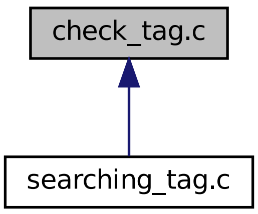
4.3 check_tag.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
```

Include dependency graph for check_tag.c:



This graph shows which files directly or indirectly include this file:



Functions

- char * check_tag (char *tag)

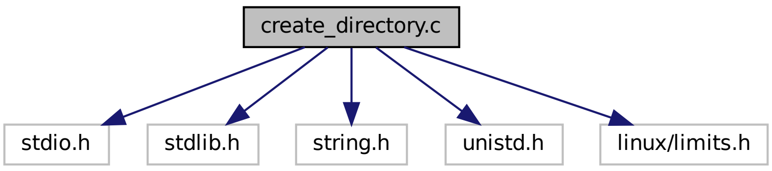
4.3.1 Function Documentation

4.3.1.1 check_tag()

```
char* check_tag (
             char * tag )
```

4.4 create_directory.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
```
Include dependency graph for create_directory.c:



Functions

- int create_directory (char *name)

Variables

- char temp [50]

4.4.1 Function Documentation

4.4.1.1 create_directory()

```
int create_directory (
             char * name )
```

The access function checks if a file already exists with the same name. If the access function returns a 0 then that means that the file can't be created because another file with the same name already exists and an exit status will show an error to the .

Here I create a parent directory called whatever the user inputed and I can create sub directories within the parent directory. The for loops runs until it reaches the end of the array called 'folders' which means that it would have created all the subdirectories needed by the end.

4.4.2 Variable Documentation

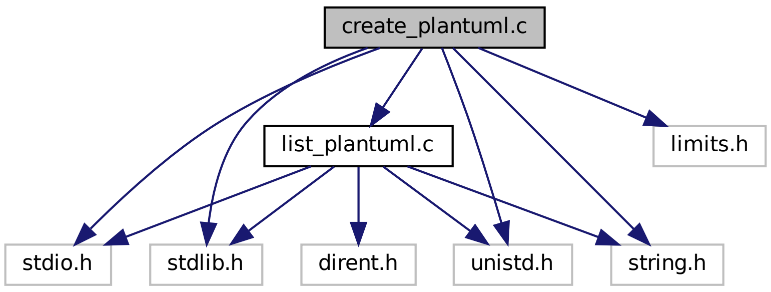
4.4.2.1 temp

```
char temp[50]
```

4.5 create_plantuml.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <string.h>
#include "list_plantuml.c"
```

Include dependency graph for create_plantuml.c:



Functions

- int create_plantuml (char ∗name)

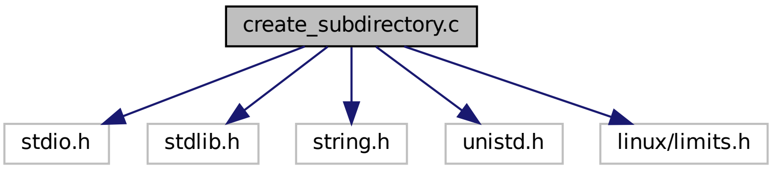
4.5.1 Function Documentation

4.5.1.1 create_plantuml()

```
int create_plantuml (
             char * name )
```

4.6 create_subdirectory.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
```
Include dependency graph for create_subdirectory.c:



Functions

- int create_subdirectory (char ∗name)

Variables

- char temp [50]

4.6.1 Function Documentation

4.6.1.1 create_subdirectory()

```
int create_subdirectory (
             char * name )
```

4.6.2 Variable Documentation

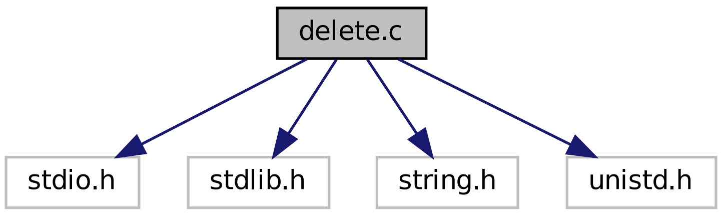
4.6.2.1 temp

```
char temp[50]
```

4.7 delete.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for delete.c:



Functions

- int delete_project (char ∗name)

Variables

- char temp2 [100]

4.7.1 Function Documentation

4.7.1.1 delete_project()

```
int delete_project (
            char * name )
```

4.7.2 Variable Documentation

4.7.2.1 temp2

```
char temp2[100]
```

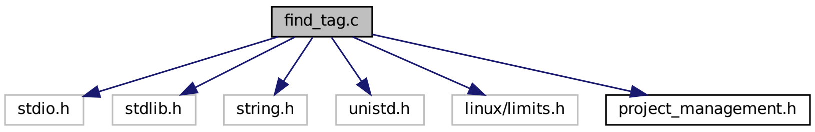
4.8 DrawUML.txt File Reference

4.9 exercise7.md File Reference

4.10 find_tag.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
#include "project_management.h"
```

Include dependency graph for find_tag.c:



Functions

- char * find_tag (char *tag)

4.10.1 Function Documentation

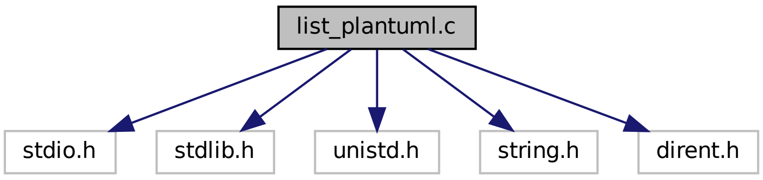
4.10.1.1 find_tag()

```
char* find_tag (
            char * tag )
```

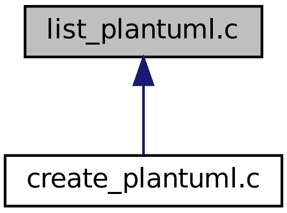
4.11 list_plantuml.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
```

Include dependency graph for list_plantuml.c:



This graph shows which files directly or indirectly include this file:



Functions

- void list_plantuml (const char ∗dirname, int level)

4.11.1 Function Documentation

4.11.1.1 list_plantuml()

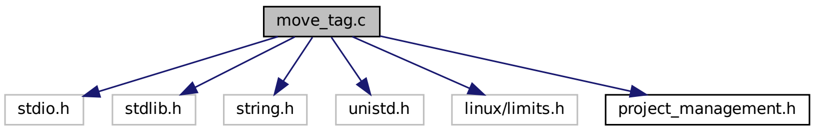
```
void list_plantuml (
            const char * dirname,
            int level )
```

4.12 menu.txt File Reference

4.13 move_tag.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
#include "project_management.h"
```

Include dependency graph for move_tag.c:



Functions

- void move_tag (char ∗tag1, char ∗tag2)

4.13.1 Function Documentation

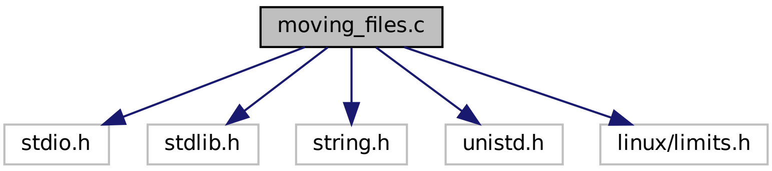
4.13.1.1 move_tag()

```
void move_tag (
            char ∗ tag1,
            char ∗ tag2 )
```

4.14 moving_files.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
#include <linux/limits.h>
```
Include dependency graph for moving_files.c:



Functions

- int moving_file (char ∗oldpath, char ∗newpath)

Variables

- char temp4 [100]

4.14.1 Function Documentation

4.14.1.1 moving_file()

```
int moving_file (
             char * oldpath,
             char * newpath )
```

4.14.2 Variable Documentation

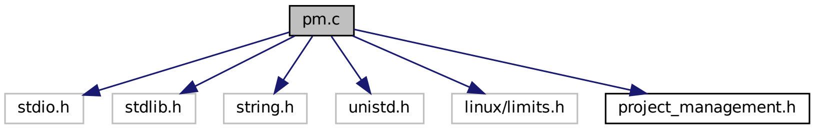
4.14.2.1 temp4

```
char temp4[100]
```

4.15 pm.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
#include "project_management.h"
```

Include dependency graph for pm.c:



Functions

- int main (int argc, char ∗argv[ ])

4.15.1 Function Documentation

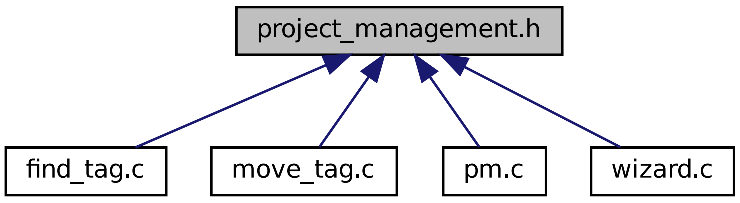
4.15.1.1 main()

```
int main (
            int argc,
            char ∗ argv[ ] )
```

This is the main function and all the other functions aren called from here

4.16 project_management.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int create_directory (char ∗name)
- int rename_project (char ∗oldname, char ∗newname)
- int delete_project (char ∗name)
- int add_tag (char ∗name)
- char ∗ check_tag (char ∗tag)
- char ∗ search_tag (char ∗tag)
- char ∗ find_tag (char ∗tag)
- void move_tag (char ∗tag1, char ∗tag2)
- int read_file ()
- int add_workload (char ∗number)
- int wizard ()
- int repository (char ∗name)
- int create_plantuml (char ∗name)
- int create_subdirectory (char ∗name)
- int read_menu ()
- int moving_file (char ∗oldpath, char ∗newpath)

4.16.1 Function Documentation

4.16.1.1 add_tag()

```
int add_tag (
             char * name )
```

4.16.1.2 add_workload()

```
int add_workload (
             char * number )
```

4.16.1.3 check_tag()

```
char* check_tag (
             char * tag )
```

4.16.1.4 create_directory()

```
int create_directory (
            char * name )
```

The access function checks if a file already exists with the same name. If the access function returns a 0 then that means that the file can't be created because another file with the same name already exists and an exit status will show an error to the .

Here I create a parent directory called whatever the user inputed and I can create sub directories within the parent directory. The for loops runs until it reaches the end of the array called 'folders' which means that it would have created all the subdirectories needed by the end.

4.16.1.5 create_plantuml()

```
int create_plantuml (
            char * name )
```

4.16.1.6 create_subdirectory()

```
int create_subdirectory (
            char * name )
```

4.16.1.7 delete_project()

```
int delete_project (
            char * name )
```

4.16.1.8 find_tag()

```
char* find_tag (
            char * tag )
```

4.16.1.9 move_tag()

```
void move_tag (
            char * tag1,
            char * tag2 )
```

4.16.1.10 moving_file()

```
int moving_file (
             char * oldpath,
             char * newpath )
```

4.16.1.11 read_file()

```
int read_file ( )
```

4.16.1.12 read_menu()

```
int read_menu ( )
```

4.16.1.13 rename_project()

```
int rename_project (
             char * oldname,
             char * newname )
```

4.16.1.14 repository()

```
int repository (
             char * name )
```

4.16.1.15 search_tag()

```
char* search_tag (
             char * tag )
```

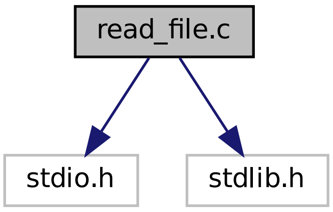
4.16.1.16 wizard()

```
int wizard ( )
```

4.17 read_file.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for read_file.c:



Functions

- int read_file ()

4.17.1 Function Documentation

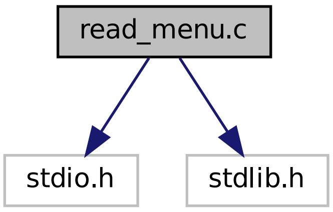
4.17.1.1 read_file()

```
int read_file ( )
```

4.18 read_menu.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for read_menu.c:



Functions

- int read_menu ()

4.18.1 Function Documentation

4.18.1.1 read_menu()

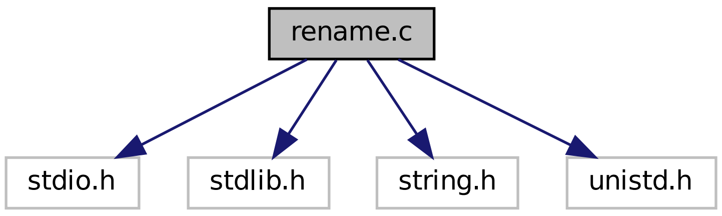
```
int read_menu ( )
```

4.19 README.md File Reference

4.20 rename.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for rename.c:



Functions

- int rename_project (char ∗oldname, char ∗newname)

Variables

- char temp [50]

4.20.1 Function Documentation

4.20.1.1 rename_project()

```
int rename_project (
            char * oldname,
            char * newname )
```

4.20.2 Variable Documentation

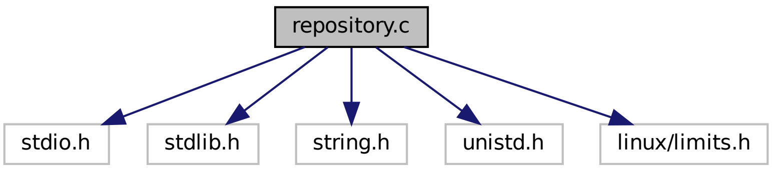
4.20.2.1 temp

```
char temp[50]
```

4.21 repository.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
```

Include dependency graph for repository.c:



Functions

- int repository (char *name)

Variables

- char temp15 [100]
- char temp16 [100]
- char temp17 [100]
- char temp18 [100]

4.21.1 Function Documentation

4.21.1.1 repository()

```
int repository (
            char * name )
```

4.21.2 Variable Documentation

4.21.2.1 temp15

```
char temp15[100]
```

4.21.2.2 temp16

```
char temp16[100]
```

4.21.2.3 temp17

```
char temp17[100]
```

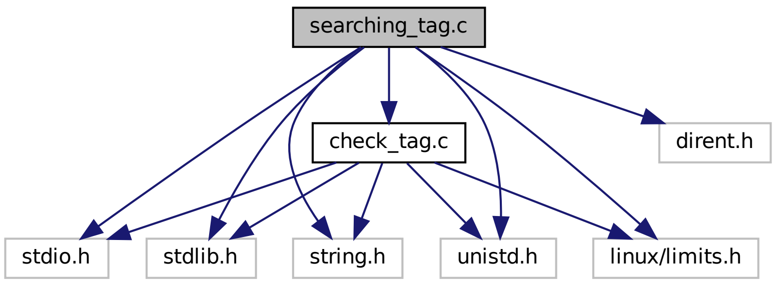
4.21.2.4 temp18

```
char temp18[100]
```

4.22 searching_tag.c File Reference

```
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
#include "check_tag.c"
```

Include dependency graph for searching_tag.c:



Functions

- char ∗ search_tag (char ∗tag)

4.22.1 Function Documentation

4.22.1.1 search_tag()

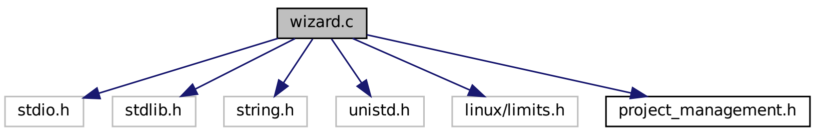
```
char∗ search_tag (
            char ∗ tag )
```

4.23 wizard.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/limits.h>
```

```
#include "project_management.h"
```

Include dependency graph for wizard.c:



Functions

- int wizard ()

4.23.1 Function Documentation

4.23.1.1 wizard()

```
int wizard ( )
```

4.24 wizard_menu.txt File Reference

Index