Computer Software II

FINAL PROJECT: SPATIAL DATABASE

Lolelanji Simposya
081441179

- *Introduction:*

In today's modern society, computers and automation serves as the basis of many different industries and improve our daily lives at the same time. Knowledge of computer programming can prove to be beneficial especially for Engineers, who utilize computers to solve many different problems as well as develop many different new technologies.
In this project, students are required to develop a spatial database program that maintains a database of spatial objects (e.g. shops, facilities).
Students will learn about different programming techniques and strengthen their understanding of the c programming language.

Spatial Database:

A spatial database is a database that is enhanced to store and access spatial data or data that defines a geometric space.
The development of the spatial database for this final project was very interesting as such a service would prove to be very useful for new visitors to the campus as well as students and professors who frequently use the different facilities on the campus, such as the cafeterias and shops. The program will be based on the 2D Mesh map of the university's Higashiyama campus. The user will be able to find specific different items on the map as well as find items closest to their position in within a distance threshold. The program will also allow the user to estimate the approximate time required to travel to a certain location on campus based on their location.

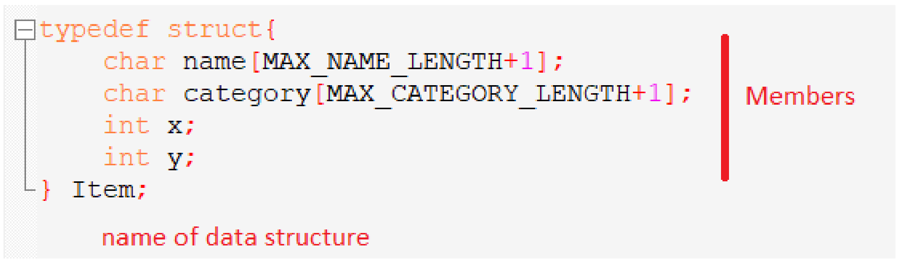
- *Explanation of program design, algorithm and approach*

The development of the program consisted of writing code for four major sections/milestones.

1. First Milestone:
- Development of Data Structure
- Insertion of data item
- Printing data items
- Main menu

Development of data structure:
A structure is a collection of values, possibly of different types. In the first milestone, the data structure, named *Item*, was defined for the different items in the database. This allowed us to create four different elements for each item, the elements consisted of:

- Name of Item (string type)
- Category of Item (string type)
- X-coordinate of Item (integer type)
- Y-coordinate of Item (integer type)



```
typedef struct{
     char name[MAX_NAME_LENGTH+1];
     char category[MAX_CATEGORY_LENGTH+1];
     int x;
     int y;
} Item;
```

Figure 1: Data structure for Item

Insertion of Data item:
A function called *insert()* is created to allow the user to insert new items into the data structure; the function reads data from the keyboard and stores it in the data structure. The function can allow the user to update the database if required.

Printing data items:
A function called *print_all()_items is* created to allow the user to display all the current items stored in the database. The function can inform the user of the different items in the database.

Main menu:
A main menu function is created in the main_menu.c file to list the different functions of the program and allow the user to choose the different functions.

2. Second Milestone:

- Loading data items from a file
- Saving data items into a file

Loading data items from a file:
A function called *load_item()s* is created to read data from a file. The function asks the user for the file name to be opened and then it reads all the data items stored in the file.

Saving data items into a file:
A function called *save_item()s* is created to write data into a file. The function asks the user for the file name to be opened and then it writes the data into the file.

3. Third Milestone:

- Searching data items based on name
- Searching data items based on category
- Printing pages for search result

Searching data items based on name:
A function called *find_items_by_name()* is created to search data items in the database based on the name. The function asks the user to input a name to be searched then it finds the items in the database whose name matches the input. The result is then passed on to the *print_page* function to be displayed in an html file.

Searching data items based on category:
A function called *find_items_by_category()* is created to search data items in the database based on the category. The function asks the user to input a category to be searched then it finds the items whose category matches the input category. The result is then passed on to the *print_page()* function to be displayed in an html file.

Printing pages for search result:
A function called *print_page_sample()* is created in the page.c file to display the results of the various functions found in the database.c file. The function lists the results in an html file for the user to access.

4. Fourth Milestone:

- Spatial search (Range query)
- Spatial search (Nearest neighbor query)
- Original search (Estimated travel time)

Spatial search (Range query):
A function called *find_items_by_range()* is created to search data items based on the distance from the user's position.
The function asks the user for:

- His/her position (x and y coordinates)
- A distance threshold (in meters)

The function then finds the items whose distance is less than or equal to the threshold distance. The results are listed and shown by the *print_page* function.

Spatial search (Nearest neighbor query):
A function called *find_nearest_neighbor()* is created to search the closest data items to the user's current position. The function asks the user for his/her position (x and y coordinates) and proceeds to find the item which has the minimum distance from the user. The result is listed and shown by the *print_page_sample()* function.

Original search (Estimated travel time):
A function called *time()* is created to find the estimated time it will take the user to get from his/her location to a specified data item in the database. The function asks the user for his/her position (x and y coordinates) and the data item the user wishes to travel to. The function then calculates the estimated time required to reach the destination if walking or using a bicycle.

The overall setup of the program is shown in the skeleton figure below:

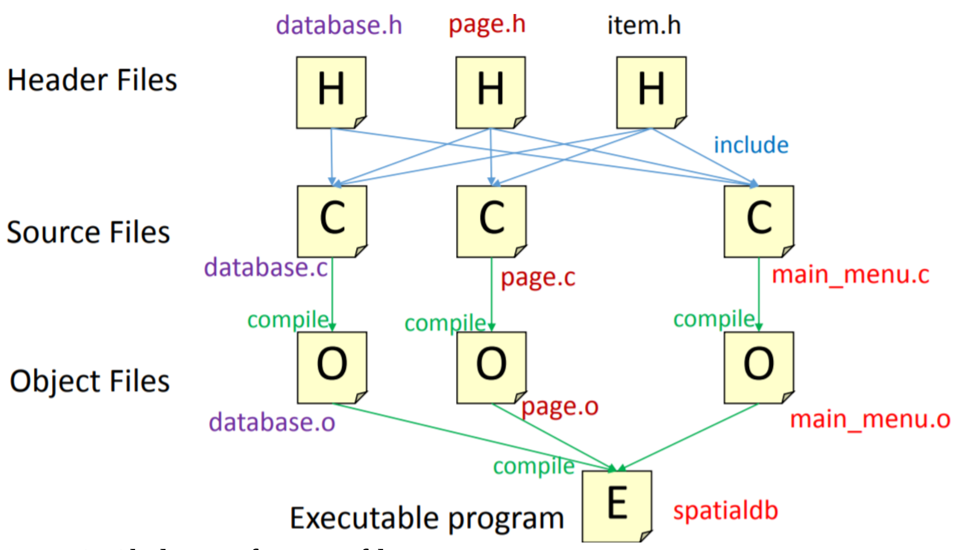


Figure 2: Skeleton of source files

The results of the spatial database are shown on the campus map located in the html result file.

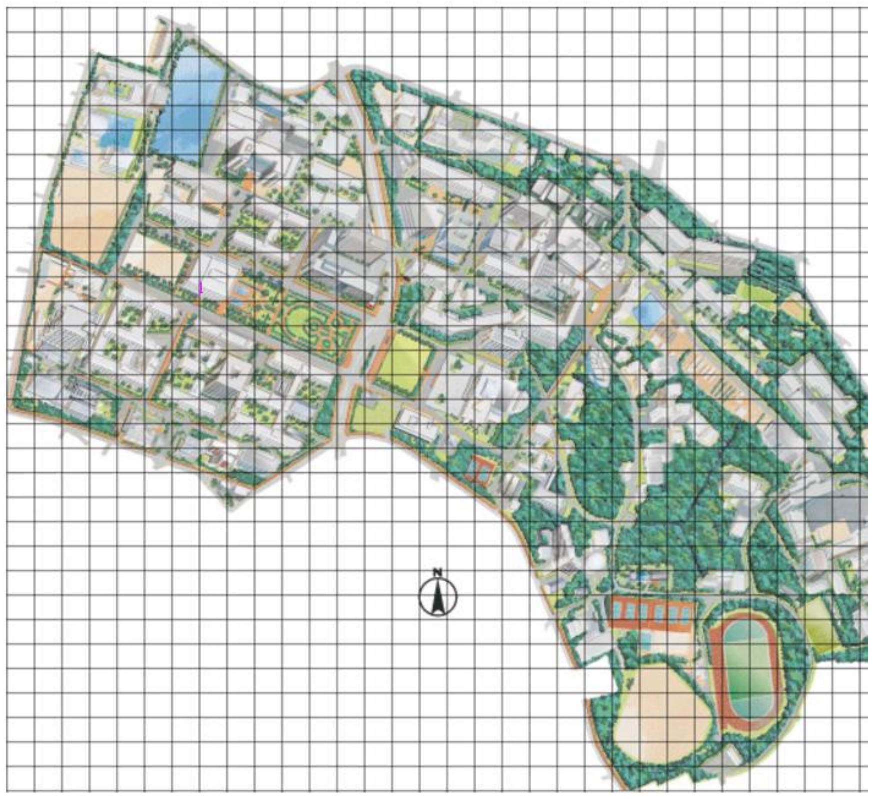
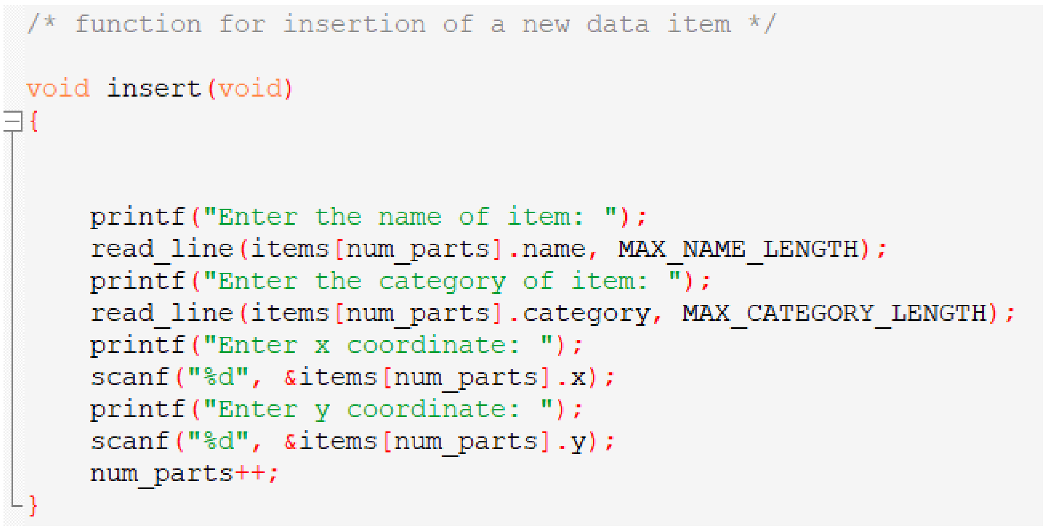


Figure 3: 2D Mesh map of Higashiyama campus

- *All Source Codes used in the Project:*

➢ FIRST MILESTONE SOURCE CODES:

- *insert()*



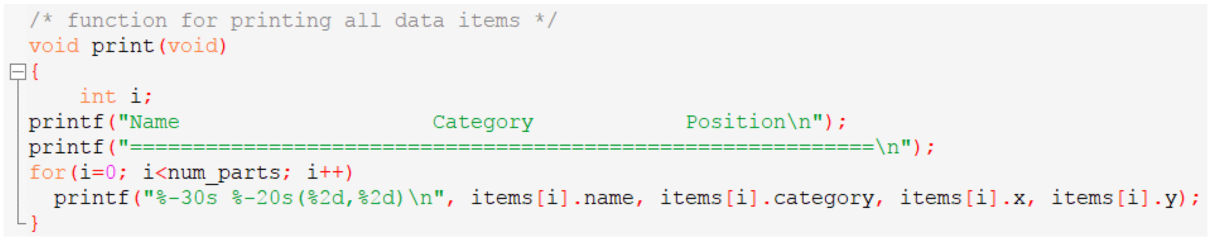
```c
/* function for insertion of a new data item */

void insert(void)
{


    printf("Enter the name of item: ");
    read_line(items[num_parts].name, MAX_NAME_LENGTH);
    printf("Enter the category of item: ");
    read_line(items[num_parts].category, MAX_CATEGORY_LENGTH);
    printf("Enter x coordinate: ");
    scanf("%d", &items[num_parts].x);
    printf("Enter y coordinate: ");
    scanf("%d", &items[num_parts].y);
    num_parts++;
}
```

Figure 4: Insert function source code

The insert function source code is shown in figure 3, the user is asked to input four elements of the spatial item and the values are stored in the corresponding data structure elements. The user is limited to:

- A maximum name length of 30 characters
- A maximum category name length of 20 characters
- A maximum number of 100 parts for the database.
- A maximum x coordinate of 32
- A maximum y coordinate of 32

- *Print()*



```c
/* function for printing all data items */
void print(void)
{
    int i;
printf("Name                    Category            Position\n");
printf("===========================================================\n");
for(i=0; i<num_parts; i++)
  printf("%-30s %-20s(%2d,%2d)\n", items[i].name, items[i].category, items[i].x, items[i].y);
}
```

Figure 5: Print function source code

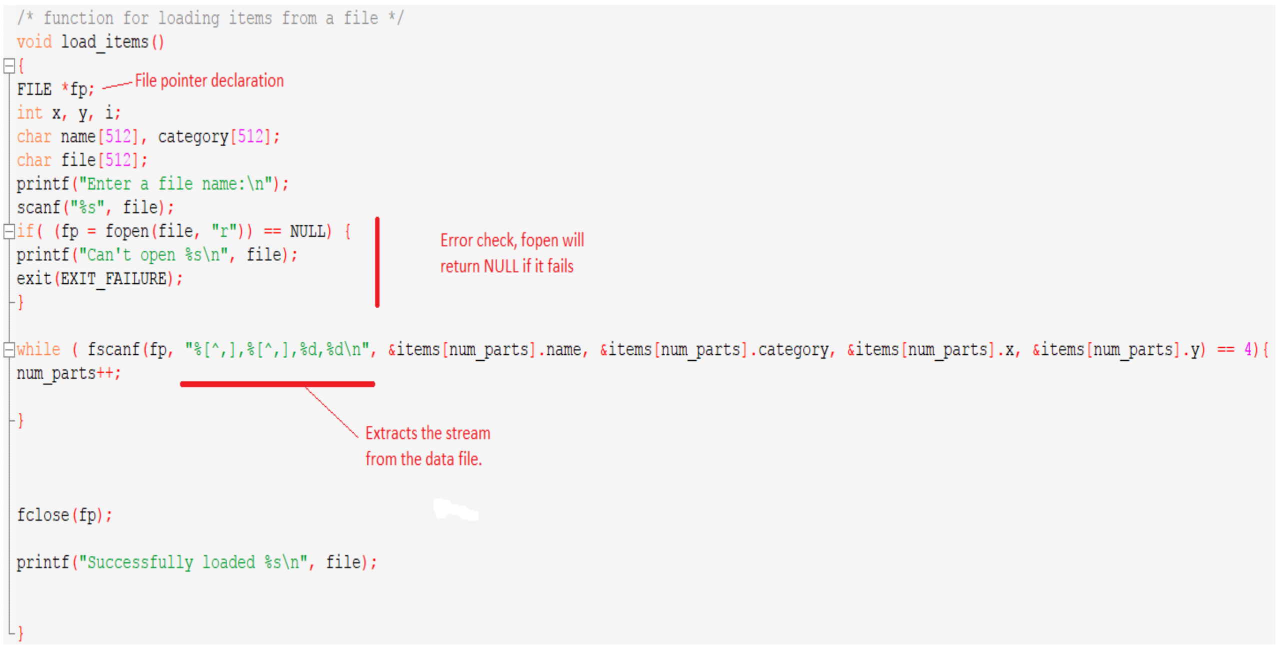
The print function displays the data found in the database in a table format. The individual data characters are printed out and each element is displayed under a heading in the table.

- "%-30s" ensures that the name is given 30 characters and printed to the left
- "%-20s" ensures that the category is given 20 characters and printed to the left.
- "%2d" outputs an integer value that fills at most 2 characters.

The spaces between the number formats ensures the values are separated when printed out in table format.
A loop is created to repeatedly print out all the parts currently in the database.

➢ SECOND MILESTONE SOURCE CODES:

- *load_items()*



```
/* function for loading items from a file */
void load_items()
{
FILE *fp;
int x, y, i;
char name[512], category[512];
char file[512];
printf("Enter a file name:\n");
scanf("%s", file);
if( (fp = fopen(file, "r")) == NULL) {
printf("Can't open %s\n", file);
exit(EXIT_FAILURE);
}

while ( fscanf(fp, "%[^,],%[^,],%d,%d\n", &items[num_parts].name, &items[num_parts].category, &items[num_parts].x, &items[num_parts].y) == 4){
num_parts++;

}


fclose(fp);

printf("Successfully loaded %s\n", file);


}
```

File pointer declaration
Error check, fopen will return NULL if it fails
Extracts the stream from the data file.

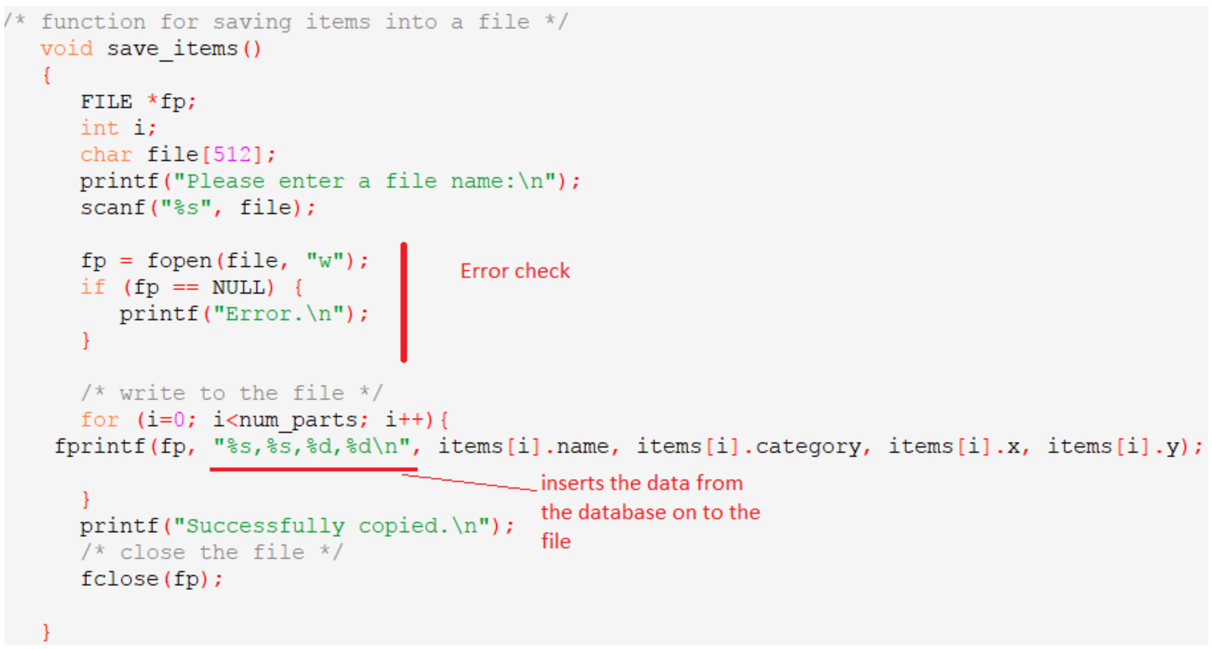
Figure 6: Load function source code

Accessing the information, or string in a file is done through a file pointer. The function first asks the user to input the name of the file to be opened, it then checks if the file can be opened. If no errors occur then the program extracts the data streams from the file. A special format is used to ensure the correct streams are extracted:

- "%[^,]” will scan the stream until a comma is encountered and end the extraction, this ensures that the item name and item category are separated and correctly extracted from the string.
- “%d” will extract the integer value of the stream.
- “\n” will move to the next line.

A while loop is implemented to extract the data from all the lines.
The *fclose()* function allows the program to successfully close the file after the process is complete.

- *Save_item()*



```
/* function for saving items into a file */
   void save_items()
   {
      FILE *fp;
      int i;
      char file[512];
      printf("Please enter a file name:\n");
      scanf("%s", file);

      fp = fopen(file, "w");
      if (fp == NULL) {
         printf("Error.\n");
      }

      /* write to the file */
      for (i=0; i<num_parts; i++){
    fprintf(fp, "%s,%s,%d,%d\n", items[i].name, items[i].category, items[i].x, items[i].y);

      }
      printf("Successfully copied.\n");
      /* close the file */
      fclose(fp);

   }
```

Figure 7: Save function source code

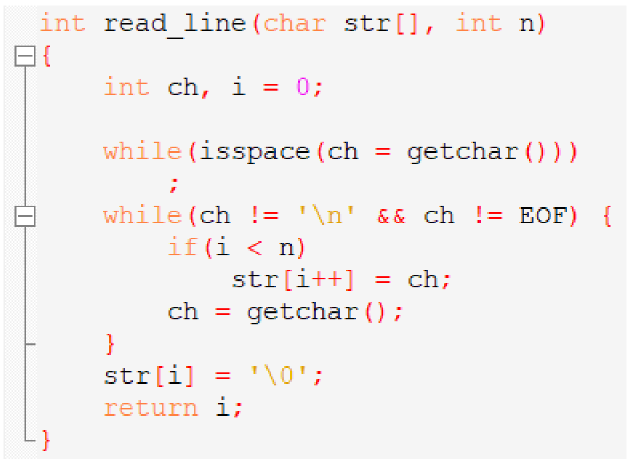
The function first asks the user to input the name of the file to be saved including the extension, it then checks if the file can be opened for writing. If no errors occur then the program inputs the data on to the new file. A standard format is used to input the data on to the file:

- "%s" prints the string for the item name and category name.
- "%d" prints the integer value of the coordinates.

A for loop is implemented to print all the items of the database.
The *fclose()* function allows the program to successfully close the file after the process is complete.

➢ THIRD MILESTONE SOURCE CODES:

- *read_line()*

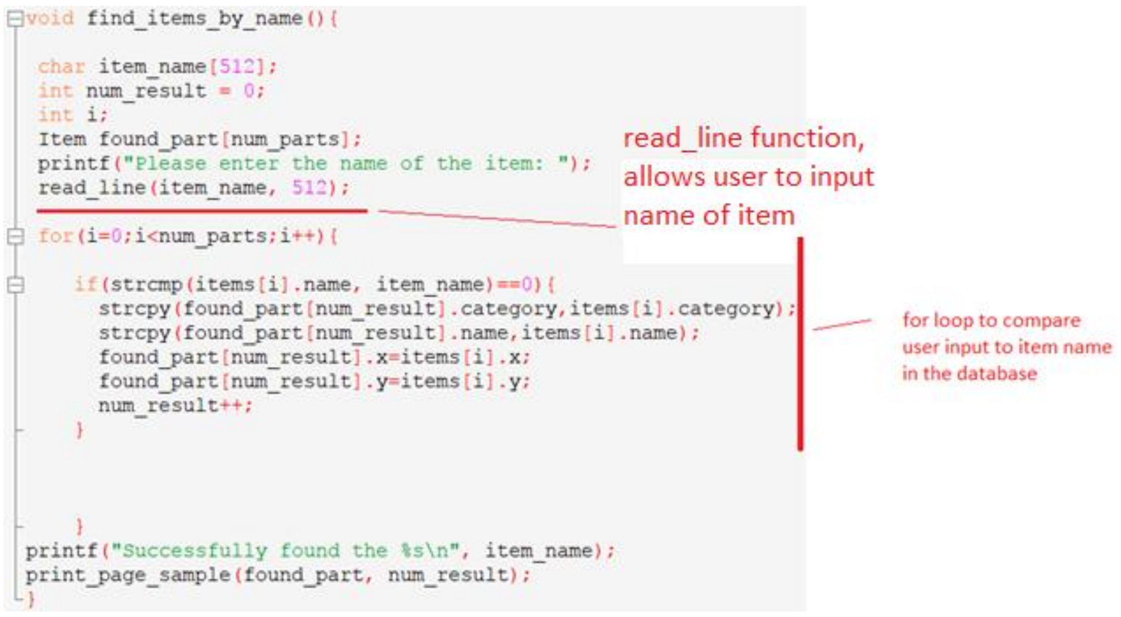


```c
int read_line(char str[], int n)
{
    int ch, i = 0;

    while(isspace(ch = getchar()))
        ;
    while(ch != '\n' && ch != EOF) {
        if(i < n)
            str[i++] = ch;
        ch = getchar();
    }
    str[i] = '\0';
    return i;
}
```

Figure 8: Read line function source code

The *read_line()* function, is important for user input of strings, it skips leading white spaces and then reads the remainder of the input line and stores it in string and shortens the line if the input exceeds the number characters allowed.

- *find_items_by_name()*



```c
void find_items_by_name(){

  char item_name[512];
  int num_result = 0;
  int i;
  Item found_part[num_parts];
  printf("Please enter the name of the item: ");
  read_line(item_name, 512);

  for(i=0;i<num_parts;i++){

     if(strcmp(items[i].name, item_name)==0){
       strcpy(found_part[num_result].category,items[i].category);
       strcpy(found_part[num_result].name,items[i].name);
       found_part[num_result].x=items[i].x;
       found_part[num_result].y=items[i].y;
       num_result++;
     }



  }
  printf("Successfully found the %s\n", item_name);
  print_page_sample(found_part, num_result);
}
```

read_line function, allows user to input name of item
for loop to compare user input to item name in the database

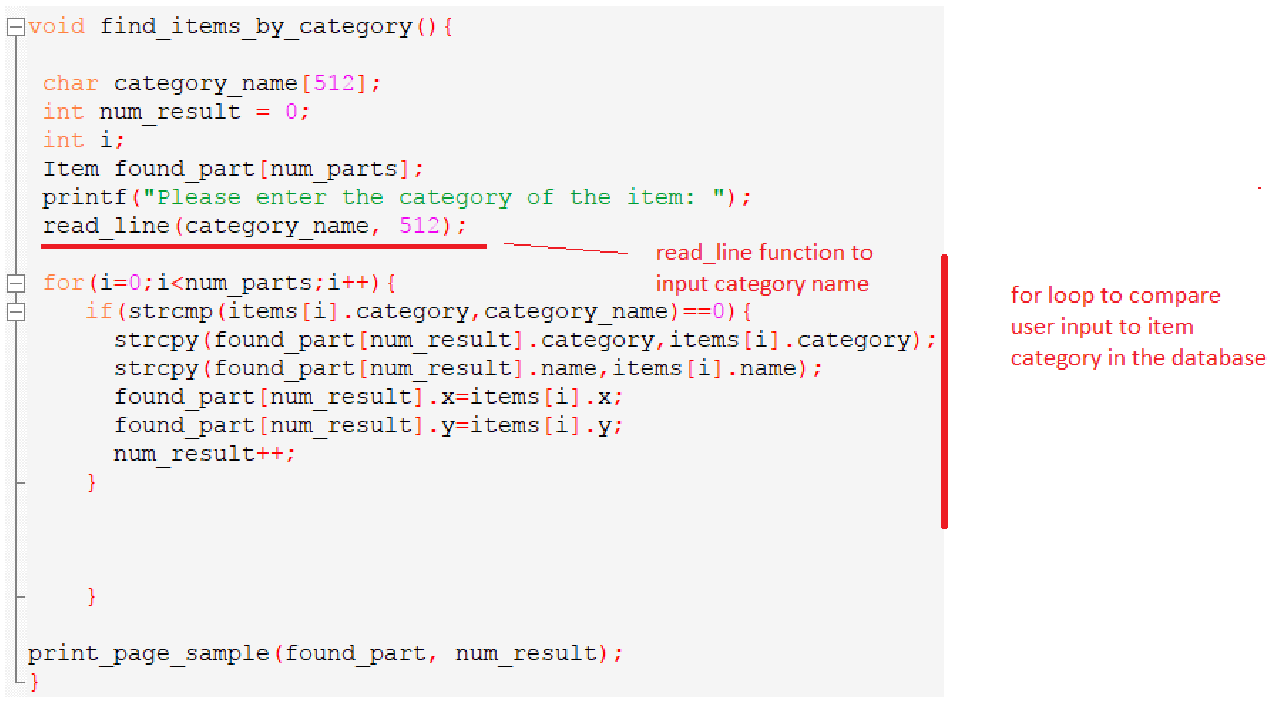
Figure 8: Find item name function source code

A new data structure of item type named “found_part” is created; this data structure has an identical format to the item data structure. This data structure will store the found items.

The function first asks the user to input the name of the item name, the *read_line()* function extracts the input name and stores it in the item_name array. A for loop is then implemented to repeat for the number of parts currently in the database. An if statement within the for loop uses the *strcmp()* function to compare the name of the item name to any of the names listed in the database. If the name matches then the result of the function is 0. If the name matches then the *strcpy()* function is implemented and copies name and category strings from the original item data structure to the found part data structure. The x and y coordinated are equated and the found part acquires all the required data for the specific spatial database item. The loop repeats if more items are found and the num_result also increases.

The result is then sent to the *print_page_sample()* function, to be displayed on the result html file.

- *find_items_by_category()*



```
void find_items_by_category(){

 char category_name[512];
 int num_result = 0;
 int i;
 Item found_part[num_parts];
 printf("Please enter the category of the item: ");
 read_line(category_name, 512);

 for(i=0;i<num_parts;i++){
    if(strcmp(items[i].category,category_name)==0){
      strcpy(found_part[num_result].category,items[i].category);
      strcpy(found_part[num_result].name,items[i].name);
      found_part[num_result].x=items[i].x;
      found_part[num_result].y=items[i].y;
      num_result++;
    }



    }

print_page_sample(found_part, num_result);
}
```

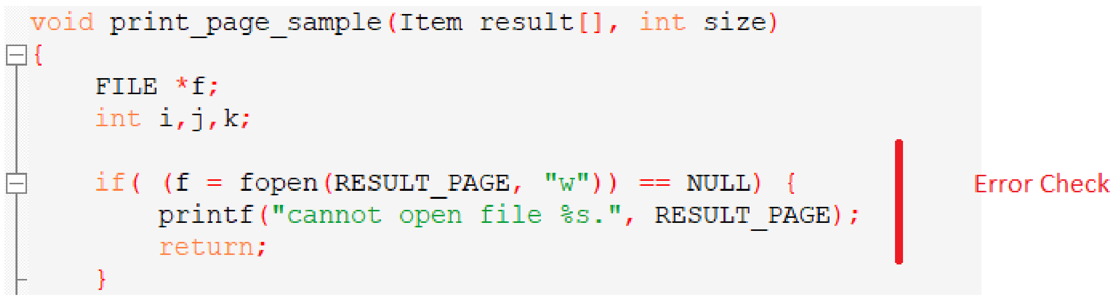
read_line function to input category name
for loop to compare user input to item category in the database

Figure 9: Find item category function source code

A new data structure of item type named “found_part” is created; this data structure has an identical format to the item data structure. This data structure will store the found items. The function first asks the user to input the category of the item, the *read_line()* function extracts the input category and stores it in the category_name array. A for loop is then implemented to repeat for the number of parts currently in the database. An if statement within the for loop uses the *strcmp()* function to compare the category of the item to the categories listed in the database. If the category matches then the result of the function is 0. If the category matches then the *strcpy()* function is implemented and copies name and category strings from the original item data structure to the found part data structure. The x and y coordinated are equated and the found part acquires all the required data for the specific spatial database item. The loop repeats if more items are found and the num_result also increases.
The result is then sent to the *print_page_sample()* function, to be displayed on the result html file.

- Print page sample

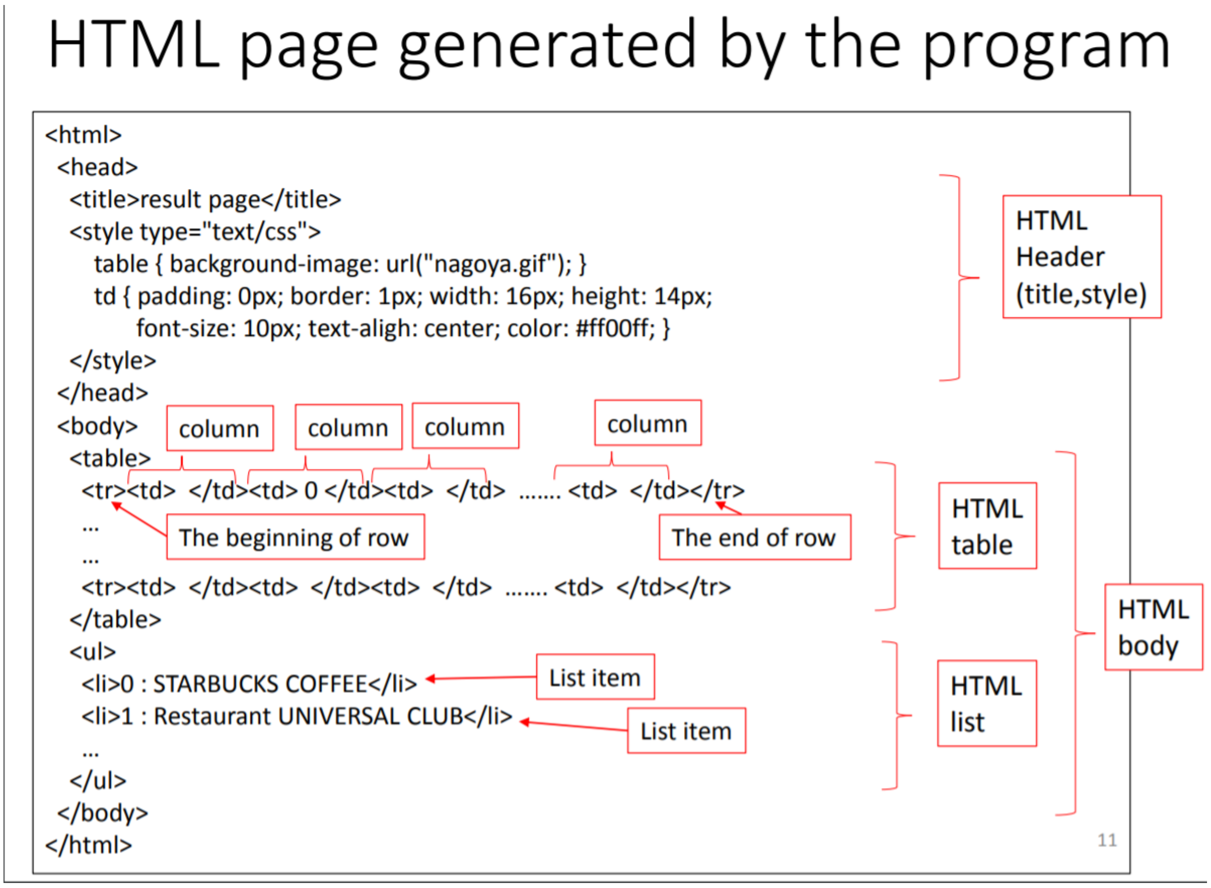
We write a function called *print_page()* which shows the results page. The search result is provided by the functions *find_items_by_name()* and *find_items_by_category()*. The program generates a HTML file that contains the search result.



```
void print_page_sample(Item result[], int size)
{
    FILE *f;
    int i,j,k;

    if( (f = fopen(RESULT_PAGE, "w")) == NULL) {
        printf("cannot open file %s.", RESULT_PAGE);
        return;
    }
```

Figure 10: print page function first section source code

HTML is the international standard format for web pages. The general format for the results.html page is shown in the figure below:

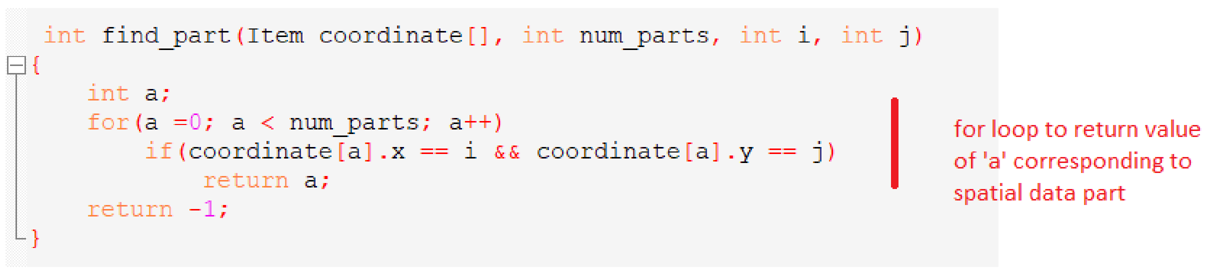


HTML page generated by the program

```
<html>
 <head>
  <title>result page</title>
  <style type="text/css">
    table { background-image: url("nagoya.gif"); }
    td { padding: 0px; border: 1px; width: 16px; height: 14px;
        font-size: 10px; text-aligh: center; color: #ff00ff; }
  </style>
 </head>
 <body>
  <table>
   <tr><td> </td><td> 0 </td><td> </td> ....... <td> </td></tr>
   ...
   ...
   <tr><td> </td><td> </td><td> </td> ....... <td> </td></tr>
  </table>
  <ul>
   <li>0 : STARBUCKS COFFEE</li>
   <li>1 : Restaurant UNIVERSAL CLUB</li>
   ...
  </ul>
 </body>
</html>
```

Figure 11: HTML page generated by the program

- *find_part()*

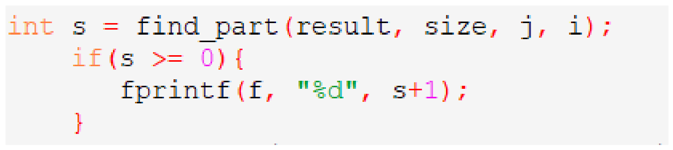


```c
int find_part(Item coordinate[], int num_parts, int i, int j)
{
    int a;
    for(a =0; a < num_parts; a++)
        if(coordinate[a].x == i && coordinate[a].y == j)
            return a;
    return -1;
}
```

for loop to return value of 'a' corresponding to spatial data part

Figure 12: find part source code

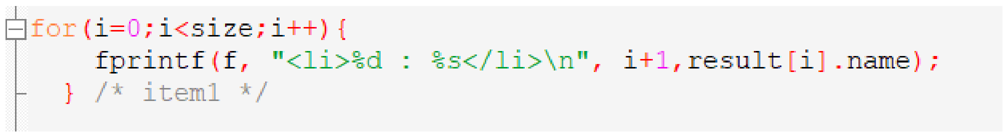
We first write a function that finds the corresponding spatial data item and enable us to print the list of items in the result html page with item labels corresponding to X=j, Y=i. After finding the value for a, the program can then print the list using the following function:



```c
int s = find_part(result, size, j, i);
    if(s >= 0){
       fprintf(f, "%d", s+1);
    }
```

Figure 13: print list function source code

The printing of list names is done through the use of the following for loop:

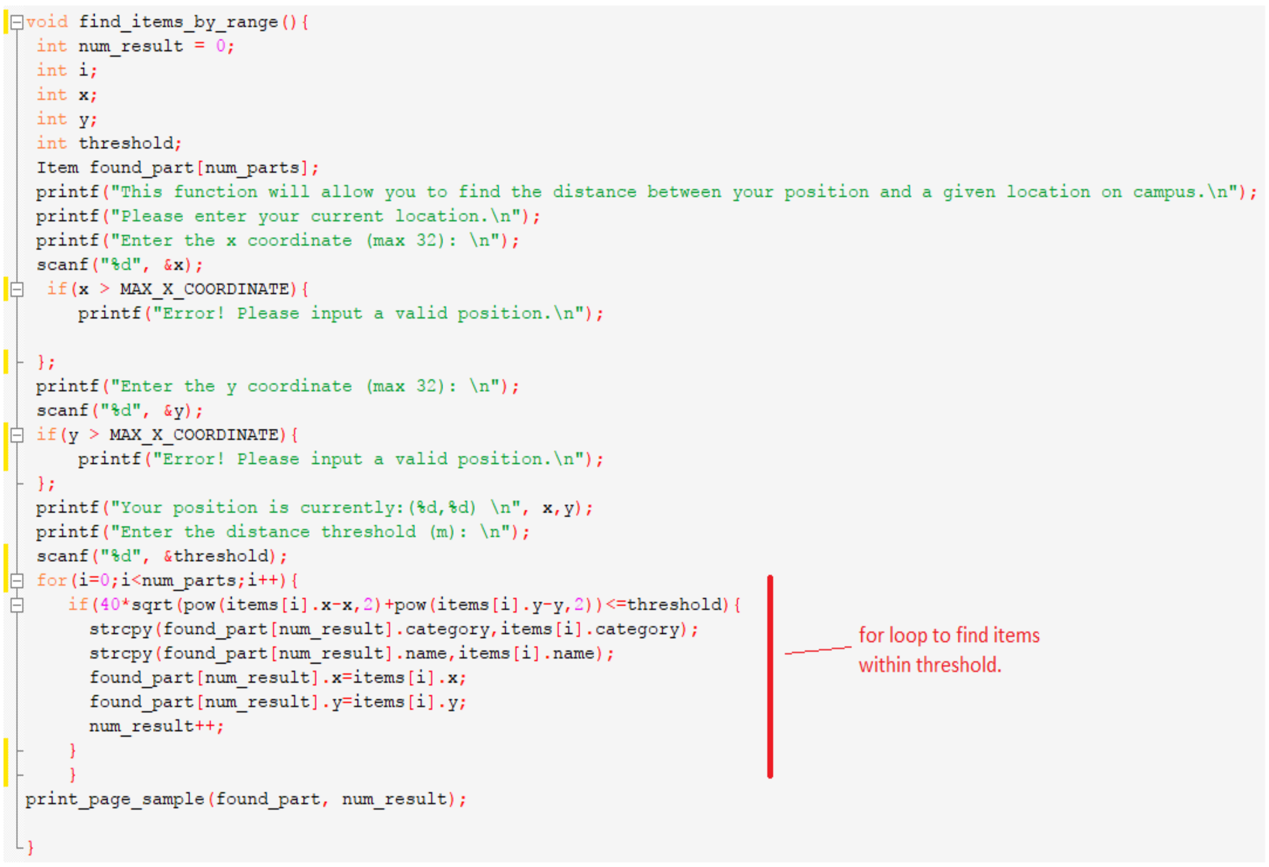


```c
for(i=0;i<size;i++){
        fprintf(f, "<li>%d : %s</li>\n", i+1,result[i].name);
    } /* item1 */
```

Figure 14: print names function source code

➢ FOURTH MILESTONE SOURCE CODES:

- *find_items_by_range()*



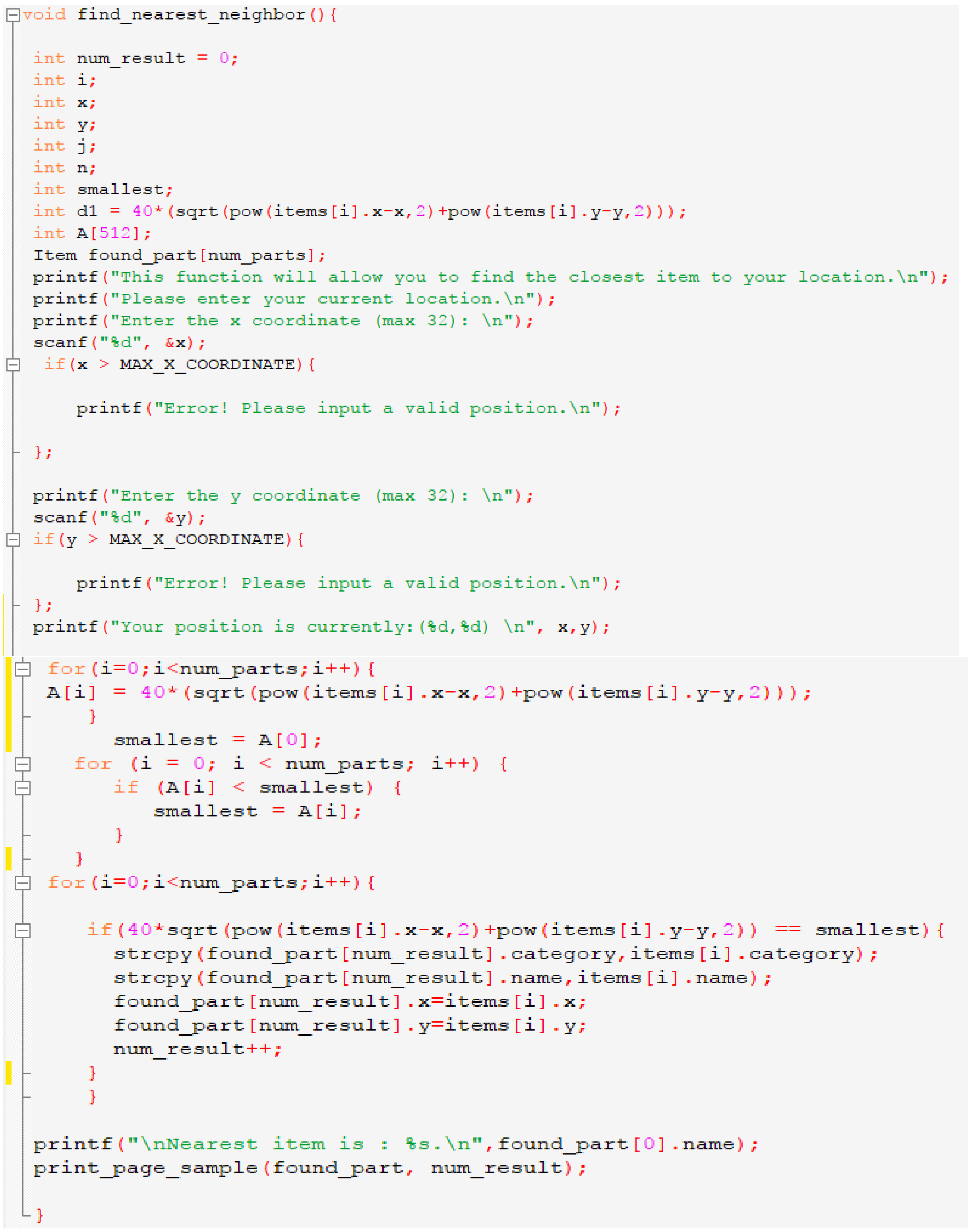
```c
void find_items_by_range(){
  int num_result = 0;
  int i;
  int x;
  int y;
  int threshold;
  Item found_part[num_parts];
  printf("This function will allow you to find the distance between your position and a given location on campus.\n");
  printf("Please enter your current location.\n");
  printf("Enter the x coordinate (max 32): \n");
  scanf("%d", &x);
   if(x > MAX_X_COORDINATE){
      printf("Error! Please input a valid position.\n");

  };
  printf("Enter the y coordinate (max 32): \n");
  scanf("%d", &y);
  if(y > MAX_X_COORDINATE){
      printf("Error! Please input a valid position.\n");
  };
  printf("Your position is currently:(%d,%d) \n", x,y);
  printf("Enter the distance threshold (m): \n");
  scanf("%d", &threshold);
  for(i=0;i<num_parts;i++){
     if(40*sqrt(pow(items[i].x-x,2)+pow(items[i].y-y,2))<=threshold){
       strcpy(found_part[num_result].category,items[i].category);
       strcpy(found_part[num_result].name,items[i].name);
       found_part[num_result].x=items[i].x;
       found_part[num_result].y=items[i].y;
       num_result++;
     }
     }
 print_page_sample(found_part, num_result);

}
```

Figure 15: Find items by range function source code

A new data structure of item type named “found_part” is created; this data structure has an identical format to the item data structure. This data structure will store the found items. The function first asks the user to input his/her current coordinates and the distance threshold.

A for loop is then implemented to repeat for the number of parts currently in the database. An if statement within the for loop is used to calculate the distance between the user and all the items in the spatial database, if the item is less than or equal to the threshold distance then the *strcpy()* function is implemented and copies name and category strings from the original item data structure to the found part data structure. The x and y coordinated are equated and the found part acquires all the required data for the specific spatial database item. The loop repeats if more items are found and the num_result also increases.
The result is then sent to the *print_page_sample()* function, to be displayed on the result html file.

- *find_nearest_neighbor()*



```c
void find_nearest_neighbor(){

 int num_result = 0;
 int i;
 int x;
 int y;
 int j;
 int n;
 int smallest;
 int d1 = 40*(sqrt(pow(items[i].x-x,2)+pow(items[i].y-y,2)));
 int A[512];
 Item found_part[num_parts];
 printf("This function will allow you to find the closest item to your location.\n");
 printf("Please enter your current location.\n");
 printf("Enter the x coordinate (max 32): \n");
 scanf("%d", &x);
  if(x > MAX_X_COORDINATE){

     printf("Error! Please input a valid position.\n");

 };

 printf("Enter the y coordinate (max 32): \n");
 scanf("%d", &y);
 if(y > MAX_X_COORDINATE){

     printf("Error! Please input a valid position.\n");
 };
 printf("Your position is currently:(%d,%d) \n", x,y);

  for(i=0;i<num_parts;i++){
  A[i] = 40*(sqrt(pow(items[i].x-x,2)+pow(items[i].y-y,2)));
     }
       smallest = A[0];
    for (i = 0; i < num_parts; i++) {
       if (A[i] < smallest) {
          smallest = A[i];
       }
    }
  for(i=0;i<num_parts;i++){

     if(40*sqrt(pow(items[i].x-x,2)+pow(items[i].y-y,2)) == smallest){
       strcpy(found_part[num_result].category,items[i].category);
       strcpy(found_part[num_result].name,items[i].name);
       found_part[num_result].x=items[i].x;
       found_part[num_result].y=items[i].y;
       num_result++;
     }
     }

 printf("\nNearest item is : %s.\n",found_part[0].name);
 print_page_sample(found_part, num_result);

 }
```

Figure 16: find nearest item function source code

A new data structure of item type named “found_part” is created; this data structure has an identical format to the item data structure. This data structure will store the found items. The function first asks the user to input his/her current coordinates.

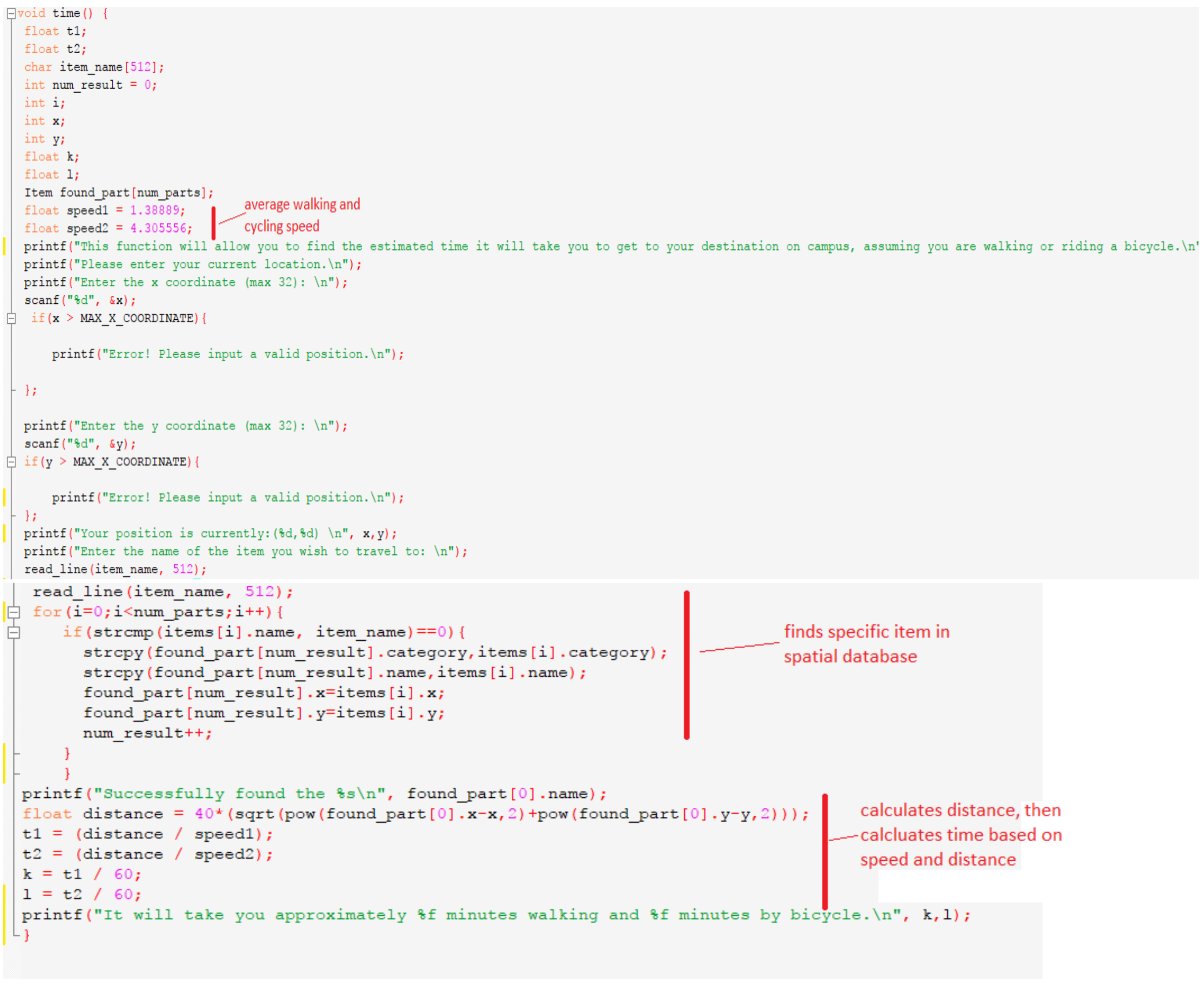
A for loop is implemented to repeat for the number of parts currently in the database. Within the for loop an array is created which stores the calculated distances between the user and all items in the spatial database. The smallest value in the array, which corresponds to the nearest data item, is chosen.

Once the smallest distance is chosen another for loop is implemented and essentially repeats the process found in the *find_items_by_range()* function but using the smallest distance as the threshold distance.

Repetition of *find_items_by_range()* function process:

A for loop is implemented to repeat for the number of parts currently in the database. An if statement within the for loop is used to calculate the distance between the user and all the items in the spatial database, if the item is less than or equal to the smallest distance then the *strcpy()* function is implemented and copies name and category strings from the original item data structure to the found part data structure. The x and y coordinated are equated and the found part acquires all the required data for the specific spatial database item. The loop repeats if more items are found and the num_result also increases.
The result is then sent to the *print_page_sample()* function, to be displayed on the result html file.

- *time()*



```c
void time() {
 float t1;
 float t2;
 char item_name[512];
 int num_result = 0;
 int i;
 int x;
 int y;
 float k;
 float l;
 Item found_part[num_parts];
 float speed1 = 1.38889;
 float speed2 = 4.305556;
 printf("This function will allow you to find the estimated time it will take you to get to your destination on campus, assuming you are walking or riding a bicycle.\n'
 printf("Please enter your current location.\n");
 printf("Enter the x coordinate (max 32): \n");
 scanf("%d", &x);
  if(x > MAX_X_COORDINATE){

     printf("Error! Please input a valid position.\n");

 };

 printf("Enter the y coordinate (max 32): \n");
 scanf("%d", &y);
 if(y > MAX_X_COORDINATE){

     printf("Error! Please input a valid position.\n");
 };
 printf("Your position is currently:(%d,%d) \n", x,y);
 printf("Enter the name of the item you wish to travel to: \n");
 read_line(item_name, 512);
  read_line(item_name, 512);
  for(i=0;i<num_parts;i++){
      if(strcmp(items[i].name, item_name)==0){
         strcpy(found_part[num_result].category,items[i].category);
         strcpy(found_part[num_result].name,items[i].name);
         found_part[num_result].x=items[i].x;
         found_part[num_result].y=items[i].y;
         num_result++;
      }
    }
 printf("Successfully found the %s\n", found_part[0].name);
 float distance = 40*(sqrt(pow(found_part[0].x-x,2)+pow(found_part[0].y-y,2)));
 t1 = (distance / speed1);
 t2 = (distance / speed2);
 k = t1 / 60;
 l = t2 / 60;
 printf("It will take you approximately %f minutes walking and %f minutes by bicycle.\n", k,l);
}
```

average walking and cycling speed
finds specific item in spatial database
calculates distance, then calcluates time based on speed and distance

Figure 17: Estimated time function source code

A new data structure of item type named “found_part” is created; this data structure has an identical format to the item data structure. This data structure will store the found items. The function first asks the user to input his/her current coordinates and the name of the spatial database item.

The *read_line()* function extracts the input name and stores it in the item_name array. A for loop is then implemented to repeat for the number of parts currently in the database. An if statement within the for loop uses the *strcmp()* function to compare the name of the item name to any of the names listed in the database. If the name matches then the result of the function is 0. If the name matches then the *strcpy()* function is implemented and copies name and category strings from the original item data structure to the found part data structure. The x and y coordinated are equated and the found part acquires all the required data for the specific spatial database item. The loop repeats if more items are found and the num_result also increases.

The distance between the user's coordinates and chosen item's position is then calculated. The time is easily calculated using the average walking speed and average bicycle speed, the formula is shown below:

$$t = \frac{x}{v}$$

- *Discussion:*

The inventory database program proved to be very helpful in acting as a foundation to refer to when developing the spatial database. Eventually the spatial database program was able to satisfy all the required functions for the project, some difficulties were encountered along the way and are listed below:

1. Loading items from the data file proved to be difficult in the early stages of the project. The commas and different variable types made extracting the data difficult, a new format had to be used to ignore the commas and scan all the data in each line.
2. The print page function was difficult to understand initially and required further research to understand how to manipulate an html file

- Interesting point during development of program:
  The calculation behind the nearest neighbor function was interesting to undertake. By using the knowledge of the previous function [*find_items_by_range()*] threshold, it was possible to incorporate the previous algorithm and find the nearest item in a creative way. This was done by calculating all the distances, choosing the shortest one and using that distance as the threshold.

- Future improvements
  The spatial database program could be improved in the following ways:
- Addition of photographs of the buildings in the html file. For example, if the user asks for the STARBUCKS COFFEE cafe, the program can display a small picture of the campus STARBUCKS COFFEE café and more information about the items available for purchase. This could make it easier for users to identify the buildings and make more informative choices.
- The time function could be improved by adding hours, minutes and seconds approximations. A countdown feature could be added so the user knows approximately how much time is left in his/her journey towards the spatial database item.

- *Conclusion:*

The project was very interesting to take part in. Students could get good experience with spatial databases and thus learn about the best methods to create a database in an efficient manner. This could potentially spark interest in software engineering topics for the students. Knowledge about how data structures can be effectively implemented as well as putting the programming skills to use was particularly enjoyable.

References

[1] Computer Software II lecture notes 2017, Yousuke WATANABE