**psh010209@gc.gachon.ac.kr, 010-7661-7035**

**SANGHUI PARK (박상희)**

# **개발 과제 #1 :**

|  |  |
| --- | --- |
|  | **Get Multiple Players and Search in those players via user-defined search queries.** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Save utilizing linked list, sort via back number: get search queries and search in the list**

* **Step 2: Outline a solution – 간단한 logic설명**

**For every single nodes, check it matches, then print.**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Get input from user about player information and save it into linked list by back number. By comparing this node’s back number and existing node’s back number. When all the inputs were processed, get searchQuery and for every nodes, check something that matches with the search query.**

* **Step 4: write a pseudo code – 수도코드 작성**

**getHowManyNodes**

**for howManyNodes:**

**allocate heap memory and handle exception**

**get player info**

**save to struct.**

**While endofLinkedList:**

**currentLinkedListNode.backNo > playerInfo.backNo  
 break**

**Insert this player’s node before currentLinkedListNode.**

**Get SearchQuery**

**While endofLinkedList:**

**If matchWithSearchQuery(currentLinkedListNode):**

**printLinkedListNode(currentLinkedListNode)**

**free heap memory**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <string.h>  *// struct for storing soccerplayer info*  typedef struct SoccerPlayer {  int backNumber;  char name[20];  int age;  int aMatches;  int goals;  } SoccerPlayer;  *// struct for linked node*  typedef struct PlayerNode {  struct SoccerPlayer soccerPlayer;  struct PlayerNode \*next;  } PlayerNode;  int main() {    *// get how many soccer players available for input*  int howMany;  printf("how many soccer players? : ");  scanf("%d", &howMany);  *// start of linked list head and nodes*  PlayerNode \*playerHead = NULL;  PlayerNode \*currentNode, \*prevNode = NULL;  *// read and write to linkedlist*  *for* (int i = 0; i < howMany; i++) {  *// initialize currentNode*  currentNode = playerHead;  *// tmp variable for storing player data.*  int backNo, age, aMatches, goals;  char name[21];  *// get player data from user*  printf("%d. \n", i);  printf("\tBackNo. ");  scanf("%d", &backNo);  fflush(stdin);  printf("\tName. ");  fgets(name, 21, stdin);  printf("\tAge. ");  scanf("%d", &age);  printf("\tGoals. ");  scanf("%d", &goals);  printf("\tA-matches. ");  scanf("%d", &aMatches);  *// if line return is found, remove it.*  *for* (int i = 0; i < 21; i++) *if* (name[i] == '\n') name[i] = 0;  *// allocate heap memory*  PlayerNode \*newNode = (PlayerNode \*)malloc(sizeof(PlayerNode));  *// exception*  *if* (newNode == NULL) {  printf("[ERROR] Memory Allocation Failed\nIs Memory full?\n");  *return* 1;  }  *// save player data to struct.*  newNode->soccerPlayer.backNumber = backNo;  newNode->soccerPlayer.aMatches = aMatches;  newNode->soccerPlayer.goals = goals;  strcpy(newNode->soccerPlayer.name, name);  newNode->soccerPlayer.age = age;  *// insert by sort*  *while* (currentNode != NULL) {  *if* (currentNode->soccerPlayer.backNumber > newNode->soccerPlayer.backNumber) {  *break*;  }  prevNode = currentNode;  currentNode = currentNode->next;  }    *// check whether head is null*  *if* (prevNode == NULL) {  *// if null, update playerhead to newNode.*  newNode->next = currentNode;  playerHead = newNode;  } *else* {  *// else just add newNode.*  prevNode->next = newNode;  newNode->next = currentNode;  }  }  printf("\n");  *// get search query*  int minAge;  int maxAge;  int minGoals;  int maxGoals;  int minAmatches;  int maxAmatches;  printf("Minimum Age to Search : ");  scanf("%d", &minAge);  printf("Maximum Age to Search : ");  scanf("%d", &maxAge);    printf("Minimum Goals to Search : ");  scanf("%d", &minGoals);  printf("Maximum Goals to Search : ");  scanf("%d", &maxGoals);  printf("Minimum A Matches to Search : ");  scanf("%d", &minAmatches);  printf("Maximum A Matches to Search : ");  scanf("%d", &maxAmatches);  printf("\n");  *// run search*  printf("=========[ %12s ]=========\n", "Soccer Players");  printf(" %10s %2s %19s %3s %4s %4s\n", "Types ", "No", "Name", "Age", "Goal", "AMtch");  *// for all linked List nodes*  *for* (currentNode = playerHead; currentNode != NULL; currentNode = currentNode->next) {  SoccerPlayer \*player = &currentNode->soccerPlayer;  *// check it matches with the data.*  *if* (minAge <= player->age && player->age <= maxAge && minGoals <= player->goals && player->goals <= maxGoals && minAmatches <= player->aMatches && player->aMatches <= maxAmatches) {  printf("%10s: %02d. %19s %3d %4d %5d\n", "Result", player->backNumber, player->name, player->age, player->goals, player->aMatches);  }  }  *// free all*  *for* (currentNode = playerHead; currentNode != NULL;) {  PlayerNode \*tmp = currentNode;  currentNode = currentNode->next;  free(tmp);  }  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **검색 예제** |
|  | |

|  |  |
| --- | --- |
| **#2** | **Sorting 예제** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

# **개발 과제 #2 :**

|  |  |
| --- | --- |
|  | **Find user-defined substring and return how many matches are available** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Find string in the specific string that user provided**

* **Step 2: Outline a solution – 간단한 logic설명**

**For each index for string search for specific string that user provided**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Get user input via argument of the program and find**

* **Step 4: write a pseudo code – 수도코드 작성**

**Get user input as a program argument.**

**Count = 0**

**For all indexes:**

**Check current location has substring of provided searchQuery**

**If found: count++**

**Print**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <string.h>  *#include* <stdlib.h>  *#define* true 1  *#define* false 0  typedef char \*string;  *// Find keyword from searchStr*  int rawSearch(string keyword, string searchStr) {  *// for length of the search String*  *for* (int i = 0; i < strlen(searchStr); i++) {  int j = 0;  *// check current index is matched with keyword*  *for* (j = 0; j < strlen(keyword) && searchStr[i+j] != 0; j++) {  *// if else*  *if* (keyword[j] != searchStr[i+j]) {  *// break*  *break*;  }  }  *// if it is a keyword*  *if* (j == strlen(keyword)) {  *// return base location of the keyword Found*  *// in the search String.*  *return* i;  }  }  *// if it is not found, return -1.*  *return* -1;  }  *// core searching algorithm*  int search(string searchQuery, string searchStr) {  *// char pointer for current searching location*  string currentSearch;  int count = 0;    *// for all index in currentSearch*  *for* (currentSearch = searchStr; \*(currentSearch) != 0; ) {  int len = strlen(searchQuery);  *// run search*  int searchLoc = rawSearch(searchQuery, currentSearch);  *// if it is found*  *if* (searchLoc >= 0) {  *//printf("Found: %d. %s @ Location. %d, CurrentSearch: %s\n", count, searchQuery, searchLoc, currentSearch);*  *//printf("Length: %d\n", len);*  *// increase count.*  count++;  currentSearch = currentSearch + searchLoc + len;  } *else* {  *// else increase currentSearch pointer for search in the next index.*  currentSearch++;  }  }  *//printf("\n");*  *// return count.*  *return* count;  }  int main(int argCount, char \*argData[]) {  *// if the argument is less than 3,*  *if* (argCount < 3) {  *// return usage*  printf("Usage:\n %s {keyword} {searchStr}\n", argData[0]);  } *else* {  *// else return how many matches are found.*  printf("%d matches!\n", search(argData[1], argData[2]));  }  } |
|  |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **다중 단어 감지** |
|  | |

|  |  |
| --- | --- |
| **#2** | **단어 없는 경우 감지** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**

# **개발 과제 #3 :**

|  |  |
| --- | --- |
|  | **Find user-defined searchQuery that contains wildcard and return is match is found or not** |

**Requirements**

* **Step 1: Understand the problem - 간략한 핵심 요구 조건/작업 설명/예시 작성**

**Search via searchString in the specific string that user provided**

* **Step 2: Outline a solution – 간단한 logic설명**

**Create char\*\* and store char\* s that Splited by wildcard symbol**

**For each index of splited searchString, check string contains that splited searchString, if found, check next splited searchString in the string after the found string in string program is searching in**

* **Step 3: Form a program structure – 프로그램 전체 구조 요약**

**Get user input via argument of the program and split searchQuery, for each splited searchString, check splited searchString is included in the currentString, which is copy of the string program need to search on. If found, set searchString after the found search keyword.**

* **Step 4: write a pseudo code – 수도코드 작성**

**Get user input as a program argument. To searchQuery, searchString**

**Split searchQuery by ‘\*’,**

**Pointer currentString = searchString**

**Foreach splitedString:**

**If currentIndexSplitedString is found in currentString**

**currentString += length(currentIndexSplitedString) + foundLocationInCurrentString  
 else  
 currentString++**

**if all splitedString was cycled:**

**print match found**

**else**

**match not found**

## **SOURCE CODE with comments**

|  |
| --- |
| *#include* <stdio.h>  *#include* <string.h>  *#include* <stdlib.h>  *#define* true 1  *#define* false 0  typedef char \*string;  *// the function that searches for specific keyword. returns keyword's location at the searchStr.*  int rawSearch(string keyword, string searchStr) {  *// for all index of searchstr.*  *for* (int i = 0; i < strlen(searchStr); i++) {  int j = 0;  *// check for keyword.*  *for* (j = 0; j < strlen(keyword) && searchStr[i+j] != 0; j++) {  *if* (keyword[j] != searchStr[i+j]) {  *// if it is not keyword. break.*  *break*;  }  }  *// if j reached to the keyword length. all of the 0th-to (keywordlength)-th char was same.*  *// meaning found the keyword.*  *if* (j == strlen(keyword)) {  *// return current index.*  *return* i;  }  }  *return* -1;  }  *// check how mnany \* in the searchQuery.*  int wildcardCounter(string searchQuery) {  int wildcardCount = 0;  *for* (int i = 0; i < strlen(searchQuery); i++) {  *if* (searchQuery[i] == '\*') {  wildcardCount++;  }  }  *return* wildcardCount;  }  *// gets wildcard contained searchQuery and split by \**  string \*searchQueryParser(string searchQuery) {    *// get how many wildcard are there*  int wildcardCount = wildcardCounter(searchQuery);  *// allocate for each string's array.*  string \*wildcardParsed = (string \*)malloc(sizeof(string) \* (wildcardCount + 1));  *// create pointer currentString and position at beginning of the search Query*  string currentString = searchQuery;  *// looping to wildcardCount +1 because we are spliting by \*, so we need to add 1 more index than how many \*'s in searchQuery.*  *for* (int i = 0; i < wildcardCount+1; i++ ) {  *// allocate memory for parsed strings.*  wildcardParsed[i] = (string)malloc((strlen(currentString) + 1) \* sizeof(char));  *//printf("\t%d. %s\n", i, currentString);*  *//copy until \* is found.*  int j;  *for* (j = 0; currentString[j] != 0 && currentString[j] != '\*'; j++) {  wildcardParsed[i][j] = currentString[j];  }  *// update currentString to 1 index later of the \*.*  currentString = currentString + j + 1;  }  *//printf("\n");*  *// return the entire parsed array.*  *return* wildcardParsed;  }  *// the main search*  int search(string searchQuery, string searchStr) {  *// check how many wildcards are out there.*  int searchQueryCount = wildcardCounter(searchQuery);  *//printf("%d\t",searchQueryCount);*    *// get parsed search Queries.*  string \*searchQueries = searchQueryParser(searchQuery);  *// set currentString as beginning of searchStr.*  string currentString = searchStr;    *// for all search Queries.*  *for* (int i = 0; i < searchQueryCount + 1; i++) {  int len = strlen(searchQueries[i]);  *// run a search.*  int searchLoc = rawSearch(searchQueries[i], currentString);  *// printf("%s\t", currentString);*  *// if found, go forward to end of the found word.*  *if* (searchLoc >= 0) {  currentString = currentString + searchLoc + len;  } *else* {  *// else return false.*  *return* false;  }  }  *// if all of them are found, return true.*  *return* true;  }  *// free all the memory for parsed searchQueries.*  void freeAll(string \*arr, int length) {  *for* (int i = 0; i < length; i++) {  free(arr[i]);  }  free(arr);  }  int main(int argCount, char \*argData[]) {  *if* (argCount < 2) {  printf("Usage:\n %s {keyword} {searchStr}\n", argData[0]);  } */\* else if (argCount < 3) {*  *int wildcardCount = wildcardCounter(argData[1]);*  *printf("Wildcard Parsing:\n");*  *printf("\tWildcard Count: %d\n\n", wildcardCounter(argData[1]));*  *string \*wildcardParsed = searchQueryParser(argData[1]);*  *for (int i = 0; i < wildcardCount + 1; i++) {*  *printf("\t%d. ", i);*  *printf("%s\n", wildcardParsed[i]);*  *}*  *freeAll(wildcardParsed, wildcardCount + 1);*  *} \*/* *else* {  printf("%s\n", search(argData[1], argData[2]) ? "match found!" : "match not found...");  }  } |

## **OUTPUT (Screen Shots)**

|  |  |
| --- | --- |
| **#1** | **와일드 카드 검색** |
|  | |

|  |  |
| --- | --- |
| **#2** | **와일드 카드 Edge Case 감지** |
|  | |

**//(필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)**