APS 105: Computer Fundamentals

Tutorial #5
Summer 2025

Problem 1: Last String in String (Winter 2022 Final Exam, Q10)

Write a function called `lastStringInString`, the prototype of which is provided below, that returns the pointer to the last occurrence of the string `s1` in the string `s2`. If the string `s1` cannot be found in the string `s2`, the function returns `NULL`. For example, if we are looking for the string "is" as `s1` in the string "This is a sample string" as `s2`, the pointer to the second "is" in the string `s2` will be returned by the function. Another example, if we are looking for the string "the" as `s1` in the string `s2` "The apple", the function should return `NULL`. This is because 't' is lower case in "the".

Note: You can use any function from the library `string.h`, except for `strstr()` you are not allowed to use it.

Function Prototype:

```
char *lastStringInString(char *s1, char *s2)
```

Example solution:

```
char * lastStringInString ( char * s1 , char * s2) {
  char * p = s2;
  char * lastFound = NULL ;
  // Alternatively , we can write the while loop as:
  // while ( endOfString - p >= strlen (s2 ))
  // while ( strlen (p) - str (s1) >= 0)
  while (*p != '\0 ') {
    if ( strncmp (s1 , p, strlen (s1 )) == 0) {
    lastFound = p;
    }
    p++;
  }
  return lastFound ;
}
```

Problem 2:Declare and Initialize Struct (Winter 2024 Final Exam, Q2)

Given the following defined data structure, write a single C statement terminated by one ;, that declares a variable named $p1$ of the type of the following data structure. In the same statement, initialize the variable such that the value of x is 1,the value of y is 2, and the value of z is 3.

```
typedef struct point {
  int z;
  int y;
  int x;
}
```

Example solution:

```
p1={3, 2, 1}
```

Problem 3: Fixing Memory Issues (Winter 2024 Final Exam, Q5)

The C program given below does not produce the expected output. You will be asked to fix the code in two specific parts, (a) and (b), as described below. The expected output is:

```
Enter last name: Patel
Enter last name: Wang
Patel
Wang
```

The program takes in two names from the user and saves them in the `lastName` member in the array of data structures students. However, the program does not produce the expected output. In the code below:

1. In (a) replace the line `students[i].lastName = str;` with code that results in the expected output. You may assume the maximum string size is 20 and the `stdlib.h` is included.
2. In (b) dynamically deallocate using free function any dynamically allocated data in the code.

The C program is:

```
typedef struct student {
  char* lastName;
} Student;

void getNames(Student students[]) {
```

```
  char str [20];
  for (int i = 0; i < 2; i++) {
    printf("Enter last name: ");
    scanf("%s", str);
    // TO BE COMPLETED
    // (a): Correct the following line
  students[i].lastName = str;
  }
}

int main(void) {
  Student* students = (Student*)malloc(2 * sizeof(Student));
  getNames(students);
  for (int i = 0; i < 2; i++) {
    printf("%s\n", students[i].lastName);
  }
  // TO BE COMPLETED
  // (b): Deallocate dynamically allocated memory
}
```

Example solution:

```
// a
students[i].lastName = (char*) malloc(sizeof(char) * 20);
strcpy(students[i].lastName , str);

// b
for(int i = 0; i < 2; i++){
  free(students[i].lastName);
}
free(students);
```

Problem 4: Reomve the Tail Node (Winter 2024 Final, Q7)

The following code intends to remove the tail node in a linked list. Complete the removeTail function such that the last node is removed, properly freed and no segmentation fault is caused. The data structure definition of LinkedList and Node are shown below.

```
typedef struct node {
  int data;
  struct node* next;
} Node;

typedef struct linkedList {
  Node* head;
} LinkedList;

void removeTail(LinkedList* list) {
  Node* current = list ->head;
  Node* prev = NULL;
  if (list ->head == NULL) {
    return;
```

```
  }
  while (current ->next != NULL) {
    prev = current;
    current = current ->next;
  }
}
```

Example solution:

```
void removeTail(LinkedList* list) {
  Node* current = list ->head;
  Node* prev = NULL;
  if (list ->head == NULL) {
    return;
  }
  while (current ->next != NULL) {
    prev = current;
    current = current ->next;
  }

  // Solution:
  if (list ->head ->next !=NULL){
    free(current);
    prev ->next = NULL;
  } else {
    free(current);
    list ->head = NULL;
  }
}
```

Problem 5: Linkedlists Iscycle (Winter 2024 Final, Q11)

The Node structure in a linked list has been defined as follows:

```
typedef struct node {
  int data;
  struct node *next;
} Node;
```

The LinkedList structure has also been defined to contain the head of a linked list:

```
typedef struct linkedList {
   Node *head;
} LinkedList;
```

We have learned in class that the tail node in the linked list has its next pointer point to `NULL`. In a linked list with a cycle, the tail node next points to the head node in the linked list. Write a C function named `isCycle`, the prototype of which is given below, that takes `LinkedList*` list as an argument and returns true if there exists a cycle, otherwise, it returns false.

Example solution:

```
bool isCycle(LinkedList* list) {
  Node* current = list ->head;
  if (current != NULL) { // if there is a next node, look at it
    current = current ->next;
}
  while (current != NULL && current != list->head) {
    current = current ->next;
  }
  return (current == list->head);
}
```

Problem 6: Doubly Linked Lists (Winter 2025 Final Exam, Q32)

Another category of linked lists called **doubly linked lists** has each node store `data`, which is an integer value, `prev`, which is a pointer to the previous node and `next`, which is a pointer to the next node. This makes it easier to traverse the list in both directions.

The definition of a node in a doubly linked list is provided below:

Function Prototype:

```
// Struct for a node in the doubly linked list
typedef struct node{
  int data;
  struct node* prev;
  struct node* next;
} Node;

// Struct for the linked list
typedef struct{
  Node* head;
  Node* tail;
} DoublyLinkedList;
```

You are required to implement the function `insertAfterNode`, the prototype of which is shown below, that inserts a new node **after** a given node in a doubly linked list as shown in the figure below. `list` is a pointer to the doubly linked list, `prevNode` is a pointer to node after which you'd need to add a new node, and `val` is the value of the data member of the new node that you're required to dynamically allocate and insert in the list. You can assume `prevNode` will be pointing to a valid node.

Note that in a doubly linked list, we are required to also keep track of the last node in the list, also known as the tail.

```
  void insertAfterNode{DoublyLinkedList* list, Node* prevNode, int val};
```

Example solution:

```
void insertAfterNode(DoublyLinkedList *list, Node *prevNode, int val)
{
```

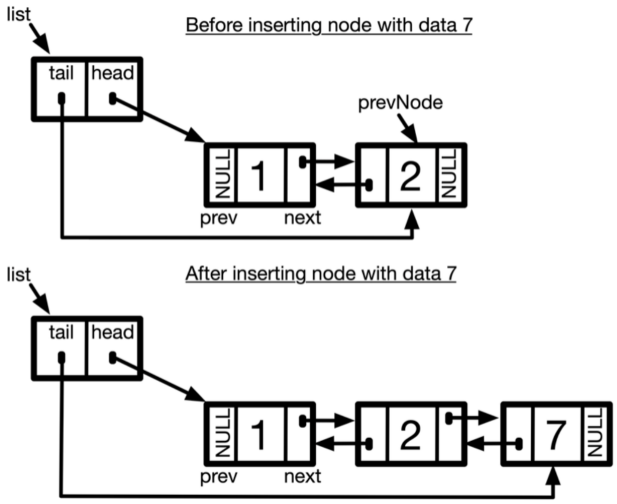

Figure 1: Examples of the doubly linked lists.

```
    // Create the new node
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = val;
    newNode->prev = prevNode;

    // If newNode is not the last node, update the prev of the next node
    if (prevNode->next != NULL)
    {
        newNode->next = prevNode->next;
        prevNode->next = newNode;
        newNode->next->prev = newNode;
    }
    else
    {
        prevNode->next = newNode;
        newNode->next = NULL;
        // If newNode is the last node, update the tail pointer
        list->tail = newNode;
    }
}
```