2.1,2.2,2.3,2.4)

```
struct list_node {
  unsigned long time;
  int light;
  int temp;
  struct list_node *next;
};

struct list_node *head = NULL;
unsigned long time = 0;
int pin_light = A3;
int pin_temp = A8;
int count = 0;

void setup(){

}

void loop(){
  if(delta_t() >= 1000){
     time = millis();

     struct list_node *n;
     n = (struct list_node *) malloc(sizeof(struct list_node));
     n->time=millis();

     int reading = analogRead(pin_light);
     n->light=map(reading, 0, 410, -50, 150);

     reading=analogRead(pin_temp);
     n->temp=map(reading, 0, 1023, 0, 100);

     n->next = head;
     head=n;
   }
}

unsigned long delta_t(){
  unsigned long ret = millis()-time;
  return ret;
}
```

als Input definieren 0,5P

5,5/6

2.5)

```
struct list_node {
  unsigned long time;
  int light;
  int temp;
  struct list_node *next;
};
```

```
struct list_node *head = NULL;
unsigned long time = 0;
int pin_light = A3;
int pin_temp = A8;
int count = 0;

void setup(){

}

void loop(){
  if(delta_t() >= 1000){
     if(count==100){
      remove_last();
      count=count-1;
     }
     time = millis();

     struct list_node *n;
     n = (struct list_node *) malloc(sizeof(struct list_node));
     n->time=millis();

     int reading = analogRead(pin_light);
     n->light=map(reading, 0, 410, -50, 150);

     reading=analogRead(pin_temp);
     n->temp=map(reading, 0, 1023, 0, 100);

     n->next = head;
     head=n;
     count=count+1;
   }
}

void remove_last(){
  struct list_node *n;
  n = (struct list_node *) malloc(sizeof(struct list_node));

  struct list_node *m;
  m = (struct list_node *) malloc(sizeof(struct list_node));
  n = head;
  m = n->next;
  while(m->next != NULL){
    n = m;
    m = m->next;
  }
  n->next = NULL;
  free(m);
}
unsigned long delta_t(){
  unsigned long ret = millis()-time;
  return ret;
```

+ 1 Bonus

```
}
```

2.6)

```
struct list_node {
  unsigned long time;
  int light;
  int temp;
  struct list_node *next;
  struct list_node *previous;
};

struct list_node *head = NULL;
unsigned long time = 0;
int pin_light = A3;
int pin_temp = A8;
int count = 0;

void setup(){

}

void loop(){
  if(delta_t() >= 1000){
     if(count==100){
      remove_last();
      count=count-1;
     }

     time = millis();

     struct list_node *n;
     n = (struct list_node *) malloc(sizeof(struct list_node));
     n->time=millis();

     int reading = analogRead(pin_light);
     n->light=map(reading, 0, 410, -50, 150);

     reading=analogRead(pin_temp);
     n->temp=map(reading, 0, 1023, 0, 100);

     if(head == NULL){
       n->next=n;
       n->previous=n;
       head=n;
     }else{
       n->next=head;
       head->previous->next=n;
       n->previous=head->previous;
       head->previous=n;
       head=n;
     }
```

```
    count=count+1;
   }
}

void remove_last(){
  struct list_node *n;
  n = (struct list_node *) malloc(sizeof(struct list_node));
  struct list_node *m;
  m = (struct list_node *) malloc(sizeof(struct list_node));

  n = head->previous;
  m = n->previous;

  head->previous=m;
  m->next=head;
  free(n);

}
unsigned long delta_t(){
  unsigned long ret = millis()-time;
  return ret;
}
```

+ 2 Bonus

```
3)
typedef struct Node{
  struct Node *left;
  struct Node *right;

  int value;
  char name[80];
  unsigned int semester;

}Node;

void insert_node(Node **root,Node* node);
Node *find_node(Node ** root,int x);
Node *two_leafs(Node *root);
void remove_node(Node*root,int element);
void print_simetrical_order(Node *root);
void print_pos_order(Node *root);
void print_pre_order(Node *root);

Node* create_node(int registration_number, int semester, String name);
char* student_name(Node *root, int registration_number);
void print_student(char* name);

Node *root = NULL;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("\n\n\nBEGIN");
```

```
  //insertions
  Node *n=create_node(100,1,"Dath Vader");
  insert_node(&root,n);
  n=create_node(50,1,"Voldemort");
  insert_node(&root,n);
  n=create_node(20,1,"Sauron");
  insert_node(&root,n);
  n=create_node(30,1,"Thanos");
  insert_node(&root,n);
  n=create_node(150,1,"Zod");
  insert_node(&root,n);
  n=create_node(15,1,"Magneto");
  insert_node(&root,n);
  n=create_node(100,1,"Dath Vader");//inserting a repetead student
  insert_node(&root,n);

  //finding a student
  int registration_number=50;
  print_student(student_name(root,registration_number));
  remove_node(&root,registration_number);//removing the same student
  print_student(student_name(root,registration_number));

  Serial.println("END");

}

void loop() {
  // put your main code here, to run repeatedly:

}

void print_student(char* name){
  if (name!=NULL){
    Serial.print("Student Found, his/her name is: ");
    Serial.println(name);
  }else{
    Serial.println("Student not Found");
  }
}

Node* create_node(int registration_number, int semester, String name){
    Node *node = (Node *)malloc(sizeof(Node));
    node->value = registration_number;
    node->semester=semester;
    name.toCharArray(node->name,80);
    node->right = node->left = NULL;
    return node;
}

void insert_node(Node **root,Node* node)
{
  if(*root == NULL) {
```

√ + 1Done

```
    *root = node;
    Serial.print("INSERTING ");
    Serial.print(node->name);
    Serial.print(" registration number: ");
    Serial.print(node->value);
    Serial.print(" and semester: ");
    Serial.println(node->semester);
    return;
  }

  if(node->value < (*root)->value) {
    insert_node(&(*root)->left,node);
    return;
  }
  if(node->value > (*root)->value) {
    insert_node(&(*root)->right,node);
    return;
  }
  Serial.print("the Student ");
  Serial.print(node->name);
  Serial.println(" already Exists");
}

Node *two_leafs(Node *root){
    if(root==NULL)
       return NULL;
    else if(root->left == NULL)
          return root;
    else
       return two_leafs(root->left);
}

void remove_node(Node**root,int element){
  if(element < (*root)->value){
    remove_node(&(*root)->left,element);
  }
  else if(element > (*root)->value){
    remove_node(&(*root)->right,element);
  }
  else if((*root)->left!=NULL && (*root)->right!=NULL){
    Node *aux= NULL;
    aux = two_leafs((*root)->right);
    (*root)->value = aux->value;
    remove_node(&(*root)->right,(*root)->value);
  }
  else {
    Node *aux = (*root);
    Serial.print("REMOVING ");
    Serial.print(aux->name);
    Serial.print(" registration number: ");
    Serial.print(aux->value);
```

```
      Serial.print(" and semester: ");
      Serial.println(aux->semester);
      if((*root)->left==NULL) {
         (*root) = (*root)->right;
      }
      else {
         *root = (*root)->left;
      }
      free(aux);

  }
}

void print_simetrical_order(Node *root){
  if(root == NULL)
      return;
  print_simetrical_order(root->left);
  Serial.println(root->value);
  print_simetrical_order(root->right);
}

void print_pos_order(Node *root){
  if(root == NULL)
      return;
  print_pos_order(root->left);
  print_pos_order(root->right);
  Serial.println(root->value);
}

void print_pre_order(Node *root){
  if(root == NULL)
      return;
  Serial.println(root->value);
  print_pre_order(root->left);
  print_pre_order(root->right);
}

Node *find_node(Node *root,int element){
  /*while ( root != NULL ) {
    if ( element < root->value ) {
      root = root->left ;
    }
    else if ( element > root->value ) {
      root = root->right ;
    }
    else { // element == root - > value ;
      return root ;
    }
  }
  return NULL;*/
  if (root!=NULL && element<root->value){
    return find_node(root->left,element);
```

```
  }else if(root!=NULL && element>root->value){
    return find_node(root->right,element);
  }else if(root!=NULL){
    return root;
  }else{
    return NULL;
  }

}

char *student_name(Node *root, int registration_number){
  Serial.print("FINDING student with");
  Serial.print(" registration number: ");
  Serial.println(registration_number);

  Node *node=find_node(root,registration_number);
  if(node!=NULL){
    return node->name;
  }else{
    return NULL;
  }

}
```

√ 6/6 + 2 Bonus

16,5/12