STUDENT

# DIT633-0010-LHB

TENTAMEN

# DIT633 aug re-exam 2025 - L

| | |
|---|---|
| Kurskod | -- |
| Bedömningsform | -- |
| Starttid | 29.08.2025 14:00 |
| Sluttid | 29.08.2025 18:00 |
| Bedömningsfrist | -- |
| PDF skapad | 26.11.2025 15:19 |
| Skapad av | Lisa Lindén |

## Instructions for the exam

# DIT 633 - Development of Embedded and Real-time systems

Welcome to this edtion of the DIT633 examination!

This exam should be an individual work for you. You are not allowed to use any outside help.

If you are allowed to use a compiler, there is a link to an online one, which will open in a separate window. You can test the code in the online compiler, but **you must remember to copy-paste it back to the exam**, otherwise your code will disappear once you close the window.

The same is true for TinkerCad, please remember to copy-paste the code from TinkerCad to the exam. Please note that the log-in to TinkerCad is different this time, so please read the instruction carefully (available in the TinkerCad question), especially regarding your **class nickname and class code:**

*Please use the following procedure to log in to TinkerCad:*
*- go to log-in*
*- choose "Students with Class Code"*
*- use the class code "*BVU-ISG-HFN*"*
*- use join with nickname. Your anonymous code is your nickname - without the dashes and using small letters, e.g., DIT633-0001-ABC becomes dit6330001abc as your nickname*

You are NOT allowed to access code that was not written during the exam (e.g., from your previous assignments). This will be considered plagiarism.

You are not allowed to copy code from your colleagues or any other external source.

**Remember: In programming questions, if the code does not compile, you get 0 points for the question!**

Grading scale:
50% correct - 3
65% correct - 4
85% correct - 5

Good luck!

/Miroslaw
031 772 1081

# 1 Sustainable programs

The following program code solve the problem of calculating Fibonacci numbers using a recursive approach.

---

```
#include <stdio.h>

// Recursive function to calculate Fibonacci numbers
int fibonacci(int n) {
    if (n == 0) {
        return 0; // Base case
    }
    else if (n == 1) {
        return 1; // Base case
    }
    else {
        return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case
    }
}
int main() {
    int n, i;
    printf("Which number should I calculate: ");
    scanf("%d", &n);
    printf("Fibonacci sequence up to %d your requested number:\n", n);
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    printf("\n");
    return 0;
}
```

---

Fibonacci numbers are calculated as follows:

1st: 0

2nd: 1

3rd: 2

4th: 3

nth: sum of fibonacci numbers n-1 and n-2

Your task is to rewrite this program so that it uses iteration instead.

In addition to the code, in the comment you must write why this new program is better for sustainability.

Grading:

1. Correct implementation: 2 points
2. Comments for the algorithm: 1 point
3. Explanation of the impact on sustainability: 2 points

**Skriv ditt svar här. Ändringar sparas automatiskt.**

```c
#include <stdio.h>

// this is the iterativ function to calc fibonacci nums
// this is better for sustainablity than recursive becuse:
// recursion uses extra function calls which increases cpu usage and energy consm
// -iteration uses a simpler loop and reuses variables (previous1 previous2) whic
// -efficiency needs less computational power which reduces energy bill and suppo
int fibonacci(int n) { //still base cases becus we dont need to calculate those
    if (n == 0) return 0;
    if (n == 1) return 1;

    int previous1 = 0, previous2 = 1, current, i; //declare and save prev nums an
    for (i = 2; i <= n; i++) {
        current = previous1 + previous2; // assign prev1 and prev2 to current
        previous1 = previous2; // change the first prev1 to the one before prev2
        previous2 = current; // prev2 is the highest (the value needed)
    }
    return previous2;
}

int main() { //same main as before
    int n, i;

    printf("Which number should I calculate: ");
    scanf("%d", &n);

    printf("Fibonacci sequence up to %d your requested number:\n", n);
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    printf("\n");

    return 0;
}
```

Besvarad.

## 2 Reading pointers

Please choose the right interpretation of "X" in each of the statements:

**int *x();**

- x is a variable of type int
- x is a pointer to a pointer to a function that returns a variable of type int
- x is a pointer to a function that returns an int
- x is a function that returns a pointer to an int ✓

**int ( * x [] ) ();**

- x is a pointer to a function that takes as an argument an array of integers
- x is an array of pointers to functions that return int ✓
- x is a function that takes an array as an argument and returns a pointer to int
- x is a function which takes as an argument an array of pointers to variables of type int

**char * ( * ( * x [ ] [8] ) ( ) ) [ ];**

- x is array of array of 8 pointers to a pointer to functions returning pointer to array of pointer to char
- x is array of array of 8 pointers to a function returning pointer to array of pointer to ch ✓
- x is an array of 8 pointers to pointer to function returning pointer to array of pointer to char
- x is a function that takes as input an array of 8 pointers to pointer to an array and returns a pointer to array of pointer to char

**int ( * ( * x ) [ ] ) ( );**

- x is a function that takes as argument an array of pointers to functions and return a pointer to int
- x is an array of pointers to functions that take no arguments and return pointers to int
- x is a function that takes as an argument a pointer to an array of pointers and returns a pointer to int
- x is a pointer to an array of pointers to functions returning an int ✔

**char ( * * x ( ) ) [20];**

- x is an array of pointers to functions returning pointers to functions returning pointers to char
- x is an array of 20 pointers to functions returning char
- x is a function returning a pointer to a pointer to an array of 20 elements of type char ✔
- x is a pointer to a function returning a pointer to an array of 20 elements of type char ✖

---

Delvis rätt. 4 av 5 poäng.

# 3 Spotify's dynamic list

Spotify wants to implement a **dynamic playlist** that resizes when songs are added. They started coding but didn’t finish.

---

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Structure for Playlist
typedef struct {
    char** songs;
    int size;
    int capacity;
} Playlist;
// Function to create a new playlist
Playlist* createPlaylist(int capacity) {
    // TODO: allocate memory for Playlist and songs
}
// Function to add a song (resize if needed)
void addSong(Playlist* pl, const char* song) {
    // TODO: reallocate if capacity is full
    // TODO: copy song into array
}
// Function to print all songs
void printPlaylist(Playlist* pl) {
    // TODO: print songs
}
// Function to free memory
void freePlaylist(Playlist* pl) {
    // TODO: free songs and playlist
}
int main() {
    Playlist* pl = createPlaylist(2);
    int choice;
    char song[100];
    while (1) {
        printf("\nSpotify Playlist:\n");
        printf("1. Add Song\n");
        printf("2. Show Playlist\n");
        printf("3. Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter song name: ");
                scanf("%s", song);
                addSong(pl, song);
                break;
            case 2:
                printPlaylist(pl);
                break;
            case 3:
                freePlaylist(pl);
```

```
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice.\n");
    }
  }
  return 0;
}
```

---

Your task is to finish the code. The song itself is just the string with the title of the song (you can make it into a struct if you want, but it is not necessary).

Grading:
1. createPlaylist correct: 3 points
2. addSong correct (with reallocation): 3 points
3. freePlaylist correct: 2 points
4. Comments: 2 points

**Skriv ditt svar här. Ändringar sparas automatiskt.**

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

// struct for playlist
typedef struct {
    char** songs;// pointer to an array of song titles (each is a string)
    int size;//num of songs currently in the playlist
    int capacity;//current maximum capacity before resizing
} Playlist;

 // create a new playlist with an initial capacity
Playlist* createPlaylist(int capacity) {
    Playlist* pl = (Playlist*) malloc(sizeof(Playlist));      //allocate memory
    pl->songs = (char**) malloc(capacity * sizeof(char*));  //allocate memory for
    pl->size = 0;       //init size and capacity
    pl->capacity = capacity;
    return pl; // return playist
}

// func to add a song to the playlist
void addSong(Playlist* pl, const char* song) {
    if (pl->size == pl->capacity) {       // check if playlist is full
        pl->capacity *= 2;    // double capacity to make room
        pl->songs = (char**) realloc(pl->songs, pl->capacity * sizeof(char*)); //
    }
    pl->songs[pl->size] = (char*) malloc((strlen(song) + 1) * sizeof(char)); //al
    strcpy(pl->songs[pl->size], song);  //copy song name into allocated memory
    pl->size++;  //increase the number of songs
}

//function to prrint all songs
void printPlaylist(Playlist* pl) {
    if (pl->size == 0) {   //check if playlist is empty
        printf("Playlist is empty.\n");   //print message if no songs
        return;   //exit the function
    }
    printf("Playlist:\n");   // print header
    for (int i = 0; i < pl->size; i++) {   //lop through  songs
        printf("%d. %s\n", i + 1, pl->songs[i]);   // print song index and name
    }
}
```

```
}

// function to free all allocated memory
void freePlaylist(Playlist* pl) {
    for (int i = 0; i < pl->size; i++) {  // loop through all songs
        free(pl->songs[i]);   // free each individual song string
    }
    free(pl->songs);// free songs array
    free(pl);  // free pl
}

// main function
int main() {
    Playlist* pl = createPlaylist(2); // create a playlist init capacity of 2
    int choice; // variable to store menu choice
    char song[100];  // buffer for entering a song name

    while (1) {   // infinite loop for menu
        printf("\nSpotify Playlist:\n"); // print menu header
        printf("1. Add Song\n");
        printf("2. Show Playlist\n");
        printf("3. Exit\n");
        scanf("%d", &choice);  // read user's choice

        switch (choice) {         // handle user choice
            case 1:
                printf("Enter song name (one word only): "); // ask user for song
                scanf("%s", song);  // read song name (up to first space)
                addSong(pl, song);  // add song to playlist
                break;
            case 2:
                printPlaylist(pl);      // print all songs
                break;
            case 3:
                freePlaylist(pl);  // free allocated memory
                printf("Exiting...\n");   // print exit message
                return 0;  // end program
            default:  // if invalid choice
                printf("Invalid choice.\n");  // print error
        }
    }
    return 0; // return success (never reached)
}
```

---

Besvarad.

# 4 Multithreaded robot race

Before the 2028 Olympic Games, the AI Committee decided to introduce multithreaded Robot Race, where each robot runs **in parallel**.

The rules of the game are:

- Each race has **four players**, and each race has **10 rounds**.
- In each round, a player randomly generates a **16-bit number**, where bits are interpreted as:
  - **Speed multiplier**: 4 bits
  - **Gear (forward = 1, reverse = 0)**: 1 bit
  - **Steps**: 11 bits
- Distance travelled = (steps × speed multiplier), and negative if in reverse gear.
- After 10 rounds, the robot with the longest total distance wins.

## Task

You are given a partially completed C program. Each robot must be implemented as a **separate thread**. The program must:

1. Create 4 threads (one for each robot).
2. Simulate 10 rounds per robot inside the threads.
3. Use bitwise operators to extract values from the 16-bit number.
4. After each round, **print the robot's status** (thread-safe):
   - Randomly drawn number
   - Player ID, speed multiplier, gear, number of steps
   - Distance travelled so far
5. At the end, print the **winner**.

You must use:

- **Bit shifting and masking** to extract values
- **pthread library** for threads
- **Mutex lock** for safe printing

--- skeleton code ---

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#define NUM_PLAYERS 4
#define ROUNDS 10
pthread_mutex_t lock;  // for synchronized printing
// Structure for player state
typedef struct {
    int id;
    int distance;
} Player;
// Thread function for each robot
void* run_robot(void* arg) {
    Player* p = (Player*) arg;
```

```
    unsigned short rnd; // 16-bit number
    int speed, gear, steps;
    for (int round = 0; round < ROUNDS; round++) {
        rnd = rand() % 65536; // generate random 16-bit number
        // TODO: extract fields using bit operations
        // player ID = bits 15-14
        // speed = bits 13-11
        // gear = bit 10
        // steps = bits 9-1 (LSB)
        // TODO: calculate travelled distance and update p->distance
        // TODO: lock mutex before printing
        pthread_mutex_lock(&lock);
        printf("Player %d, Round %d: Number=%u, ID=%d, Speed=%d, Gear=%d, Steps=%d, Total
Distance=%d\n",
                p->id, round+1, rnd, /*fill*/, /*fill*/, /*fill*/, /*fill*/, p->distance);
        pthread_mutex_unlock(&lock);
    }
    pthread_exit(NULL);
}
int main() {
    srand(time(NULL));
    pthread_t threads[NUM_PLAYERS];
    Player players[NUM_PLAYERS];
    pthread_mutex_init(&lock, NULL);
    // TODO: initialize players and create threads
    // pthread_create(...)
    // TODO: wait for all threads to finish
    // pthread_join(...)
    // TODO: determine and print the winner
    pthread_mutex_destroy(&lock);
    return 0;
}
```

---

Grading:

1. Correct use of bit operations: 3 points
2. Correct implementation of multithreading (pthread create/join): 3 points
3. Correct and thread-safe status printing: 2 points
4. Comments on new code: 2 points

**Skriv ditt svar här. Ändringar sparas automatiskt.**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <time.h>
5
6  #define NUM_PLAYERS 4
7  #define ROUNDS 10
8
9  pthread_mutex_t lock;  // for synchronized printing
10
11 // struct for player state
12 typedef struct {
```

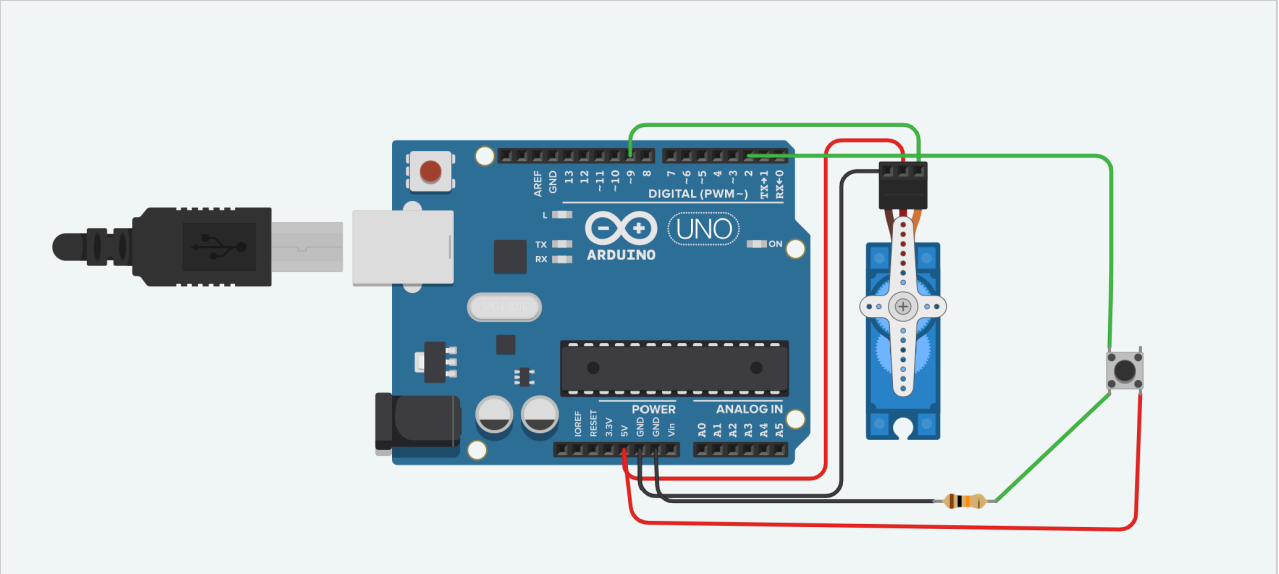
```
    int id;
    int distance;
} Player;

// Thread function for each robot
void* run_robot(void* arg) {
    Player* p = (Player*) arg;
    unsigned short rnd; // 16-bit number
    int speed, gear, steps;
    int rnd_id;
    int delta;
    for (int round = 0; round < ROUNDS; round++) {
        rnd = rand() % 65536; // generate random 16-bit
        // decode using bit shifts/ masks
        rnd_id =  (rnd >> 14) & 0x3; // extract bits 15to14 (2 bits)
        speed  =  (rnd >> 11) & 0x7;  // extract bits 13to11 (3 bits)
        gear   =  (rnd >> 10) & 0x1; // bit 10 (1 bit) 1 forward and 0 revers
        steps  =   rnd         & 0x3FF; //bits 9to0 (10 bits )

        // calc signed distance this round == steps * speed, if gear indicates re
        delta = steps * speed; // multiply steps by speed multiplier
        if (gear == 0) delta = -delta; // if in reverse (gear==0) the delta is ne
        p->distance += delta; //all distance into total

        /*
        p->distance += delta;
        */

        pthread_mutex_lock(&lock);
        printf("Player %d, Round %d: Number=%u, ID=%d, Speed=%d, Gear=%d, Steps=%
               p->id, round+1, (unsigned)rnd, rnd_id, speed, gear, steps, p->dist
        pthread_mutex_unlock(&lock);
    }
    pthread_exit(NULL);
}

int main() {// Program entry point
    srand((unsigned)time(NULL));// seed once (in all threds)
    pthread_t threads[NUM_PLAYERS];
    Player players[NUM_PLAYERS];

    pthread_mutex_init(&lock, NULL);// Initialize mutex with defaults

    //create and start one thread per player
    for (int i = 0; i < NUM_PLAYERS; i++) {//initialize and launch threads
        players[i].id = i + 1;// assign human-readable ids
        players[i].distance = 0;//initialize
        int rc = pthread_create(&threads[i], NULL, run_robot, &players[i]);
        if (rc != 0) {
            fprintf(stderr, "pthread_create failed for player %d (rc=%d)\n", i +
            exit(1);// exit
        }
    }

    // Wait for all player threads to finish
    for (int i = 0; i < NUM_PLAYERS; i++) {// join each thread to complete it
        int rc = pthread_join(threads[i], NULL);// wait to terminate
        if (rc != 0) {
            fprintf(stderr, "pthread_join failed for player %d (rc=%d)\n", i + 1,
            exit(1);// exit
        }
    }

    // Determine the winner by maximum total distance
    int winner_idx = 0;
    for (int i = 1; i < NUM_PLAYERS; i++) {
        if (players[i].distance > players[winner_idx].distance) {
            winner_idx = i;// Update index
        }
    }

    // print the final winner summary
    printf("\nWinner: Player %d with total distance %d\n",
           players[winner_idx].id,// readable ID of the winner
```

```
87              players[winner_idx].distance);//final distance
88
89      pthread_mutex_destroy(&lock);
90      return 0;// exit
91  }
```

---

Besvarad.

# 5 Arduino counter




In this task, you should design a simple counter.

In the circuit, you see that there is a button and a servo. Every time we press a button, the servo should move. Once it gets into 180 degrees, it should reset itself.

You should use interrupts for the button.
You should write the counter on the serial output.

Grading:
1. Implementing the button: 1 point
2. Using interrupt on the button: 1 point
3. Moving the servo: 1 point
4. Resetting the servo: 1 point
5. Writing to serial: 1 point
6. Comments: 1 point

*Please use the following procedure to log in to TinkerCad:*
*- go to log-in*
*- choose "Students with Class Code"*
*- use the class code "BVU-ISG-HFN"*
*- use join with nickname. Your anonymous code is your nickname - without the dashes and using small letters, e.g., DIT633-0001-ABC becomes dit6330001abc as your nickname*

**Skriv ditt svar här. Ändringar sparas automatiskt.**

```
1
```

Obesvarad.

# 6 Validation

During our visit to Volvo, they asked as for help to design their validation mechanism.

Pre-requisite: Each card have two sides. One side is a letter and the other is a digit.

Their requirement: If the letter is A then the digit must be 3, if the letter is B then the digit must be 4

Their task: check if the cards K-5 and A-3 fulfill the requirement.

Every time the requirement changes, they need to re-write the program for validation of cards. We can do better than that.

Your task
Write a program that takes the requirements as command line arguments, then randomly generates 10 cards, checks if the cards fulfill the requirements. Then it asks the user for one card and checks if it fulfills the requirement. The program should be able to take 1-10 requirements as input in the command line.

**For example**
Input (command line): *main.exe A-3 B-2 D-0*

Output:
*Requirements:*
*A-3*
*B-2*
*D-0*
---
*generating: A-3 -- OK*
*generating: C-0 -- OK*
*generating: D-2 -- not OK*
*...*
---
*Please input a card: <D-0>*
*Card <D-0> -- OK*

Grading:
1. Correct funtionality - 4 points
2. Using pointers and dynamic arrays - 2 points
3. Commenting the code - 2 points
4. Fail-safety - 2 points

You can use the online compiler in this question here

**Skriv ditt svar här. Ändringar sparas automatiskt.**

```
1
```

Obesvarad.