# System and Device Programming
# Standard Exam
# 25.06.2024

## Ex 1 (1.25 points)

Suppose to run the following program. Indicate the possible admissible outputs. Note that wrong answers imply a penalty in the final score.

```
std::map<int, int> f (const std::vector<int>& vec, std::set<int> s) {
    std::map<int, int> m;
    for (const int &e : vec) {
      auto it = s.find(e);
      if (it != s.end())
        ++m[e];
    }
    return m;
}

int main() {
  std::vector<int> vec = {1, 2, 6, 2, 3, 3, 2, 3, 4, 5, 4, 4, 4, 5, 5, 5, 6, 6};
  std::set<int> s = {2, 4, 6};
  std::map<int, int> m = f(vec, s);
  for (const auto& pair : m) {
    std::cout << pair.first << ":" << pair.second << "-";
  }
  return 0;
}
```

Choose one or more options.

1. ⬜ 1:1-2:3-3:3:4:4-5:4-6:3-
2. ✅ 2:3-4:4-6:3-
3. ⬜ 2:2-4:4-6:6-
4. ⬜ 3:2-4:4-3:6-
5. ⬜ 1:1-2:2-6:3-2:4-3:5-3:6-2:7-3:8-4:9-5:10-4:11-4:12-4:13-5:14-5:15-5:16-6:17-6:18-
6. ⬜ 1:1-2:3-3:3-4:4-5:4-6:3-

## Ex 2 (1.25 points)

Analyze the following code snippet in C++. Indicate the possible output or outputs obtained by executing the program.

```
int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};
    auto lf = [&vec](long unsigned int index) -> int {
        if (index >= 0 && index < vec.size()) {
            int value = vec[index];
            return value;
        }
        std::cout << "Out" << " ";
        return -1;
    };
    std::cout << lf(2) << " ";
    std::cout << lf(5) << " ";
    std::cout << lf(0) << " ";
    return 0;
}
```

Choose one or more options:

1. ⬜ 2 Out -1 0
2. ⬜ 3 Out 1
3. ⬜ 3 -1 1
4. ✅ 3 Out -1 1
5. ⬜ 2 -1 0
6. ⬜ 3 Out -1 Out 1 Out

## Ex 3 (1.5 points)

Analyze the following code snippet in C++. When the main is executed, Indicate which of the following statements are correct.

```
class C {
private:
  ...
public:
  ...
};

void f1(C &e) { cout << "{f1}";}
void f2(C *e) { cout << "{f2}";}
void f3(shared_ptr<C> e) { cout << "{f3}";}

int main() {
  cout << "{01}"; C e1;
  cout << "{02}"; C *e2 = new C[3];
  cout << "{03}"; shared_ptr<C> e3 = shared_ptr<C> (new C);
  cout << "{04}"; shared_ptr<C> e4 = shared_ptr<C> (new C);
  cout << "{05}"; f1 (e1);
  cout << "{06}"; f2 (e2);
  cout << "{07}"; f3 (e3);
  cout << "{08}"; e1 = (std::move(e2[0]));
  cout << "{09}"; delete[] e2;
  cout << "{10}"; return 0;
}
```

Solution

```
{01}[C]{02}[C][C][C]{03}[C]{04}[C]{05}{f1}{06}{f2}{07}{f3}{08}[MAO]{09}[D][D][D]{10}
[D][D][D]
```

Choose one or more options:

1. ⬜ Line number 1 includes one copy constructor.
2. ✅ Line number 2 includes three standard constructors.
3. ⬜ Each one of lines number 3 and 4 includes one standard constructor and one copy assignment operator.
4. ⬜ Line number 5 includes one standard constructor.
5. ⬜ Line number 6 includes one standard constructor.
6. ⬜ Line number 7 includes one standard constructor.
7. ✅ Line number 8 includes one move assignment operator.
8. ✅ Line number 9 includes three destructors.
9. ✅ Line number 10 includes three destructors.

## Ex 4 (2.0 points)

Explain what a condition variable in C (or C++) is and how it is typically used with mutexes. Describe a typical use case scenario for condition variables in a producer-consumer problem and why each construct must be inserted in the overall scenario.

## Solution

A condition variable is a synchronization primitive that blocks one or more threads until a particular condition is met. It is typically used with a mutex to ensure that threads wait in a way that avoids race conditions. The mutex protects the condition check and modification, and the condition variable allows threads to wait atomically, releasing the mutex while waiting and reacquiring it upon waking up.

In a producer-consumer problem, a condition variable is used to signal consumers when new data is available and to signal producers when space is available. Producers wait on the condition variable if the buffer is full, and consumers wait if the buffer is empty.

```
while (buffer_is_full()) {
  pthread_cond_wait(&cond, &mutex);
}
add_to_buffer(data);
pthread_cond_signal(&cond);
```

## Ex 5 (4.0 points)

A C (or C++) program executes five threads: TA, TB, TC, TD, and TE. These threads are cyclical, run forever, and cooperate to generate sets of symbols on subsequent lines of the standard output. Each one can display one character, `'A'`, `'B'`, `'C'`, `'D'`, `'E'`, respectively, and a symbol "-" or/and a "new line" for each iteration of their primary cycle. Each line must have the following format:

```
A-{B|C}-{D&E}-A
```

This means that for each sequence, there are four sub-sequences separated by "-" symbols and terminated by a new line; the subsequences are the following:

- `A`: Exactly one symbol A.
- `{B|C}`: One symbol B **or** one symbol C.
- `{D&E}`: One symbol D **and** one E in any order (DE or ED)
- `A`: Exactly one symbol A.

The following are correct sequences generated by the program:

```
A-B-ED-A
A-C-ED-A
A-B-DE-A
A-C-DE-A
A-B-DE-A
...
```

## Solution 1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "pthread.h"
#include "semaphore.h"

sem_t *sa, *sbc, *sd, *se;
pthread_mutex_t *me;
int na, nde;

static void *TA ();
static void *TB ();
static void *TC ();
static void *TD ();
static void *TE ();

int main (int argc, char **argv) {
  pthread_t th;

  sa = (sem_t *) malloc (sizeof(sem_t));
  sbc = (sem_t *) malloc (sizeof(sem_t));
  sd = (sem_t *) malloc (sizeof(sem_t));
  se = (sem_t *) malloc (sizeof(sem_t));
  me = (pthread_mutex_t *) malloc (sizeof (pthread_mutex_t));
  na = nde = 0;
  sem_init (sa, 0, 1);
  sem_init (sbc, 0, 0);
  sem_init (sd, 0, 0);
  sem_init (se, 0, 0);
  pthread_mutex_init (me, NULL);

  setbuf(stdout, 0);

  pthread_create (&th, NULL, TA, NULL);
  pthread_create (&th, NULL, TB, NULL);
  pthread_create (&th, NULL, TC, NULL);
  pthread_create (&th, NULL, TD, NULL);
  pthread_create (&th, NULL, TE, NULL);

  pthread_exit(0);
}

static void *TA () {
  pthread_detach (pthread_self ());

  while (1) {
    //sleep (1);
    sem_wait (sa);
    na++;
    if (na==1) {
      printf ("A-");
      sem_post (sbc);
    } else {
      na = 0;
      printf ("-A\n");
      sem_post (sa);
    }

  }

  return 0;
}

static void *TB () {
  pthread_detach (pthread_self ());

  while (1) {
    //sleep (1);
    sem_wait (sbc);
    printf ( "B-");
    sem_post (sd);
    sem_post (se);
  }

  return 0;
}

static void *TC () {
  pthread_detach (pthread_self ());

  while (1) {
    //sleep (1);
    sem_wait (sbc);
    printf ( "C-");
    sem_post (sd);
    sem_post (se);
  }

  return 0;
}

static void *TD () {
  pthread_detach (pthread_self ());

  while (1) {
    //sleep (1);
    sem_wait (sd);
    pthread_mutex_lock (me);
    printf ("D");
    nde++;
    if (nde==2) {
      nde = 0;
      sem_post (sa);
    }
    pthread_mutex_unlock (me);
  }

  return 0;
}

static void *TE () {
  pthread_detach (pthread_self ());

  while (1) {
    //sleep (1);
    sem_wait (se);
    pthread_mutex_lock (me);
    printf ("E");
    nde++;
    if (nde==2) {
      nde = 0;
      sem_post (sa);
    }
    pthread_mutex_unlock (me);
  }

  return 0;
}
```

## Solution 2

```
#include <iostream>
#include <thread>
#include <semaphore>

using namespace std;

counting_semaphore<2> sem_a(2);
binary_semaphore sem_b_or_c(0);
binary_semaphore sem_d(0);
binary_semaphore sem_e(0);
bool start_of_line = true;
bool d_or_e_done = false;
mutex mtx;

void TA() {
   while (true) {
       sem_a.acquire();
       sem_a.acquire();
       cout << 'A';
       if (start_of_line) {
           cout << "-";
           start_of_line = false;
           sem_b_or_c.release();
       } else {
           cout << endl;
           start_of_line = true;
           sem_a.release();
           sem_a.release();
       }
   }
}

void TB() {
   while (true) {
       sem_b_or_c.acquire();
       cout << "B-";
       sem_d.release();
       sem_e.release();
   }
}

void TC() {
   while (true) {
       sem_b_or_c.acquire();
       cout << "C-";
       sem_d.release();
       sem_e.release();
   }
}

void TD() {
   while (true) {
       sem_d.acquire();
       cout << "D";
       {
           lock_guard<mutex> lock(mtx);

           if (d_or_e_done) {
               cout << '-';
               d_or_e_done = false;
           } else {
               d_or_e_done = true;
           }
       }
       sem_a.release();
   }
}

void TE() {
   while (true) {
       sem_e.acquire();
       cout << "E";
       {
           lock_guard<mutex> lock(mtx);

           if (d_or_e_done) {
                cout << '-';
                d_or_e_done = false;
            } else {
                d_or_e_done = true;
            }
        }
        sem_a.release();
    }
}

int main() {
   thread t[5];
   t[0] = thread(TA);
   t[1] = thread(TB);
   t[2] = thread(TC);
   t[3] = thread(TD);
   t[4] = thread(TE);
   for (auto &i: t) {
       i.join();
   }

}
```

## Ex 6 (3.0 points)

The following code implements the recursive merge sort procedure (without the merge function, which we do not have to care about).

```
void mergesort(std::vector<int>& arr, int left, int right) {
  if (left>=right)
    return;
  int mid = left + (right-left) / 2;
  mergesort (arr, left, mid);
  mergesort (arr, mid + 1, right);
  merge(arr, left, mid, right);
  return;
}

int main() {
  const int size = 12;
  std::vector<int> array;

  for (int i=0; i<size; i++)
    array.push_back(rand()%10000);

  mergesort(array, 0, array.size() - 1);

  std::cout << "Sorted array: ";
  for (int val: array) {
    std::cout << val << " ";
  }
  std::cout << std::endl;
  return 0;
}
```

Implement in C++ a parallel version using promises and futures. The main has to run a task, set the values into the array, and wake up the task to start the sorting process. Promises and futures can be used to return the results of previously sorted subarrays or simply to synchronize tasks.

## Solution 1

```
#include <iostream>
#include <vector>
```

```cpp
#include <algorithm>
#include <thread>
#include <future>

void merge(std::vector<int>& arr, int left, int mid, int right) {
  int n1 = mid - left + 1;
  int n2 = right - mid;

  std::vector<int> leftArr(n1), rightArr(n2);

  for (int i = 0; i < n1; ++i)
      leftArr[i] = arr[left+i];
  for (int i = 0; i < n2; ++i)
      rightArr[i] = arr[mid + 1 + i];

  int i = 0, j = 0, k = left;
  while (i<n1 && j<n2) {
      if (leftArr[i] <= rightArr[j]) {
          arr[k++] = leftArr[i++];
      } else {
          arr[k++] = rightArr[j++];
      }
  }

  while (i < n1) {
      arr[k++] = leftArr[i++];
  }

  while (j < n2) {
      arr[k++] = rightArr[j++];
  }

  return;
}

void parallelMergeSort(std::vector<int>& arr, int left, int right) {
  if (left>=right)
    return;

  int mid = left + (right-left) / 2;

  auto leftFuture = std::async(std::launch::async, [&arr, left, mid]() {
    parallelMergeSort(arr, left, mid);
  });
  auto rightFuture = std::async(std::launch::async, [&arr, mid, right]() {
    parallelMergeSort(arr, mid + 1, right);
  });

  leftFuture.get();
  rightFuture.get();

  merge(arr, left, mid, right);
  return;
}

void pms(std::future<int> &f, std::vector<int> &arr) {
  int size = f.get();
  parallelMergeSort(arr, 0, size-1);
}

int main() {
  const int size = 12;
  std::promise<int> p;
  std::future<int> f = p.get_future();
  std::vector<int> array;

  std::thread t (pms, std::ref(f), std::ref(array));
  for (int i=0; i<size; i++)
    array.push_back(rand()%10000);
  std::cout << "Initial array: ";
  for (int val: array) {
    std::cout << val << " ";
  }
  std::cout << std::endl;

  p.set_value (size);

  t.join();

  std::cout << "Sorted array: ";
  for (int val: array) {
    std::cout << val << " ";
  }
  std::cout << std::endl;
  return 0;
}
```

## Ex 7 (2.0 points)

Specify what are templates in C++ and why they are helpful.
After that, consider the following code snippet in which a class template is instantiated to store objects of different types. Define a class template that can be compiled and work with the main program reported below.

```
int main() {
  Box<int> intBox(123);
  Box<std::string> strBox("Hello");
  Box<float> fB;
  fB.setValue (13.24);
  std::cout << intBox.getValue() << std::endl;
  std::cout << strBox.getValue() << std::endl;
  return 0;
}
```

## Solution

Templates in C++ allow functions and classes to operate with generic types. They enable code reuse and type safety by allowing a function or class to work with any data type. Templates are handy for implementing generic data structures and algorithms.

```
template<typename T>
class Box {
private:
    T value;
public:
  Box() {}
  Box(T val) : value(val) {}
  void setValue (T val) {value=val;}
  T getValue () const { return value; }
};

int main() {
  Box<int> intBox(123);
  Box<std::string> strBox("Hello");
  Box<float> fB;
  fB.setValue (13.24);
  std::cout << intBox.getValue() << std::endl;
  std::cout << strBox.getValue() << std::endl;
  return 0;
}
```