System and Device Programming
Standard Exam
07.07.2023

Ex 1 (1.5 points)

Suppose that the following program is run using the command
`./pgrm abc 2`
Indicate the possible output or outputs of the program. Note that wrong answers imply a penalty in the final score.

```
#define L 100

int main (int argc, char *argv[]) {
  int i, j;
  char str1[L], str2[L];
  setbuf(stdout,0);
  i = atoi (argv[2]);
  for (j=0; j<i; j++) {
    if (fork () == 0) {
      sprintf (str1, "echo -n [%d]", j);   // "-n" indicates no "new line"
      system (str1);
    } else {
      printf ("{%d}", j);
      sprintf (str1, "%s", argv[1]);
      sprintf (str2, "%d", j);
      execlp (argv[0], "myPgrm", str1, str2, NULL);
    }
  }
  return (0);
}
```

Choose one or more options:

1. ⬜ `[1]{1}[0]{0}{1}[1]`
2. ⬜ `[0]{0}[0]{0}{1}[1]`
3. ✅ `{0}[0]{1}{0}[1][0]`
4. ⬜ `{0}[0]{0}{1}[1][0]`
5. ✅ `[0]{0}[1]{1}{0}[0]`
6. ✅ `{0}[0][1]{1}[0]{0}`
7. ⬜ `{0}{0}{1}[1][0][0]`

Ex 2 (1.5 points)

Analyze the following code snippet in C++. When the main is executed, indicate how many (standard) constructors, copy constructors, and destructors are called.

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

int main() {
```

```
  C e1;
  C e2 = e1;
  C e3 = *new C;
  return 0;
}
```

Solution

```
{1}[C]{2}[CC]{3}[C][CC]{4}[D][D][D]
```

Choose one or more options:

1. ⬜ 1 constructor, 2 copy constructors, and 3 destructors.
2. ⬜ 1 constructor, 1 copy constructor, and 2 destructors.
3. ⬜ 2 constructors, 2 copy constructors, and 4 destructors.
4. ✅ 2 constructors, 2 copy constructors, and 3 destructors.
5. ⬜ 3 constructors, 1 copy constructor, and 3 destructors.
6. ⬜ 3 constructors, 2 copy constructors, and 3 destructors.
7. ⬜ 1 constructor, 2 copy constructors, and 2 destructors.

Ex 3 (1.5 points)

Analyze the following code snippet in C++. Indicate the possible output or outputs obtained by executing the program. Note that wrong answers imply a penalty in the final score.

```
auto lambda = [](  std::string h )->bool{
   return ( h != "-" && h != "." );
};

int main() {
   std::string s("123.456.789-00");
   std::vector<std::string> num;
   for (int i = 0; i < s.length() ; i++) {
      num.push_back( s.substr(i, 1) );
   }
   cout << s << "#";
   for( auto z : num ){ if (lambda(z)) std::cout << z; }; std::cout << '\n';
   return 0;
}
```

Choose one or more options:

1. ⬜ The program displays the sequence “123.456.789-00#“
2. ⬜ The program displays the sequence “123.456.789-00#..-“
3. ⬜ The program does not run as there is a bug.
4. ✅ The program displays the sequence “123.456.789-00#12345678900”
5. ⬜ The program displays the sequence “123.456.789-00#123.456.789-00”
6. ⬜ The program displays the sequence “12345678900#12345678900”

Ex 4 (2.5 points)

A file stores the information concerning a set of students. For each student, one line of the file indicates the following information: The register number, last and first name (we suppose all students have only two names), the number of examinations passed, and the mark for each one of those exams. The following is a correct example of such a file:

```
100000 Granger Hemione 8 30 30 30 29 29 30 30 29
124567 Potter Harry 5 30 18 24 29 28
113567 Weasley Ron 4 28 26 27 28
```

...

Write a C++ function that:

- Receives the file name as a parameter.
- Store the file content in a hash map of sets. Each element of the hash table stores: The register number (which is also the key of the hash table), the last and first name (standard fields), and the set of the marks received by the student (into a field of type set).

The function must return a reference to the data structure it has created.

Solution 1

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <unordered_map>
#include <set>

using namespace std;

struct my_data {
    string ln;
    string fn;
    multiset<int> marks;
};

unordered_map <int, my_data> &read_file(const string& fn) {
    unordered_map<int, my_data> *db;
    db = new unordered_map<int, my_data>;
    ifstream fin(fn);

    if (!fin.is_open())
    {
    cout << "Error opening " << fn << endl;
    }
    else
    {
        int n;
        while (fin >> n) {
            int n_ex;
            my_data d;
            fin >> d.ln >> d.fn >> n_ex;

            for (int i = 0; i < n_ex; ++i) {
                int m;
                fin >> m;
                d.marks.emplace(m);
            }
            db->emplace(n, d);
        }
        fin.close();
    }
    return *db;
}


int main() {
    unordered_map<int, my_data> reg;
```

```
    reg = read_file("db.txt");

    for (auto &i:reg)
    {
        cout << i.first << " ";
        cout << i.second.ln << " " << i.second.fn << " ";
        for (auto &j : i.second.marks)
           cout << j << " ";
        cout << endl;
    }

    return 0;
}
```

Solution 2

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <unordered_map>
#include <set>

using namespace std;

struct my_data {
    string ln;
    string fn;
    multiset<int> marks;
};

unordered_map<int, my_data> read_file(const string& fn) {
    unordered_map<int, my_data> db;
    ifstream fin(fn);

    if (!fin.is_open())
    {
    cout << "Error opening " << fn << endl;
    }
    else
    {
        int n;
        while (fin >> n) {
            int n_ex;
            my_data d;
            fin >> d.ln >> d.fn >> n_ex;

            for (int i = 0; i < n_ex; ++i) {
                int m;
                fin >> m;
                d.marks.emplace(m);
            }
            db.emplace(n, d);
        }
        fin.close();
    }
    return db;
}

int main() {
    unordered_map<int, my_data> reg;

    reg = read_file("db.txt");

    for (auto &i:reg)
```

```
    {
        cout << i.first << " ";
        cout << i.second.ln << " " << i.second.fn << " ";
        for (auto &j : i.second.marks)
           cout << j << " ";
        cout << endl;
    }

    return 0;
}
```

Ex 5 (2.5 points)

Describe how to use C++ templates and which problem they solve. Report an example to manipulate a FIFO list of different types (booleans, integers, floats, strings, etc.). Illustrate how to use this template class (i.e., write the client program) and indicate at least two different ways to organize it into the header (.h) and the source (.cpp) file.

Solution

C++ templates are used to make functions and classes independent of data types, supporting generic programming. If we use templates, we can write more general functions and classes that can accept any data types. At compile-time, the compiler produces the needed versions of the functions and classes depending on the parameters used for calling them. We need to define a template before defining a function/class using that template.

The syntax can be inferred from the following examples.
For functions:

```
template <typename T>
bool compare(T first, T second){
  return(first<second);
}
```

For classes:

```
template <typename T>
Class Foo{
 private:
 T element;

 public:
 T get_elem{ cout << element <<endl;};
};
```

Note that we can also use different types within a template:

```
template<typename T, typename R>
void print(T first, R second){
cout << first << " " << second << endl;
```

When we call such functions and classes from the main, for functions we can omit the datatype, but for classes the compiler cannot deduce the type and we must specify it:

```
compare(5, 3);
print(5,"ok");
but:
Foo<int> f1;
```

Now we consider the example of a FIFO list of different types:

```
template<typename T>
class FifoList{
  private:
```

```
  std::list<T> list();

  public:
  void insert(T element){
   list.push_back(element);
  }
  T delete(){
   T t = *list.end();
   list.pop_back();
   return t;
  }
  void print(){
   for(auto i: list)
     cout<< i <<  " " ;
  }
}
```

The main:

```
int main(){
  FifoList<int> numbers;
  FifoList<string> words;
  numbers.insert(5);
  numbers.insert(7);
  numbers.delete();
  numbers.print();
  words.insert("ok");
  words.inser("fine");
  words.print();
  words.delete();
  words.print();
}
```

To organize this code into separate files we have numerous options, such as:

1. define and declare template classes and functions in a .h file, to be included in the main cpp file
2. declare the template classes and functions a .h file, write the template class or function implementation in an .hpp file, and then include both .h and .hpp files in the main cpp file
3. declare the class and the template in the .h file, write the template class or function implementation in a .cpp file, adding into the latter an explicit template instantiation, such as template class FifoList<string>; template class FifoList<int>;

Ex 6 (2.5 points)

The Pthread library implements binary semaphores with the functions `pthread_mutex_init`, `pthread_mutex_lock`, and `pthread_mutex_unlock`. Using these functions, implement a counting semaphore, i.e., a non-binary semaphore, whose starting value is `count`.

Solution

```
typedef struct {
  int count;              /* the counter */
  pthread_mutex_t lock;   /* mutex ensuring exclusive access to count */
  pthread_mutex_t s;      /* real semaphore */
} Semaphore;

static void semaphore_init (Semaphore *s, int i) {
  pthread_mutex_init (&s->lock, NULL);
  pthread_mutex_init (&s->s, NULL);
  pthread_mutex_lock(&s->s);
  s->count = i;
}
```

```
static void semaphore_wait (Semaphore *s) {
  pthread_mutex_lock (&s->lock);
  s->count--;
  if (s->count < 0){
    pthread_mutex_unlock (&s->lock);
    pthread_mutex_lock (&s->s);
  } else
    pthread_mutex_unlock (&s->lock);
}

static void semaphore_signal (Semaphore *s) {
  pthread_mutex_lock (&s->lock);
  s->count++;
  if (s->count <= 0)
    pthread_mutex_unlock (&s->s);
  pthread_mutex_unlock (&s->lock);
}
```

Ex 7 (3.0 points)

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

$A^2\ \{B|C\}^{2+}\ D^2$

This means that for each sequence, there are:

- A^2: Exactly two symbols A.
- $\{B|C\}^{2+}$: Two or more symbols B or C (i.e., at least `BB`, `CC`, `BC`, or `CB`, and then maybe others `B` or `C`).
- D^2: Exactly two symbols `D`.

Each sequence is terminated by a “new line” character.
The following is a correct example of the execution of such a program:

```
AABCDD
AACBDD
AACBCDD
AABCCDD
AABCBCCCDD
AACBCBDD
...
```

Solution in C

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

sem_t *sa, *sbc, *sd;
int na, nbc, nd;

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

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));
  na = nbc = nd = 0;
  sem_init (sa, 0, 1);
  sem_init (sbc, 0, 0);
  sem_init (sd, 0, 0);

  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_exit(0);
}

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

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

  return 0;
}

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

  while (1) {
    sem_wait (sbc);
    printf ( "B");
    nbc++;
    if (nbc==2) {
      sem_post (sd);
    }
    sem_post (sbc);
  }

  return 0;
}

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

  while (1) {
    sem_wait (sbc);
    printf ( "C");
    nbc++;
    if (nbc==2) {
      sem_post (sd);
    }
```

```
    sem_post (sbc);
  }

  return 0;
}

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

  while (1) {
    sem_wait (sd);
    if (nd==0) {
      sem_wait (sbc);
    }
    printf ("D");
    nd++;
    if (nd==1) {
      sem_post (sd);
    } else {
      printf ("\n");
      nbc = nd = 0;
      sem_post (sa);
    }
  }

  return 0;
}
```

Solution in C++

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

using namespace std;

counting_semaphore<1> sa(1), sbc(0), sd(0);
int counta, countbc, countd;

void f_A() {

    while (true) {
        sa.acquire();
        ++counta;
        cout << "A";
        if (counta == 2) {
            counta = 0;
            sbc.release();
        } else sa.release();
    }

}

void f_B() {
    while (true) {
        sbc.acquire();
        cout << "B";
        countbc++;
        if (countbc > 1) sd.release();
        else sbc.release();
    }
}
```

```
void f_C() {
    while (true) {
        sbc.acquire();
        cout << "C";
        countbc++;
        if (countbc > 1) sd.release();
        else sbc.release();
    }
}


void f_D() {
    while (true) {
        sd.acquire();
        cout << "D";
        countd++;
        if (countd > 1) {
            cout << endl;
       //     this_thread::sleep_for(chrono::seconds(1));
            countbc = countd = 0;
            sa.release();
        } else sd.release();
    }
}

int main() {

    counta = countbc = countd = 0;

    thread TA(f_A);
    thread TB(f_B);
    thread TC(f_C);
    thread TD(f_D);

    TA.detach();
    TB.detach();
    TC.detach();
    TD.detach();

    this_thread::sleep_for(chrono::seconds(5));

}
```