PPL-HW1

Assignment 1 of CE2004, Principles of Programming Languages

Score: 100 points

Due Time: 24:00 6th April

> **P.S.:**
>
> 1. You need to type your answers in a file and print them out in answer sheets, then submit your answer sheets to the TAs through new-eeclass.
> 2. Late submission will not be accepted.
> 3. You can discuss these questions with your classmates; however, copying other student’s answers is strictly prohibited.

---

(1) (6 points)

The CPU of Mary’s computer can complete an instruction more quickly than the CPU of Tom’s CPU; hence, Mary’s computer can always complete a program more quickly than Tom’s computer. Is the above statement correct? Give your explanation.

Ans.

This statement is definitely not correct. Faster CPU indeed accompany with the faster speed of processing the program; nonetheless, the factors deciding the speed of completing a program not only stick to how quickly of the CPU, but also relate to the size of memory, amount of storage, efficiency of the code, and so on. Hence, despite of Mary's computer with a faster CPU, her computer not always complete a program more quickly than Tom's computer if Tom's computer with larger memory, storage or if the program is not optimized for the Mary's CPU architecture.

---

(2) (12 points)

Good language readability can improve writability. Good language writability is detrimental to readability.

(a) Which one of the above two statements is correct? Which one of the above two statements is wrong?

(b) Give your explanation.

Ans.

Good language readability can improve writability. This statement is correct. Once the programming is easily to read which means more understandable either, it can help the programmer easily to express their thoughts and efficiently to realize what others' communicating about.

Good language writability is detrimental to readability. This statement is wrong. Actually, if language is easy to write, it often leads to writing more accurately, clearly, and easier to understand.

Above all, good programming language readability and writability are not the opposite relationship. Both are important for the developer to cummunicate mutually. Therefore, we have to take care of both of them when we are designing the program to make our code better.

---

(3) (9 points)

What follows is an excerpt of a **Javascript** program. Assume before location 1, variable `list` has never been used.

```
  :                 -- location 1
list = [1, 2]
prefix= list        -- location 2
prefix = 47
list = prefix       -- location 3
  :
```

(a) At location 1, what is the data type of variable `list`?

(b) At location 2, what is the data type of variable `prefix`?

(c) At location 3, what is the data type of variable `list`?

Ans.

(a)

The data type of variable `list` is array.

(b)

The data type of variable `prefix` is array.

(c)

The data type of variable `list` is number.

---

(4) (12 points)

A program consists of the following two files, `fileu.c` and `filev.c`

```
/*=========== fileu.c ===============*/
int a = 100;          // location 1
extern int t;         // location 2
```

```c
int bar(int y) {      // location 3
    int x;            // location 4
    x=y+t;            // location 5
    return(x);
}                     // location 6

/*============ filev.c ================*/
#include<stdio.h>
int t=9;                               // location 7
extern int a;                          // location 8
extern int bar(int);                   // location 9
int main(){                            // location 10
    int z;                             // location 11
    printf("a=%d\n", a);
    printf("bar(3)=%d\n",bar(3));
}                                      // location 12
```

(a) List the locations of all variable definitions in the above two files.

(b) List the locations of all variable declarations in the above two files.

(c) List the locations of all function definitions in the above two files.

(d) List the locations of all function declarations in the above two files.

P.S.: A function formal parameter is also deemed as a variable.

Ans.

(a) Variable definitions

- fileu.c:
  - Location 1: a (global variable)
  - Location 4: x (local variable)
- filev.c:
  - Location 7: t (global variable)
  - Location 11: z (local variable)

(b) Variable declarations

- fileu.c:
  - Location 2: t (external variable)
- filev.c:
- Location 8: a (external variable)

(c) Function definitions

- fileu.c:
  - Location 3: bar function

(d) Function declarations

- filev.c:
    - Location 9: bar function (external function)

---

(5) (8 points) What follows is a **C** program.

```
#include <stdio.h>
int total_income, total_visitors_global;

void zoo(char *name, int visitors) {
    int adult, children;
    static int total_visitors=0;
        :
    total_visitors=total_visitors+visitors;    // location 1
    total_visitors_global=total_visitors;
        :
}
int main() {
    int ticket_price_each_animal_type=2;
    printf(“Good Morning!\n”);                // location 2
    zoo(“giraff”, 600);
    zoo(“elephant”, 300);
    zoo(“hippo”,100);
    total_income=ticket_price_each_animal_type*total_visitors_global;
        :
}
```

(a) At location 1, list the names of variables or parameters that have memory assigned to it.

(b) At location 2, list the names of variables or parameters that have memory assigned to it.

Ans:

(a) Location 1

- total_visitors: static local variable and being assigned 0.
- visitors: parameter of `zoo` function, the value is assigned by the value passed by calling `zoo`.

(b) Location 2

- ticket_price_each_animal_type: local variable and being assigned 2.
- total_visitors_global: global variable and being assigned value of total_visitors.

---

(6) (9 points)

What follows is the content of program `add_a.c`.

```
/*---------------------------------------------------*/
#include <stdio.h>
int a=1 , b=6;
int c[10000]={1};
int main() {
    a=b+c[0];          /* location 1*/
}
/*---------------------------------------------------*/
```

Assume add_a.exe is the executable of add_a.c.

What follows is the content of program add_b.c.

```
/*---------------------------------------------------*/
#include <stdio.h>
int a=1 , b=6;
int c[10000];
int main() {
    c[0]=1;
    a=b+c[0];         /*location 2*/
}
/*----------------------------------------------------*/
```

Assume add_b.exe is the executable of add_b.c.

(a) At location 1 of add_a.c what is the value of variable a?

(b) At location 2 of add_b.c what is the value of variable a?

(c) For files add_a.exe and add_b.exe, which of these two files has larger size and why?

Ans.

(a)

The value of variable a is 7.

(b)

The value of variable a is 7.

(c)

add_a.exe is larger than add_b.exe. It is that in add_a.c the whole of the array c is assigned initially with 1. However, in add_b.c is only the first element of array c is assigned 1. Therefore, add_a.exe is larger than add_b.exe.

---

(7) (12 points)

(a) What follows is a C program.

```c
# include <stdio.h>
int a;
int bar(int x, int y) {
    int b;
    return b = x+y;
}
int main() {
    int *p;
    p = (int *) malloc (sizeof(int));
    *p = bar (8,9);
}
```

In the above program,

(i) which variables are static variables?

(ii) And which variables are stack dynamic variables?

(iii) And which variables are explicit-heap dynamic variables?

P.S.: A function formal parameter is also deemed as a variable.

(b) What follows is a Java program excerpt.

```java
class Circle {
    int setVariable(int s) {
        int r;
        r=6;
        return s+r;
    }
}
public class ShowArea {
    public static void main(String args[]) {
        Circle cir= new Circle();
        int a;
        a= cir.setVariable(8);
    }
}
```

In the above program,

(i) which variables are static variables?

(ii) And which variables are stack dynamic variables?

(iii) And which variables are explicit-heap dynamic variables?

Ans.

(a)

- (i) There are no static varables.
- (ii) `a, b, p, x, y`.
- (iii) `p` because it is allocated using malloc().

(b)

- (i) There are no static variables.
- (ii) `cir` and `a` because they are declared inside a `main()`.
- (iii) There are no explicit-heap dynamic variables.

---

(8) (10 points)

What follows is the content of a C program `example.c`.

```c
#include <stdio.h>            // location 11
int bar(int x) {
    int a, b, c;              // location 1
    c=x;                      // location 2
    b=x*9;                    // location 3
    a=foo();                  // location 4
    return a;                 // location 5
}
int foo() {
    int a=1, b=2, c;          // location 6
    c=a+b;                    // location 7
    return c;                 // location 8
}
int main() {
    int a=1, b=2, c=3;        // location 9
    return bar(a);            // location 10
}
```

(a) Are variable `b` defined at location 1, variable `b` defined at location 6, and variable `b` defined at location 9 the same variable?

(b) During run time, at what locations of the above program, variable `b` defined at location 9 and variable `b` defined at location 1 have storage bound to them, but variable `b` defined at location 6 does not have storage bound to it?

Ans.

(a)

- Location 1: only being defined.
- Location 6: being defined and assigned 2
- Location 9: being defined and assigned 2

In sum up, they are not the same variable.

(b)

- During run time, variable `b` defined at `location 9` has storage bound to it when execuating `main` function.
- Variable `b` defined at `location 1` has storage bound to it only when the `bar` function is called, and then released when `bar` returns.
- Variable `b` defined at `location 6` has storage bound to it only when the `foo` function is called, and then released when `foo` returns.

---

(9) (8 points)

Assume each integer variable uses four bytes to store it values. And each float point variable uses four bytes to store its value. For the following two C program excerpts, (a) and (b), which of them have a type error? Explain your answers.

(a)

```
int a;
union course {
    int b;
    float c;
} security;
security.b = 3;      // location 1
a = security.b;      // location 2
```

(b)

```
int a;
union course {
    int   b;
    float c;
} security;
security.c = 3.3;    // location 3
a = security.c;      // location 4
```

Ans.

In program(a), No type error. `a` is defined as `int` type, then in the union of `course` the type of `b` is `int` type. Therefore, in location 1, the variable 3 assigned to `security.b` is acceptable.

In program(b), Type error occur in the location 4. `security.c` is defined as `float` type, `a` is defined as `int` type; therefore, while `security.c` is assigned to `a`, it will occur the type error and it might have resulted in truccation error, which is a loss of precision.

---

(10) (8 points)

What follows is a **Fortran** program.

```
PROGRAM Hello
  IMPLICIT NONE
  INTEGER :: Patrick island, a
  PRINT *, 'Welcome to Fortran'
  a=6
  Pat rick is land = 19
  Patrick island = Pat rick is land + a    ! line 8
END PROGRAM Hello
```

After line 8 of the above program is executed,

(i) what is the value of variable `Patrick island`?

(ii) what is the value of variable `Pat rick is land`?

Ans:

(i)

It will occur the `syntax error`, `Patrick island` and `Pat rick is land` is not allowed, we can rename with `Patrick_island`, then we got the value of `Patrick_island`, 25.

(ii)

It also occur the `syntax error`, `Pat rick is land` is not allowed; therefore the value of it is not relevant.

---

(11) (6 points)

Compiler Optimization tries to improve programs by making them smaller or faster or both. Hence, we should always use compiler optimization to compile our programs. Is the above claim correct? Give your explanation.

Ans.

"We should always use compiler optimization to compile our programs" is not completely correct. It may not always the best choice to conduct the program.

- Optimization might have resulted in some bugs because not every cases are suitable to optimize the code.
- Optimization will increase the time of conducting the larger projects because it take more time to optimize, not to conduct.
- Hard to debug. It might have broken the original style of the code that we can't find the error when we want to realize what we have written.
- Continue with the last points, hard to maintain because the original style of the code have changed, then we need to take more time to understand.

Above all, it is not always suitable for optimization with the compiler, it will decrease the value of the programmer. We have to realize the results when using the tools we have, then decide which tools we can use case-by-cases to prevent the results we don't hope.