KPIT TECHNOLOGIES

WEEKLY REPORT

WEEK 4- Report (DATE: 10/6/2024)

Student name	Week	Branch	USN
Juweriah Herial	4	Circuit (ECE)	1NH20EC053

Yashavant Kanetkar Book 19th edition

Question 151-200:

151) Point out the error, if any, in the following programs:

```
# include <stdio.h>
int main( )
{
typedef struct patient
{
char name[ 20 ] ; int age ;
int systolic_bp ; int diastolic_bp ;
} ptt ;
ptt p1 = { "anil", 23, 110, 220 } ;
printf ( "%s %d\n", p1.name, p1.age ) ;
printf ( "%d %d\n", p1.systolic_bp, p1.diastolic_bp ) ;
return 0 ;
}
```

Logical error: The diastolic blood pressure (bp) value should be in a reasonable range. Typically, diastolic bp should be less than systolic bp and a value of 220 for diastolic bp is not plausible. A typical normal range for diastolic bp is between 60 and 90 mm Hg. Here, 80 is used as a more reasonable example.

152) Point out the error, if any, in the following programs:

```
# include <stdio.h>
void show( ) ;
int main( )
{
```

void (*s)( ) ;
s = show ;
(*s)( ) ;
s( ) ;
return 0 ;
}
void show( )
{
printf ("don't show off. It won't pay in the long run\n") ;
}

The corrected/improved code is:

```
#include <stdio.h>

void show();
int main()
{
   void (*s)();
   s = show;

   (*s)();
   s();

   return 0;
}

void show()
{
   printf("don't show off. It won't pay in the long run\n");
}
```

153) Point out the error, if any, in the following programs:

 # include <stdio.h>
void show (int, float) ;
int main( )
{
void (*s)(int, float) ;
s = show ;
(*s)(10, 3.14) ;
return 0 ;
}
void show (int i, float f)
{

printf ("%d %f\n", i, f) ;
}

There are no errors in this program. It will execute without any issues.

154) Write a program, which stores information about a date in a structure containing three members—day, month and year. Using bit fields, the day number should get stored in first 5 bits of day, the month number in 4 bits of month and year in 12 bits of year. Write a program to read date of joining of 10 employees and display them in ascending order of year.

```
#include <stdio.h>
struct Date {
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year : 12;
};
int compareDates(const struct Date *date1, const struct Date *date2) {
    if (date1->year != date2->year)
        return date1->year - date2->year;
    if (date1->month != date2->month)
        return date1->month - date2->month;
    return date1->day - date2->day;
}
int main() {
    struct Date employees[10];
    int i;
    printf("Enter date of joining for 10 employees (DD MM YYYY):\n");
    for (i = 0; i < 10; ++i) {
        printf("Employee %d: ", i + 1);
        scanf("%u %u %u", &employees[i].day, &employees[i].month,
&employees[i].year);
    }
    for (i = 0; i < 10 - 1; ++i) {
        for (int j = 0; j < 10 - i - 1; ++j) {
            if (compareDates(&employees[j], &employees[j + 1]) > 0) {
                // Swap the dates
                struct Date temp = employees[j];
                employees[j] = employees[j + 1];
                employees[j + 1] = temp;
            }
        }
    }
    printf("\nDate of joining of employees in ascending order of year:\n");
    for (i = 0; i < 10; ++i) {
```

```
        printf("%02u/%02u/%04u\n", employees[i].day, employees[i].month,
employees[i].year);
    }
    return 0;
}
```

155)Write a program to read and store information about insurance policy holder. The information contains details like gender, whether the holder is minor/major, policy name and duration of the policy. Make use of bit-fields to store this information.

```
#include <stdio.h>
#include <string.h>
struct InsurancePolicy {
    unsigned int gender : 1;
    unsigned int isMinor : 1;
    unsigned int duration : 6;
    char policyName[20];
};

int main() {
    struct InsurancePolicy holder;
    printf("Enter information about the insurance policy holder:\n");
    printf("Gender (0 for male, 1 for female): ");
    scanf("%u", &holder.gender);
    printf("Is the holder minor? (0 for major, 1 for minor): ");
    scanf("%u", &holder.isMinor);

    printf("Duration of the policy (in years, max 63): ");
    scanf("%u", &holder.duration);
    printf("Policy name: ");
    scanf("%s", holder.policyName);
    printf("\nInformation about the insurance policy holder:\n");
    printf("Gender: %s\n", (holder.gender == 0) ? "Male" : "Female");
    printf("Is the holder minor? : %s\n", (holder.isMinor == 0) ? "No" :
"Yes");
    printf("Duration of the policy: %u years\n", holder.duration);
    printf("Policy name: %s\n", holder.policyName);
    return 0;
}
```

156) What is a Programming Paradigm?

Programming paradigm means the principle that is used for organizing programs. There are two major Programming Paradigms, namely, Structured

Programming and Object-Oriented Programming (OOP). C language uses the Structured Programming Paradigm, whereas, C++, C#, VB.NET or Java make use of OOP. OOP has lots of advantages to offer. But even while using this organizing principle you would still need a good hold over the language elements of C and the basic programming skills.

157) Is it true that Operating Systems like Windows, Linux and UNIX are written in C?

Android are written in C. This is because even today when it comes to performance (speed of execution) nothing beats C. Also, the functions exposed by the Operating System API can be easily called through any language. Moreover, if one is to extend the operating system to work with new devices one needs to write Device Driver programs. These programs are exclusively written in C.

158) What do you mean by scope of a variable? What are the different types of scopes that a variable can have?

Sol. Scope indicates the region over which the variable's declaration has an effect. The four kinds of scopes are—file, function, block and prototype.

159)Which of the following statement is a declaration and which is a definition?

extern int i ;

int j ;

First is declaration, second is definition.

160) What are the differences between a declaration and a definition?

There are two differences between a declaration and a definition:

In the definition of a variable space is reserved for the variable and some initial value is given to it, whereas a declaration only identifies the type of the variable. Thus, definition is the place where the variable is created or assigned storage, whereas declaration refers to places where the nature of the variable is stated but no storage is allocated. Secondly, redefinition is an error, whereas, redeclaration is not an error.

161) Is it true that a global variable may have several declarations, but only one definition? [Yes/No]
Yes

162) Is it true that a function may have several declarations, but only one definition? [Yes/No]
Yes

163) When we mention the prototype of a function are we defining the function or declaring it?
We are declaring it. When the function, along with the statements belonging to it is mentioned, we are defining the function.

164) Some books suggest that the following definitions should be preceded by the word *static*. Is it correct?
int a[ ] = { 2, 3, 4, 12, 32 } ;
struct emp e = { "sandy", 23 } ;
Pre-ANSI C compilers had such a requirement. Compilers which conform to ANSI C standard do not have such a requirement.

165) If you are to share the variables or functions across several source files, how would you ensure that all definitions and declarations are consistent?
The best arrangement is to place each definition in a relevant '.c' file. Then, put an external declaration in a header file ('.h' file) and use #include to bring in the declaration wherever needed. The '.c' file which contains the definition should also include the header file, so that the compiler can check that the definition matches the declaration.

166) Global variables are available to all functions. Does there exist a mechanism by way of which I can make it available to some and not to others?

No. The only way this can be achieved is to define the variable locally in main( ) instead of defining it globally and then passing it to the functions which need it.

167) What are the different types of linkages?
There are three different types of linkages—external, internal and none. External linkage means global, non-static variables and functions, internal linkage means static variables and functions with file scope, and no linkage means local variables.

168) What is size_t ?
It is the type of the result of the sizeof operator. size_t is used to express the size of something or the number of characters in something. For example, it is the type that you pass to malloc( ) to indicate how many bytes you wish to allocate. Or it is the type returned by strlen( ) to
indicate the number of characters in a string. Each implementation chooses a type like unsigned int or unsigned long (or something else) to be its size_t, depending on what makes most sense. Each implementation publishes its own choice of size_t in several header files like 'stdio.h', 'stdlib.h', etc. In most implementations size_t is defined as:
typedef unsigned int size_t ;
This means that on this particular implementation size_t is an unsigned int. Other implementations may make other choices. What is important is that you should not worry about what size_t looks like for a particular implementation; all you should care about is that it is the *right* type for representing object sizes and count.

169) What is more efficient, a switch statement or an if-else chain?
As far as efficiency is concerned there would hardly be any difference, if at all. If the cases in a **switch** are sparsely distributed the compiler may internally use the equivalent of an **if-else** chain instead of a compact jump table. However, one should use **switch** where one can. It is definitely a cleaner way to program and certainly is not any less efficient than the **if-else** chain.

170) Can we use a switch statement to switch on strings?
No. The cases in a **switch** must either have integer constants or constant expressions that evaluate to integer costants.

171) In which order do the Relational, Arithmetic, Logical and Assignment operators get evaluated in C?
Arithmetic, Relational, Logical, Assignment

172) How come that the C standard says that the expression
j = i++ * i++ ;
is undefined, whereas, the expression
j = i++ && i++ ;
is perfectly legal?
According to the C standard an object's stored value can be modified only once (by evaluation of expression) between two sequence points. A sequence point occurs:

- At the end of full expression (expression which is not a subexpression in a larger expression)
- At the &&, || and ?: operators
- At a function call (after the evaluation of all arguments, just before the actual call)

Since in the first expression **i** is getting modified twice between two sequence points the expression is undefined. Also, the second expression is legal because a sequence point is occurring at **&&**, and **i** is getting modified once before and once after this sequence point.

173) If a[i] = i++ is undefined, then by the same reason i = i + 1 should also be undefined. But it is not so. Why?
The standard says that if an object is to get modified within an expression, then all accesses to it within the same expression must be for computing the value to be stored in the object. The expression a[i] = i++ is disallowed because one of the accesses of i (the one in a[i]) has nothing to do with the value that ends up being stored in i. In this case the compiler may not know whether the access should take place before or after the incremented value is stored. Since there's no good way to define it, the standard declares it as undefined. As against this, theexpression i = i + 1 is allowed because i is accessed to determine i's final value.

174) Will the expression *p++ = c be disallowed by the compiler?
No. Because here even though the value of p is accessed twice it is used to modify two different objects p and *p.

175) Why should you use functions in your C program?
There are two reasons for using functions:
(a) Writing functions avoids rewriting the same code over and over. Suppose you have a section of code in your program that calculates area of a triangle. If later in the program you want to calculate the area of a different triangle, you won't like it if you are required to write the same instructions all over again. Instead, you would prefer to jump to a 'section of code' that calculates area and then jump back to the place from where you left off. This section of code is nothing but a function.
(b) By using functions, it becomes easier to write programs and keep track of what they are doing. If the operation of a program can be divided into separate activities, and each activity placed in a different function, then each could be written and checked more or less independently. Separating the code into modular functions also makes the program easier to design and understand.

176) In what form are the library functions provided?
Library functions are never provided in source code form. They are always made available in object code form obtained after compilation.

177) What is the type of the variable b in the following declaration?
#define FLOATPTR float *
FLOATPTR a, b ;
float and not a pointer to a float, since on expansion the declaration becomes:
float *a, b ;

178) Is it necessary that the header files should have a .h extension?
No. However, traditionally they have been given a .h extension to identify them as something different from the .c program files.

179)What do the header files usually contain?
Header files contain Preprocessor directives like #define, structure, union and enum declarations, typedef declarations, global variable declarations and external function declarations. You should not write the actual code (i.e., function bodies) or global variable definition (that is defining or initializing instances) in header files. The #include directive should be used to pull in header files, not other '.c' files.

180) Will it result into an error if a header file is included twice? [Yes/No]
Yes, unless the header file has taken care to ensure that if already included it doesn't get included again.

181) How can a header file ensure that it doesn't get included more than once?
All declarations must be written in the manner shown below. Assume that the name of the header file is ‘funcs.h’.

```
/* funcs.h */
#ifndef _FUNCS
#define _FUNCS
/* all declarations would go here */
#endif
```

Now if we include this file twice as shown below, it will get included only once.

```
#include "funcs.h"
#include "funcs.h"
int main( )
{
/* some code */
return 0 ;
}
```

182)On using #include where do the header files get searched?
If #included using < > syntax, the files get searched in the predefined include path. It is possible to change the predefined include path. If #included with the " " syntax, in addition to the predefined include path, the files also get searched in the current directory (usually the directory from which you invoked the compiler).

183) Can you combine the following two statements into one?
char *p ;
p = (char *) malloc (100) ;
char *p = (char *) malloc (100);
Note that the typecasting operation can be dropped completely if this program is built using gcc compiler.

184) Are the expressions *ptr++ and ++*ptr same?
No. *ptr++ increments the pointer and not the value pointed by it, whereas ++*ptr increments the value being pointed to by ptr and not ptr.

185) Can you write another expression which does the same job as ++*ptr does?
(*ptr)++

186)What would be the equivalent pointer expression for referring the array element a[i][j][k][1]?
((*(*(a+1)+j)+k)+1)

187)Where can one think of using pointers?
At lot of places, some of which are:
Accessing array or string elements
In passing big objects like arrays, strings and structures to functions
Dynamic memory allocation
Call by reference
Implementing linked lists, trees, graphs and many other data structures

188) How will you declare an array of three function pointers where each function receives two ints and returns a float?
float (*arr[3]) (int, int) ;

189) Is the NULL pointer same as an uninitialized pointer? [Yes/No]
No. An uninitialized pointer (often called a wild pointer) contains some garbage address. A pointer that contain NULL indicates that it is a valid pointer but it is not pointing to anything right now.

190) In which header file is the NULL macro defined?
In files "stdio.h" and "stddef.h".

191) Is there any difference between the following two statements?
char *p = 0 ;
char *t = NULL ;
No. NULL is #defined as 0 in the 'stdio.h' file. Thus, both p and t are null pointers.

192) What is a null pointer?
For each pointer type (like say a **char** pointer) C defines a special pointer value, which is guaranteed not to point to any object or function of that type. Usually, the null pointer constant used for representing a null pointer is the integer 0.

193) What's the difference between a null pointer, a NULL macro, the ASCII NUL character and a null string?
A null pointer is a pointer, which doesn't point anywhere. A NULL macro is used to represent the null pointer in source code. It has a value 0 associated with it.
The ASCII NUL character has all its bits as 0 but doesn't have any relationship with the null pointer.
The null string is just another name for an empty string "".

194) Is there any difference in the following two statements?
char *ch = "Nagpur" ;
char ch[ ] = "Nagpur" ;
Yes. In the first statement, the character pointer ch stores the address of the string "Nagpur". The pointer ch can be made to point to some other character string (or even nowhere). The second statement, on the other hand, specifies that space for 7 characters be allocated and that the name of the location is ch. Thus, it specifies the size as well as initial values of the characters in array ch.

195) When are char a[ ] and char *a treated as same by the compiler?
When using them as formal parameters while defining a function.

196) What is the difference in the following declarations?
char *p = "Samuel" ;
char a[ ] = "Samuel" ;
Here a is an array big enough to hold Samuel and the '\0' following the it. Individual characters within the array can be changed but the address of the array will remain constant.
On the other hand, p is a pointer, initialized to point to a string constant. The pointer p may be modified to point to another string, but if youattempt to modify the string at which
p is pointing the result is undefined.

197)While handling a string do we always have to process it character-bycharacter or there exists a method to process the entire string as one unit.
A string can be processed only on a character-by-character basis.

198) What is the similarity between a structure, union and an enumeration?
All of them let you define new data types.

199) Can a structure contain a pointer to itself?
Yes. Such structures are known as self-referential structures. They are commonly used in declaration of a node while implementing a data structure like linked list or binary tree.

200) How are structure passing and returning implemented by the compiler?

When structures are passed as arguments to functions, the entire structure is pushed on the stack. For big structures this is an extra overhead. This overhead can be avoided by passing pointers to structures instead of actual structures. To return structures a hidden argument generated by the compiler is passed to the function. This argument points to a location where the returned structure is copied.