EXAMEN FINAL CISCO ESTRUCTURA DE DATOS (73%)

Question 1

The following symbol may be (in certain circumstances) taken as an equivalent of:

```
:>
```

- a bracket
- a tilde
- a hash
- ◉ a parenthesis

Question 2

Which of the given examples is contemporaty equivalent of the obsolete function declaration?

```
void f(x) float x; { return x * x; }
```

- ◉ `float f(float x) { return x * x; }`
- `void f() float x; { return x * x; }`
- `void f(float) x; { return x * x; }`
- `void f(x) float; { return x * x; }`

Question 3

The *ellipsis* (...) used in a function header, may be:

- preceded by parameter of pointer type
- surrounded by brackets
- the first element on parameters list
- ◉ the only element on parameters list

Question 4

The so-called "**calling convention**" determines:

- ◉ the way in which the arguments are put on and cleared off the stack
- the way in which the programmer builds functions' interface
- the possible function names
- the possible argument names

Question 5

A macro named `va_arg`, comes from:

- (◉) `<stdarg.h>`
- `<vargs.h>`
- `<varg.h>`
- `<args.h>`

Question 6

In the argument passing convention named "**stdcall**":

- the invoker cleans the stack after return
- (◉) the invokee cleans the stack before return
- the stack is cleaned periodically by OS
- the stack cleans its state itself

Question 7

What is the expected output of the following code?

```c
#include <stdarg.h>
#include <stdio.h>
int f(int n, ...) {
    va_list list;
    va_start(list,n);
    char c;
    while(va_arg(list,int) != 0)
    c = va_arg(list,int);
    va_end(list);
    return c;
}
}

    int main(void) {
    printf("[%c]\n", f('a','b','c','\0'));
    return 0;
}

```

- (◉) c
- b
- a
- an empty line

Question 8

Each invocation of `va_start()` must be matched by:

- (●) exactly one matching `va_end()` invocation
- ( ) at least one matching `va_end()` invocation
- ( ) not more than two matching `va_end()` invocations
- ( ) not more than one matching `va_end()` invocation

Question 9

The "*endline translation*" occurs when the file is opened with `O_TEXT` flag and takes place during:

- (●) reads and writes
- ( ) writes only
- ( ) reads only
- ( ) `lseek()` function invocation

Question 10

Symbolic links are available in:

- ◉ both Linux/Unix and MS Windows
- neither in Linux/Unix nor MS Windows
- MS Windows only
- Linux/Unix only

Question 11

Which of the expressions can be used to discover that `read()` invocation reached end of file?

- ◉ `read(fd, t, sizeof(t)) == 0`
- `read(fd, t, sizeof(t)) > 0`
- `read(fd, t, sizeof(t)) == sizeof(t)`
- `read(fd, t, sizeof(t)) < 0`

Question 12

The file descriptor of value equal to `1`:

- is connected to `stdin`
- (●) is connected to `stdout`
- is connected to `stderr`
- is not connected and may be freely used

Question 13

The `lseek()` is able to move the file pointer beyond file's end.

- (●) always true
- always false
- true if the file is not empty
- true if the file is empty

Question 14

The "`st_dev`" field of "`struct stat`" in MS Windows environment reflects:

- the drive number
- device location
- device vendor's name
- ◉ 32 bit long device identifier

Question 15

What should be placed instead of `<?>` to make the following snippet work properly?

```
1  if( <?> access("input", W_OK) ) {
2      puts("File could not be written");
3      exit(1);
4  }
5
```

- ◉ `0 ==`
- `NULL ==`
- `0 <`
- `0 >`

Question 16

To store UNICODE code points the UTF-8 uses:

- (●) varying number of bits
- ( ) 16 bits
- ( ) 32 bits
- ( ) 8 bits

Question 17

The `wcscpy()` function is:

- (●) a "wide" version of `strcpy()`
- ( ) used to invoke Python programs
- ( ) used to invoke C# programs
- ( ) not a member of any standard library

Question 18

What is the expected output of the following code?

```c
#include <string.h>
#include >stdio.h>
int main(void) {
    char p[] = "10101", *q;
    int i = 0;
    q = strtok(p,"0");
    while(q) {
    i++;
    tq = strtok(NULL,"0");
     }
    printf("%d\n",i);
    return 0;
}
```

- 3
- 1
- 2
- the code falls in an infinitive loop

Question 19

What is the expected output of the following code?

```c
#include <string.h>
#include >stdio.h>
int main(void) {
    int i, *p = &i;
    printf("%d\n",memcmp(&i,&p,4));
    return 0;
}
```

- a number most probably different from zero
- a number always less than zero
- a number always greater than zero
- the output is unpredictable

Question 20

The result of the following expression:

```
int n = '9' - '1';

```

- depends on encoding system used by specific hardware platform
- is always `6`
- is always `7`
- ◉ is always `8`

Question 21

The memory block allocated by `malloc()` function:

- ◉ is filled with random values
- is filled with `0xdeadbeef`
- is filled with `0xFF`
- is filled with zeros

Question 22

The value of the following expression:

```
sizeof(wint_t) < sizeof(wchar_t)

```

- is always `0`
- is always `1`
- cannot be determined
- ◉ depends on target platform

Question 23

What is possible output of the following program?

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    puts(*argv);
    return 0;
}
```

- (●) a line containing program name
- ( ) an empty line
- ( ) a random string
- ( ) zero

Pantalla de Jacky

Question 24

The string literal passed to `_Pragma` keyword is the subject of:

- (●) destringization
- ( ) dereferencing
- ( ) defragmentation
- ( ) deliteration

Question 25

Which of the following `main()` function's headers is not valid?

- ( ) `int main(int argc, char *argv[], char ***env[])`
- (●) `int main(int argc, char *argv[], char **env)`
- ( ) `int main(int argc, char *argv[], char *env[])`
- ( ) `int main(int argc, char *argv[])`

Question 26

The `_Bool` keyword, introduced by C11, denotes a type being an equivalent of:

- unsigned `int`
- unsigned `char`
- ◉ boolean
- `void`

Question 27

Which of the following statements is true?

- ◉ trigraphs are recognized inside string literals, digraphs aren't
- trigraphs are not recognized inside string literals, digraphs are
- both trigraphs and digraphs are recognized inside string literals
- both trigraphs and digraphs are not recognized inside string literals

Question 28

A variadic function is a function which:

- ◉ accepts varying number of arguments
- accepts more than one invocation at a time
- changes its name during program execution
- changes its address during program execution

Question 29

The following symbol may be (in certain circumstances) taken as an equivalent of:

```
??'
```

- a caret
- (●) a tilde
- a hash
- a parenthesis

Pantalla de Jacky

Question 30

According to C11 standard, the `__func__` symbol is a:

- parameterless macro
- parameterized macro
- (●) parameterless function
- global variable

Question 31

The `spawn()` family functions:

- always return to invoker
- return to invoker in case of success
- (●) return to invoker in case of error
- never return to invoker

Question 32

A working process is identified by:

- ◉ its unique PID
- name of program used to start the process
- source device
- user's name

Question 33

The term "**lock a mutex**" being used in P-threads terminology:

- ◉ means the same as "obtain a mutex" in MS Windows environment
- means the same as "release a mutex" in MS Windows environment
- means the same as "close a mutex" in MS Windows environment
- is not used in any other environment

Assuming that the code was compiled and ran in Unix/Linux environment what is its expected output?

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int x = 0;
int main(void) {
    if(fork()) {
    wait(NULL);
                printf("%d", x);
    return 0;
    } else {
                x++;
    return 0;
     }
}

```

- `0`
- `1`
- `2`
- an empty line

Question 35

The `exec()` family functions:

- return to invoker in case of error
- never return to invoker
- return to invoker in case of success
- always return to invoker

Question 36

The shortest part of a double value is:

- ◉ sign
- exponent
- significand
- all parts have equal sizes

Question 37

The length of a double typed variable:

- ◉ is defined by IEEE754 standard
- is defined by the "C" language standards
- is user defined
- is hardware dependent

Question 38

The `FP_SUBNORMAL` flag marks a float value that:

- ◉ cannot be normalized
- has been successfully normalized
- has to be normalized
- exceeds any of IEEE defined limits

Question 39

Removing the letter '`f`' from both literals used in the following expression:

- will change the value printed to `stdout`
- ◉ won't change the value printed to `stdout`
- will cause runtime error
- will make the expression syntactically incorrect

Question 40

To input a 64 bit unsigned int value as hex number some would use the following portable specifier:

- ◉ `SCNx64`
- `SCNuMAX`
- `PRIx64`
- `PRIuMAX`

Question 41

The normalized float value:

- ◉ has the highest significand's bit set to `1`
- has the highest significand's bit set to `0`
- has the highest exponent's bit set to `0`
- has the highest exponent's bit set to `0`

Question 42

The GMP library uses the `mpf_t` type to represent:

- ◉ multiprecision float values
- multiprecision double values
- multiprecision rational values
- multiprecision integer values

Question 43

The UDP protocol:

- ◉ provides unreliable communication channel
- does not provide any communication channel
- is only able to transmit datagrams
- provides reliable communication channel

Question 44

The so-called "**network order**" is identical to:

- ◉ big-endian order
- little-endian order
- native hardware platform order
- target hardware platform order

Question 45

The `MSG_PEEK` flag is used in connection with:

- (●) `recvfrom()`
- ( ) `bind()`
- ( ) `accept()`
- ( ) `socket()`

Question 46

The regular IP address consists of:

- (●) 4 octets
- ( ) 8 octets
- ( ) 1 octets
- ( ) 2 octets

Question 47

Using WinSock, comparing to the BSD sockets:

- [■] requires some additional initializations and terminations
- [■] requires some addition initializations
- [■] requires some additional terminations
- [ ] is exactly the same

Question 48

Which of the following is a proper `bind()` function header?

- ◉ `int bind(int sockfd, struct sockaddr *addr, socklen_t addrlen);`
- `int bind(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`
- `int bind(int sockfd, struct sockaddr addr, socklen_t addrlen);`
- `int bind(int sockfd, struct sockaddr addr, socklen_t *addrlen);`

Question 49

Resolving a host's name based on given IP address can be done by invoking:

- ◉ `gethostbyname()`
- `getIP()`
- `getservbyname()`
- `getIPaddr()`

Question 50

The following declarator declares the fun symbol as:

```
1  void (*fun)(int*, int*);
2
```

- ◉ a pointer to function returning void
- a pointer to function returning pointer to void
- a void pointer
- a pointer to void

Question 51

The `setjmp()` function performs its return sequence:

- usually twice
- more than twice
- ◉ always once
- not more than once

Question 52

The following snippet:

```
int f(int i) {
    i++;
    if(i == 0) goto go_to;
    return i;
}

    int main(void) {
    f(0);
go_to:
    return 0;
}

```

- will cause infinite loop
- will cause runtime error
- ◉ will cause compilation error

Question 53

Select the true statement:

- (selected) a variable may be `const` and `volatile` at the same time
- a `const`-`volatile` variable may be explicitly modified in the code containing its declaration
- a `const`-`volatile` variable must not be implicitly modified by the background process
- a compiler will try to store the `const`-`volatile` variable in CPU registers

Question 54

The following snippet:

```
1  int const v = 1;
2  int const * const p = &v;
3
```

- (selected) is fully correct
- will cause compiler error
- may cause compiler warning
- may cause runtime error

Question 55

The `volatile` specifier:

- [x] forces the compiler to fetch the variable from memory each time it is read
- [ ] is obsolete
- [ ] tells the compiler to not use the variable
- [ ] suggests the compiler to store the variable in one of CPU registers

Question 56

A syntax used in the following snippet is named:

```
1 | double vec[3] = { [0] = 1.23, [2] = 3.21 };
2 |
```

- [x] designated initializer
- [ ] enumerated initializer
- [ ] indexed initializer
- [ ] inverted initializer