**17. ¿Cómo cree que se calcularía más rápido la operación "a = b * c" suponiendo que el valor de c es 5?**

a) a = c * b;
b) a = b * c;
c) a = b + (b << 2);
d) a = b + b + b + b + b;

**18. Dado el siguiente código y suponiendo el vector v inicializado, ¿qué opción es verdadera?**

```
for(int i = 0; i <1000; ++i)
{
        if ((v[i] % 3) == 0)
                foo(v[i]);
        else
                switch((v[i] % 3))
                {
                        case 1: foo(v[i] + 2); break;
                        case 2: foo(v[i] + 1); break;
                }
}
```

a) Los valores contenidos en v no afectan a la velocidad de ejecución
b) La ejecución finaliza antes si v contiene muchos múltiplos de 3
c) Sólo el desarrollo de bucle puede servir para optimizar el código
d) La ejecución finaliza antes si v no contiene ningún múltiplo de 3

**19. ¿Cuál de las siguientes formas de implementar el mismo algoritmo cree más rápida?**

```
const int N = 5000; REP = 40000;
int R[REP + 1];
struct S { int a, b; } s[N];
```

d)
```
int sa = 0, sb = 0;
for (int i = 0; i <N; ++i)
{
        sa += s[i].a;
        sb += s[i].b;
}
sa *= 2;
sb *= 3;
for (int ii = 1; ii<=REP; ++ii)
        R[ii] = std::min(sa + N * ii, sb – N * ii)
```

**20. ¿Cuál de las siguientes versiones de una función que multiplica un entero por 6 cree que se obtendrá al compilar con optimización en espacio (-Os)?**

```
int f(int x)
{
        return x * 6;
```

}

a) 0x401106 <+0>: lea (%rdi,%rdi,2), %eax 0x401109 <+3>: add %eax,%eax
0x40110b <+5>: retq

**25. ¿Qué forma de ordenar las opciones case que aparecen en una sentencia switch ofrece mejores prestaciones?**

a) deben aparecer primero las opciones que tengan una mayor probabilidad de ser ciertas

b) el orden de aparición es indiferente

c) deben aparecer primero las opciones que contengan un mayor número de instrucciones

d) deben aparecer primero las opciones que contengan un menor número de instrucciones

**26. ¿Cómo ordenaría los índices del siguiente algoritmo de multiplicación de matrices?**

**int a[100][100], b[100][100], c[100][100];**

a)
```
for (int i = 0; i  < 100; ++i)
        for (int k = 0; k < 100; ++k)
                for(int j = 0; j < 100; ++j)
                        a[i][j] += b[i][k] * c[k][j];
```

b)
```
for (int j = 0; j  < 100; ++j)
        for (int i = 0; i < 100; ++i)
                for(int k = 0; k < 100; ++k)
                        a[i][j] += b[i][k] * c[k][j];
```

c)
```
for (int i = 0; i  < 100; ++i)
        for (int j = 0; j < 100; ++j)
                for(int k = 0; k < 100; ++k)
                        a[i][j] += b[i][k] * c[k][j];
```

d)
```
for (int k = 0; k  < 100; ++k)
        for (int j = 0; j < 100; ++j)
                for(int i = 0; i < 100; ++i)
                        a[i][j] += b[i][k] * c[k][j];
```

**27. ¿Cómo cree que se calcularía más rápido la operación "a = b * c" suponiendo que el valor de c es 5?**

a) a = b + (b << 2);

b) a = c * b;

c) a = b + b + b + b + b;

d) a = b * c;

**28. ¿Cuál cree que es la implementación óptima del siguiente algoritmo?**

```
int f(int n)
{
        int s = 0;
```

```
        for (int i = 0; i < n; ++i)
                s += i % 5 + 1;
        return s;
}
```

a) f(int):
```
        leal    0(,%rdi,4), %eax
        ret
```
b) f(int):
```
        xorl    %ecx, %ecx
        xorl    %r8d, %r8d
        movl    $5, %esi
.L3:
        cmpl    %edi, %ecx
        jge     .L1
        movl    %ecx, %eax
        incl    %ecx
        cltd
        idivl   %esi
        leal    1(%r8,%rdx), %r8d
        jmp     .L3
.L1:
        movl    %r8d, %eax
        ret
```
c) f(int):
```
        leal    (%rdi,%rdi,2), %eax
        ret
```
d) ninguna otra respuesta es correcta

**29. ¿Cuál de las siguientes formas de implementar el mismo algoritmo cree más rápida?**

```
const int N = 5000 , REP = 40000;
int R[REP + 1];
struct S { int a, b; } s[N];
```

e)
```
int sa = 0, sb = 0;
for (int i = 0; i <N; ++i)
{
        sa += s[i].a;
        sb += s[i].b;
}
sa *= 2;
sb *= 3;
for (int ii = 1; ii<=REP; ++ii)
        R[ii] = std::min(sa + N * ii, sb – N * ii)
```

**30. ¿Qué código cree que calculará de forma correcta y en menor tiempo el producto de dos matrices en un sistema multiprocesador? Suponga matrices cuadradas, c inicializada a cero y N muy grande**

Int a[N][N], b[N][N], c[N][N];

a)
```
for (int i=0; i<N; ++i)
        #pragma omp parallel for
        for (int j=0; j<N; ++j)
                for (int k=0; k<N; ++k)
                        c[i][j] += a[i][k] * b[k][j];
```

b)
```
for (int i=0; i<N; ++i)
        #pragma omp parallel for
        for (int j=0; j<N; ++j)
                for (int k=0; k<N; ++k)
                        #pragma omp atomic
                        c[i][j] += a[i][k] * b[k][j];
```

c)
```
for (int i=0; i<N; ++i)
        #pragma omp parallel for
        for (int j=0; j<N; ++j)
                for (int k=0; k<N; ++k)
                        #pragma omp critical
                        c[i][j] += a[i][k] * b[k][j];
```

d)
```
for (int i=0; i<N; ++i)
        for (int j=0; j<N; ++j)
                for (int k=0; k<N; ++k)
                        c[i][j] += a[i][k] * b[k][j];
```

17. ¿Cuál de las siguientes afirmaciones es correcta?

a) La optimización de código siempre debe realizarse en lenguaje ensamblador

b) Hay optimizaciones que son aplicables a cualquier procesador

c) Ninguna otra respuesta es correcta

d) el proceso de optimización se debe realizar siempre el final del desarrollo de la aplicación

19. Escoja la mejor forma de calcular el valor de la sumatoria de la siguiente estructura

```
const int N = 1000;
struct S { int a[N], b[N]; } s
int sum = 0;
```

b) `for (int i = 0; i < N; i += 2){ sum += s.a[i] + s.a[i + 1]; sum += s.b[i] + s.b[i + 1]; }`

20. ¿Cómo cree que implementará el compilador una función que multiplica un entero por 11 al compilar con optimización máxima (-O3)?

```
int function_f(int x)
{
        return x * 11;
```

}

b) 0x401120 <+0>: lea (%rdi,%rdi;4), %eax 0x401123 <+3>: lea (%rdi, %rax,2), %eax 0x401126 <+6>: retq