Início	quarta, 16 de março de 2022 às 08:46
Estado	Prova submetida
Data de submissão:	domingo, 20 de março de 2022 às 13:28
Tempo gasto	4 dias 4 horas
Nota	**100** do máximo 100

Pergunta 1

Correta Pontuou 20 de 20

Write a C++ function `int adigits(int a, int b, int c)` that receives 3 integers, each one with a single decimal digit and returns the highest integer number that can be assembled with the 3 digits given as parameters.

Por exemplo:

Teste	Resultado
`cout << adigits(4, 2, 5);`	542
`cout << adigits(9, 1, 9);`	991
`cout << adigits(1, 2, 3);`	321
`cout << adigits(1, 0, 0);`	100

Resposta: (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

```
int adigits(int a, int b, int c){
    if (a >= b && a >= c){
        if (b >= c) return a * 100 + b * 10 + c;
        if (c > b) return a * 100 + c * 10 + b;
    }
    if (b >= a && b >= c){
        if (a >= c) return b * 100 + a * 10 + c;
        if (c > a) return b * 100 + c * 10 + b;
    }
    if (b>a){
        return c * 100 + b * 10 + a;
    }
    return c * 100 + a * 10 + b;
}
```

	Teste	Esperado	Recebido	
✔	`cout << adigits(4, 2, 5);`	542	542	✔
✔	`cout << adigits(9, 1, 9);`	991	991	✔
✔	`cout << adigits(1, 2, 3);`	321	321	✔
✔	`cout << adigits(1, 0, 0);`	100	100	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
int adigits(int d1, int d2, int d3) {
  if (d1 >= d2 and d1 >= d3) {
    d1 = d1 * 100;
    if (d2 >= d3) {
      d2 = d2 * 10;
    }
    else d3 = d3 * 10;
  }
  else if (d2 >= d1 and d2 >= d3) {
    d2 = d2 * 100;
    if (d1 >= d3) {
      d1 = d1 * 10;
    }
    else d3 = d3 * 10;
  }
  else { // (d3 >= d1 and d3 >= d2)
    d3 = d3 * 100;
    if (d1 >= d2) {
```

```
19        d1 = d1 * 10;
20      }
21      else d2 = d2 * 10;
22    }
```

Correta

Nota desta submissão: 20/20

Pergunta 2

Correta Pontuou 20 de 20

Write a C++ function `void advance(int delta, int& d, int& m, int& y)` to update a date, implicitly defined by `d`/`m`/`y`, by `delta` days.

You can assume that delta is greater or equal to 0 and that the values of `d`, `m`, and `y` define a valid date when the function is called. On exit, `d`, `m`, and `y` should represent an elapse of delta days over the original date.

Note that you should account for leap years: `y` is a leap year ("bissexto'') if it is divisible by 4, except when it is divisible by 100 but not 400 (e.g., 2004 and 2000 are leap years, but 2100 is not).

Por exemplo:

Teste	Resultado
`int d = 1, m = 1, y = 2022; advance(0, d, m, y); cout << d << ' ' << m << ' ' << y;`	`1 1 2022`
`int d = 1, m = 10, y = 2022; advance(10, d, m, y); cout << d << ' ' << m << ' ' << y;`	`11 10 2022`
`int d = 1, m = 8, y = 2022; advance(31, d, m, y); cout << d << ' ' << m << ' ' << y;`	`1 9 2022`
`int d = 25, m = 1, y = 2000; advance(4, d, m, y); cout << d << ' ' << m << ' ' << y;`	`29 1 2000`
`int d = 1, m = 1, y = 2024; advance(365, d, m, y); cout << d << ' ' << m << ' ' << y;`	`31 12 2024`

Resposta: (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

```
bool is_leap_year(int y) {
    return (y % 4 == 0 && y % 100 != 0) || y % 400 == 0;
}

int days_in_month(int m, int y) {
    int d;
    if (m == 2)
        d = is_leap_year(y) ? 29 : 28;
    else if (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12)
        d = 31;
    else
        d = 30;
    return d;
}

void advance(int delta, int& d, int& m, int& y){
    for (int i = 0; i < delta; i++){
        int days_month = days_in_month(m, y);
        if (d < days_month){
            d++;
        }
        else if (m < 12){
```

	Teste	Esperado	Recebido	
✔	`int d = 1, m = 1, y = 2022; advance(0, d, m, y); cout << d << ' ' << m << ' ' << y;`	`1 1 2022`	`1 1 2022`	✔
✔	`int d = 1, m = 10, y = 2022; advance(10, d, m, y); cout << d << ' ' << m << ' ' << y;`	`11 10 2022`	`11 10 2022`	✔
✔	`int d = 1, m = 8, y = 2022; advance(31, d, m, y); cout << d << ' ' << m << ' ' << y;`	`1 9 2022`	`1 9 2022`	✔
✔	`int d = 25, m = 1, y = 2000; advance(4, d, m, y); cout << d << ' ' << m << ' ' << y;`	`29 1 2000`	`29 1 2000`	✔
✔	`int d = 1, m = 1, y = 2024; advance(365, d, m, y); cout << d << ' ' << m << ' ' << y;`	`31 12 2024`	`31 12 2024`	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
void advance(int& d, int& m, int& y) {
  int ldm; // last day in month
  switch(m) {
    case 2:
```

```
      ldm = y % 400 == 0 || (y % 100 != 0 && y % 4 == 0) ? 29 : 28;
      break;
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      ldm = 31;
      break;
    default:
      ldm = 30;
  }
  if (d == ldm) {
    // Last day in month
    d = 1;
    if (m == 12) {
      // Last day in December
      y++;
      m = 1;
    } else {
      m++;
    }
```

Correta

Nota desta submissão: 20/20

Pergunta 3

Correta Pontuou 20 de 20

A *Mersenne number* is a number of the form $M_n = 2^n - 1$ for some positive integer n.

Write a C++ function `bool is_mersenne_number(unsigned long n)` that returns `true` (`1`) if `n` is a Mersenne number and `false` (`0`) otherwise.

Hint: The Mersenne numbers consist of all 1s in base-2.

You cannot use the `pow` or other `cmath` functions.

Por exemplo:

Teste	Resultado
`cout << is_mersenne_number(0);`	`0`
`cout << is_mersenne_number(3);`	`1`
`cout << is_mersenne_number(4);`	`0`
`cout << is_mersenne_number(65535);`	`1`
`cout << is_mersenne_number(18446744073709551614UL);`	`0`

Resposta: (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

```
bool is_mersenne_number(unsigned long n){
    if (n <= 0) return false;
    while(n > 0){
        if (n%2 == 0) return false;
        n = n / 2;
    }
    return true;
}
```

	Teste	Esperado	Recebido	
✔	`cout << is_mersenne_number(0);`	`0`	`0`	✔
✔	`cout << is_mersenne_number(3);`	`1`	`1`	✔
✔	`cout << is_mersenne_number(4);`	`0`	`0`	✔
✔	`cout << is_mersenne_number(65535);`	`1`	`1`	✔
✔	`cout << is_mersenne_number(18446744073709551614UL);`	`0`	`0`	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
bool is_mersenne_number(unsigned long n) {
  do {
    if (n % 2 == 0) { // could also be (n & 1) == 0
     return false;
    }
    n /= 2; // could also be n >>= 1;
  } while (n != 0);
  return true;
}
```

```
10
11 // another solution
12 bool is_mersenne_number_(unsigned long n) {
13   unsigned long mn = ~0; // = FF...FF
14   while (mn != 0) {
15     if (mn == n) {
16       return true;
17     }
18     mn >>= 1;
19   }
20   return false;
21 }
22
```

Correta

Nota desta submissão: 20/20

Pergunta 4

Correta Pontuou 20 de 20

A quadratic equation of the form $ax^2 + bx + c = 0$ in the domain of real numbers is known to have 0, 1, or 2 solutions according to the value of $\Delta = b^2 - 4ac$:

- no solutions if $\Delta < 0$
- 1 solution $x_1 = \frac{-b}{2a}$ if $\Delta = 0$
- 2 solutions $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ and $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$ when $\Delta > 0$

Write a C++ function to solve the quadratic equation for the case where a, b, c are integer coefficients, with $a \neq 0$:

```
int solve_eq(int a, int b, int c, double& x1, double& x2)
```

The function should return the number of solutions for the equation (0, 1, or 2) and, in the case there are solutions, assign the corresponding values to `x1` and `x2` as described above. `x1` and `x2` should be set to NAN defined in header `cmath` when there are no corresponding solutions (x2 = NAN when $\Delta = 0$; and x1 = x2 = NAN when $\Delta < 0$).

To perform the square root computation, you should use the function `std::sqrt`, defined in header `cmath`.

Note: the test cases illustrate solutions with a precision of 7 decimal digits.

Por exemplo:

Teste	Resultado
`double x1, x2; int r = solve_eq(1, 1, 0, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`2 -1.0000000 0.0000000`
`double x1, x2; int r = solve_eq(1, -2, 1, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`1 1.0000000 nan`
`double x1, x2; int r = solve_eq(1, -1, 0, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`2 0.0000000 1.0000000`
`double x1, x2; int r = solve_eq(16, -8, 1, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`1 0.2500000 nan`
`double x1, x2; int r = solve_eq(3, 4, -2, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`2 -1.7207592 0.3874259`

Resposta: (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

```
#include <iostream>
#include <limits>
#include <cmath>

int solve_eq(int a, int b, int c, double& x1, double& x2){
    double delta = b * b - 4 * a * c;
    if (delta < 0){
        x1 = std::numeric_limits<double>::quiet_NaN();
        x2 = std::numeric_limits<double>::quiet_NaN();
        return 0;
    }
    else if(delta == 0){
        x1 = -b / (2.0*a);
        x2 = std::numeric_limits<double>::quiet_NaN();
        return 1;
    }
    x1 = (-b - sqrt(delta)) / (2*a);
    x2 = (-b + sqrt(delta)) / (2*a);
    return 2;
}
```

	Teste	Esperado	Recebido	
✔	`double x1, x2; int r = solve_eq(1, 1, 0, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`2 -1.0000000 0.0000000`	`2 -1.0000000 0.0000000`	✔
✔	`double x1, x2; int r = solve_eq(1, -2, 1, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	`1 1.0000000 nan`	`1 1.0000000 nan`	✔

	Teste	Esperado	Recebido	
✔	`double x1, x2; int r = solve_eq(1, -1, 0, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	2 0.0000000 1.0000000	2 0.0000000 1.0000000	✔
✔	`double x1, x2; int r = solve_eq(16, -8, 1, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	1 0.2500000 nan	1 0.2500000 nan	✔
✔	`double x1, x2; int r = solve_eq(3, 4, -2, x1, x2); cout << fixed << setprecision(7) << r << ' ' << x1 << ' ' << x2;`	2 -1.7207592 0.3874259	2 -1.7207592 0.3874259	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
#include <cmath>

int solve_eq(int a, int b, int c, double& x1, double& x2) {
  int delta = b * b - 4 * a * c;
  if (delta < 0) {
    x1 = x2 = NAN;
    return 0;
  }
  if (delta == 0) {
    x1 = - b / (2.0 * a);  // We must use 2.0, not 2
    x2 = NAN;
    return 1;
  }
  double d = sqrt(delta);
  x1 = (-b - d) / (2.0 * a);
  x2 = (-b + d) / (2.0 * a);
  return 2;
}

// private test(s (1000 points each)
// double x1, x2; int r = solve_eq(2, -2, 3, x1, x2); cout << fixed << setprecision(7) << r << '
// double x1, x2; int r = solve_eq(1, 4, 17, x1, x2); cout << fixed << setprecision(7) << r << '
```

Correta

Nota desta submissão: 20/20

Pergunta 5

Correta Pontuou 20 de 20

Bell numbers $B_0, B_1, B_2, \ldots$ count the possible partitions of a set, i.e., the number or partitions for a set with $n \ge 0$ elements is given by B_n, where $B_0 = B_1 = 1$ and for $n \ge 2$

$$ B_n = \sum_{k=0}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array}\right) B_k $$

Note that binomial coefficients are used in the definition above. An efficient method to compute individual binomial coefficients is given by the *multiplicative formula*:

$$ \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n \times (n - 1) \times \ldots \times (n - k + 1)}{k \times (k - 1) \times \ldots \times 1} $$

Write a C++ function `unsigned long bell(unsigned long n)` to compute Bell numbers.

Por exemplo:

Teste	Resultado
`cout << bell(0);`	1
`cout << bell(1);`	1
`cout << bell(2);`	2
`cout << bell(3);`	5
`cout << bell(4);`	15

Resposta: (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

```
unsigned long factorial(unsigned long n){
    if (n == 0 || n == 1) return 1;
    return n * factorial(n-1);
}

unsigned long binomial_coefficients(unsigned long n, unsigned long k){
    unsigned long fac_n = 1;
    for(unsigned long i = n; i >= n - k + 1; i--){
        fac_n *= i;
    }
    return fac_n / factorial(k);
}

unsigned long bell(unsigned long n){
    if(n == 0 || n == 1) return 1;
    unsigned long bn = 0;
    for(unsigned long k = 0; k < n; k++){
        bn += bell(k) * binomial_coefficients(n-1, k);
    }
    return bn;
}
```

	Teste	Esperado	Recebido	
✔	`cout << bell(0);`	1	1	✔
✔	`cout << bell(1);`	1	1	✔
✔	`cout << bell(2);`	2	2	✔
✔	`cout << bell(3);`	5	5	✔
✔	`cout << bell(4);`	15	15	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
unsigned long bc(unsigned long n, unsigned long k) {
  unsigned long num = 1;
  unsigned long den = 1;
  for (unsigned long i = 1; i<= k; i++) {
     num *= (n - i + 1);
```

```
6       den *= i;
7     }
8     return num/den;
9   }
10
11  unsigned long bell(unsigned long n) {
12    if (n <= 1) return 1;
13    unsigned long int r = 0;
14    for (unsigned long k = 0; k < n; k++) {
15      r += bc(n-1, k) * bell(k);
16    }
17    return r;
18  }
19
20  // private test(s (1000 points each)
21  // cout << bell(6); => 203
22  // cout << bell(7); => 877
```

Correta

Nota desta submissão: 20/20

◀ T01 08/03

Ir para...

T02 15/03 ▶