Início	quarta, 30 de março de 2022 às 08:44
Estado	Prova submetida
Data de submissão:	quinta, 31 de março de 2022 às 22:31
Tempo gasto	1 dia 13 horas
Nota	**100** do máximo 100

Pergunta 1

Correta Pontuou 20 de 20

Consider the code given in alist.cpp, supporting the definition of "array lists" (lists represented using an array) containing elements of type `int`. Type `alist` defines member fields `size` of type `int`, expressing the number of elements in an array list, and `elements`, a pointer to a dynamically allocated array of elements of type `int` containing the elements and length equal to `size`.

```
struct alist {
  int size;       // Number of elements.
  int* elements;  // List elements.
};
```

The following functions are defined in association to `alist`:

- `alist* build(int n, int v[])`: create a new array list with `n > 0` elements with initial values given in array `v`.
- `alist* empty()`: create a new array list that is initially empty, i.e., has no elements — for an empty list `al`, `al->size` is set to `0` and `al->elements` is set to `nullptr`.
- `destroy(alist* al)`: releases the memory allocated to `al`; and
- `print(alist* al)`: prints the elements in `al`.

Define a new function `void append(alist* a, const alist* b)` such that a call to `append(a, b)` appends all the elements in list `b` to list `a`. Note that, except when `a` or `b` represent the empty list, the internal array used by `a` must be replaced by a new array with size `a->size + b->size`. Moreover, `b` should remain unchanged.

You need to include `alist.cpp` file in your code i.e. `#include "alist.cpp"`.

You cannot use any C++ library classes or functions, including `vector`, `list`, or `string`.

Por exemplo:

Teste	Resultado

`alist* a = empty();`
`const alist* b = empty();`
`append(a, b);`
`print(a);`
`destroy(a);`
`destroy(b);`
	`[ ]`

`alist* a = empty();`
`const int nB = 2;`
`int eB[nB] = { 1, 2 };`
`const alist* b = build(nB, eB);`
`append(a, b);`
`print(a);`
`destroy(a);`
`destroy(b);`
	`[ 1 2 ]`

`const int nA = 2;`
`int eA[nA] = { 1, 2 };`
`alist* a = build(nA, eA);`
`const alist* b = empty();`
`append(a, b);`
`print(a);`
`destroy(a);`
`destroy(b);`
	`[ 1 2 ]`

`const int nA = 2;`
`int eA[nA] = { 1, 2 };`
`alist* a = build(nA, eA);`
`const int nB = 3;`
`int eB[nB] = { 3, 4, 5 };`
`const alist* b = build(nB, eB);`
`append(a, b);`
`print(a);`
`destroy(a);`
`destroy(b);`
	`[ 1 2 3 4 5 ]`

Teste	Resultado

`const int nA = 4;`
`int eA[nA] = { 1, 2, 3, 4 };`
`alist* a = build(nA, eA);`
`const int nB = 3;`
`int eB[nB] = { 5, 6, 7 };`
`const alist* b = build(nB, eB);`
`append(a, b);`
`print(a);`
`destroy(a);`
`destroy(b);`
	`[ 1 2 3 4 5 6 7 ]`

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

Limpar resposta

```
1  #include "alist.cpp"
2
3  void append(alist* a, const alist* b){
4    int newsize = a->size + b->size;
5    int* vect = new int [newsize];
6    for(int i = 0; i < a->size; i++){
7      vect[i] = a->elements[i];
8    }
9    for (int i = 0; i < b->size; i++)
10   {
11     vect[i+a->size] = b->elements[i];
12   }
13   delete[] a->elements;
14   a->elements = vect;
15   a->size = newsize;
16 }
```

	Teste	Esperado	Recebido	

✔	`alist* a = empty();`
	`const alist* b = empty();`
	`append(a, b);`
	`print(a);`
	`destroy(a);`
	`destroy(b);`
		`[ ]`	`[ ]`	✔

✔	`alist* a = empty();`
	`const int nB = 2;`
	`int eB[nB] = { 1, 2 };`
	`const alist* b = build(nB, eB);`
	`append(a, b);`
	`print(a);`
	`destroy(a);`
	`destroy(b);`
		`[ 1 2 ]`	`[ 1 2 ]`	✔

✔	`const int nA = 2;`
	`int eA[nA] = { 1, 2 };`
	`alist* a = build(nA, eA);`
	`const alist* b = empty();`
	`append(a, b);`
	`print(a);`
	`destroy(a);`
	`destroy(b);`
		`[ 1 2 ]`	`[ 1 2 ]`	✔

	Teste	Esperado	Recebido	

✔	const int nA = 2;
	int eA[nA] = { 1, 2 };
	alist* a = build(nA, eA);
	const int nB = 3;
	int eB[nB] = { 3, 4, 5 };
	const alist* b = build(nB, eB);
	append(a, b);
	print(a);
	destroy(a);
	destroy(b);
		[1 2 3 4 5]	[1 2 3 4 5]	✔

✔	const int nA = 4;
	int eA[nA] = { 1, 2, 3, 4 };
	alist* a = build(nA, eA);
	const int nB = 3;
	int eB[nB] = { 5, 6, 7 };
	const alist* b = build(nB, eB);
	append(a, b);
	print(a);
	destroy(a);
	destroy(b);
		[1 2 3 4 5 6 7]	[1 2 3 4 5 6 7]	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
#include "alist.cpp"

void append(alist* a, const alist* b) {
  if (b->size == 0)
    return; // b is the empty list

  // Create new array of elements
  int new_size = a->size + b->size;
  int* new_elements = new int[new_size];

  // Copy values from a and b
  for (int i = 0; i < a->size; i++) {
    new_elements[i] = a->elements[i];
  }
  for (int i = 0; i < b->size; i++) {
    new_elements[a->size + i] = b->elements[i];
  }
  // Delete old array
  delete [] a->elements;

  // Point to new array
  a->size = new_size;
```

Correta

Nota desta submissão: 20/20

Pergunta 2

Correta Pontuou 20 de 20

Consider the code given in node.cpp containing the definition of type `node`, supporting the definition of singly-linked lists with `int` values, and associated functions:

- `node* build(int x, node* n)`: builds a new node with value `x` (the `value` member), followed by `n` (the `next` member);
- `void destroy(node* n)`: releases the memory allocated to `n` and successor nodes; and
- `void print(node* n)`: prints values in the node pointed by `n` and successor nodes.

Define a new function `node* remove(int x, node* n)` such that a call to `remove(x, n)` removes the first occurrence of value `x` in the node list pointed by `n`, and returns a pointer to an updated list (with the first occurrence of `x` removed). If value `x` can not be found, the function should return `n` unchanged.

To avoid memory leaks, the implementation should release memory appropriately using the `delete` operator.

You need to include `node.cpp` file in your code i.e. `#include "node.cpp"`.

Por exemplo:

Teste	Resultado

`node* n = nullptr;`
`n = remove(0, n);`
`print(n);`
`destroy(n);`
	`[]`

`node* n = build(1, nullptr);`
`n = remove(1, n);`
`print(n);`
`destroy(n);`
	`[]`

`node* n = build(1, build(2, nullptr));`
`n = remove(0, n);`
`print(n);`
`destroy(n);`
	`[1,2]`

`node* n = build(1, build(2, build(3, nullptr)));`
`n = remove(2, n);`
`print(n);`
`destroy(n);`
	`[1,3]`

`node* n = build(1, build(2, build(3, build(4, nullptr))));`
`n = remove(4, n);`
`print(n);`
`destroy(n);`
	`[1,2,3]`

`node* n = build(1, build(2, build(3, build(4, nullptr))));`
`n = remove(1, n);`
`print(n);`
`destroy(n);`
	`[2,3,4]`

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

Limpar resposta

```
#include "node.cpp"

node* remove(int x, node* n){
  if(n == nullptr){
    return nullptr;
  }

  node* prev = n;
  node* cur = n->next;

  if(prev->value == x){
    node* res = n->next;
    delete(n);
    return res;
  }

  while(cur != nullptr){
    if(cur->value == x){
      prev->next = cur->next;
      delete(cur);
      return n;
```

```
22 |   }
```

	Teste	Esperado	Recebido	

✔	node* n = nullptr;
	n = remove(0, n);
	print(n);
	destroy(n);
		[]	[]	✔

✔	node* n = build(1, nullptr);
	n = remove(1, n);
	print(n);
	destroy(n);
		[]	[]	✔

✔	node* n = build(1, build(2, nullptr));
	n = remove(0, n);
	print(n);
	destroy(n);
		[1,2]	[1,2]	✔

✔	node* n = build(1, build(2, build(3, nullptr)));
	n = remove(2, n);
	print(n);
	destroy(n);
		[1,3]	[1,3]	✔

✔	node* n = build(1, build(2, build(3, build(4, nullptr))));
	n = remove(4, n);
	print(n);
	destroy(n);
		[1,2,3]	[1,2,3]	✔

✔	node* n = build(1, build(2, build(3, build(4, nullptr))));
	n = remove(1, n);
	print(n);
	destroy(n);
		[2,3,4]	[2,3,4]	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```c
#include "node.cpp"

//! Removes the first occurrence of value in the node list.
node* remove(int value, node* n) {
  node* curr = n;
  node* prev = nullptr;
  while (curr != nullptr && curr -> value != value) {
    prev = curr;
    curr = curr -> next;
  }
  if (curr != nullptr) {
    // value found
    if (prev == nullptr) {
      // value in first node
      n = curr -> next;
    } else {
      prev -> next = curr -> next;
    }
    delete curr;
  }
  return n;
}
```

Correta

Nota desta submissão: 20/20

Pergunta 3

Correta Pontuou 20 de 20

Consider the code given in dlnode.cpp,containing the definition of type `dlnode`, supporting the definition of doubly-linked lists with `int` values, and associated functions:

- `node* build(int v, dlnode* n)`: builds a new node with value `v` (the `value` member), followed by `n` (the `next` member) — if `n != nullptr` then `n->prev` is set to point to the new node;
- `void destroy(dlnode* n)`: releases the memory allocated to `n` and successor nodes; and
- `void print(const dlnode* n)`: prints the contents in the node chain pointed by `n`.

Define a new function `dlnode* insert(dlnode* dln, int k, int v)` that returns the node chain that results from `dln` by inserting a node with value `v` before the `k`-th node in the chain, where `k >= 0` and `k <= n` where `n` is the number of nodes in the `dln` chain. For `k = 0`, the new node must be a predecessor of the first node in the `dln` chain, and for `k = n` it must be a successor of the last node. The function should return a pointer to the first element in the resulting node chain.

You need to include `dlnode.cpp` file in your code i.e. `#include "dlnode.cpp"`.

You cannot use any C++ library classes or functions, including `vector`, `list`, or `string`.

Por exemplo:

Teste	Resultado

`dlnode* dln = nullptr;`
`dln = insert(dln, 0, 0);`
`print(dln); destroy(dln);`
	`(\<0<\)`

`dlnode* dln = build(1, build(2));`
`dln = insert(dln, 0, 0);`
`print(dln); destroy(dln);`
	`(\<0<1)(0<1<2)(1<2<\)`

`dlnode* dln = build(1, build(2));`
`dln = insert(dln, 1, 0);`
`print(dln); destroy(dln);`
	`(\<1<0)(1<0<2)(0<2<\)`

`dlnode* dln = build(1, build(2));`
`dln = insert(dln, 2, 0);`
`print(dln); destroy(dln);`
	`(\<1<2)(1<2<0)(2<0<\)`

`dlnode* dln = build(1, build(2, build(3, build(4))));`
`dln = insert(dln, 3, 0);`
`print(dln); destroy(dln);`
	`(\<1<2)(1<2<3)(2<3<0)(3<0<4)(0<4<\)`

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

Limpar resposta

```
#include "dlnode.cpp"

dlnode* insert(dlnode* dln, int k, int v){
  if(dln ==  nullptr){
    dlnode* nova = build(v, nullptr);
    nova->prev = nullptr;
    return nova;
  }
  if(k==0){
    dlnode* nova = build(v, dln);
    dln->prev = nova;
    return nova;
  }
  dlnode* atual = dln;
  dlnode* prox = dln->next;
  for(int i=1; i<=k; i++){
    if(i==k){
      dlnode* nova = build(v, prox);
      nova->prev = atual;
      atual->next = nova;
      if(prox!=nullptr){
        prox->prev = nova;
```

	Teste	Esperado	Recebido	

	Teste	Esperado	Recebido	

✔	`dlnode* dln = nullptr;`
	`dln = insert(dln, 0, 0);`
	`print(dln); destroy(dln);`
		`(\<0<\)`	`(\<0<\)`	✔

✔	`dlnode* dln = build(1, build(2));`
	`dln = insert(dln, 0, 0);`
	`print(dln); destroy(dln);`
		`(\<0<1)(0<1<2)(1<2<\)`	`(\<0<1)(0<1<2)(1<2<\)`	✔

✔	`dlnode* dln = build(1, build(2));`
	`dln = insert(dln, 1, 0);`
	`print(dln); destroy(dln);`
		`(\<1<0)(1<0<2)(0<2<\)`	`(\<1<0)(1<0<2)(0<2<\)`	✔

✔	`dlnode* dln = build(1, build(2));`
	`dln = insert(dln, 2, 0);`
	`print(dln); destroy(dln);`
		`(\<1<2)(1<2<0)(2<0<\)`	`(\<1<2)(1<2<0)(2<0<\)`	✔

✔	`dlnode* dln = build(1, build(2, build(3, build(4))));`
	`dln = insert(dln, 3, 0);`
	`print(dln); destroy(dln);`
		`(\<1<2)(1<2<3)(2<3<0)(3<0<4)(0<4<\)`	`(\<1<2)(1<2<3)(2<3<0)(3<0<4)(0<4<\)`	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```c
#include "dlnode.cpp"

//! Inserts a node with value before the pos-th node in the chain
dlnode* insert(dlnode* dln, int pos, int value) {
  if (pos == 0) {
    dlnode* new_node = new dlnode { value, nullptr, dln };
    if (dln != nullptr) { dln->prev = new_node; }
    return new_node;
  }
  dlnode* curr = dln;
  dlnode* prev;
  do {
    prev = curr;
    curr = curr->next;
    pos--;
  } while (pos > 0);
  prev->next = new dlnode { value, prev, curr };
  if (curr != nullptr) { curr->prev = prev->next; }
  return dln;
}

/*
```

Correta

Nota desta submissão: 20/20

Pergunta 4

Correta Pontuou 20 de 20

Consider the code given in etree.cpp, containing the definition of type `etree`, supporting the definition of expression trees corresponding to the use of integer constants combined with arithmetic operators for addition, subtraction, multiplication, and division.

```
struct etree {
  int value;     // Number or operator
  etree* left;   // Left operator
  etree* right;  // Right operator
};
```

An expression tree `et` represents:

- An integer `n` if `et->value` holds value `n` and `et->left` and `et->right` are both null pointers (`nullptr`).
- The operation `L op R` where `op` is an operation code (one of `ADD`, `SUB`, `MUL`, `DIV`) if `et->value` holds value `op` and `L` and `R` are represented by `et->left` and `et->right` respectively.

In association, the following functions are provided:

- `etree* build(int v, etree* left, etree* right)`: builds a new expression tree initialised using value `v` an subtrees `left` and `right`;
- `destroy(etree* et)`: releases the memory allocated to `et` and its sub-trees;
- `number(int n)`: shorthand function to create a new expression tree representing an integer value `n` (used to express test cases); and
- `add(etree* left, etree* right)` and similarly defined `sub`, `mul`, and `div`: shorthand functions to create expression tree corresponding respectively to addition, subtraction, multiplication, and division (also used to express test cases);

Write a C++ function `int eval(const etree* t)` that computes the result of evaluating an expression tree. For instance, if `et` is the expression tree defined by `add(number(123), number(-122))` then `eval(et)` should return `1`.

You need to include `etree.cpp` file in your code i.e. `#include "etree.cpp"`.

You cannot use any C++ library classes or functions, including `vector`, `list`, or `string`.

Hint: define `eval` recursively.

Por exemplo:

Teste	Resultado

`etree* et = number(123);`
`int v = eval(et);`
`cout << v << '\n';`
`destroy(et);`
	123

`etree* et = add(number(123), number(-122));`
`int v = eval(et);`
`cout << v << '\n';`
`destroy(et);`
	1

`etree* et = sub(mul(number(30), number(2)), div(number(10), number(5)));`
`int v = eval(et);`
`cout << v << '\n';`
`destroy(et);`
	58

`etree* et = mul(mul(mul(mul(number(2), number(2)), number(2)), number(2)), number(2));`
`int v = eval(et);`
`cout << v << '\n';`
`destroy(et);`
	32

`etree* et = add(number(2), sub(number(2), mul(number(2), div(number(2), number(1)))));`
`int v = eval(et);`
`cout << v << '\n';`
`destroy(et);`
	0

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

Limpar resposta

```
1  #include "etree.cpp"
2
3  int eval(const etree* t){
4    if(t->right==nullptr && t->left==nullptr) return t->value;
5    if(t->value == '+') return eval(t->left) + eval(t->right);
```

```
  if(t->value == '-') return eval(t->left) - eval(t->right);
  if(t->value == '*') return eval(t->left) * eval(t->right);
  if(t->value == '/') return eval(t->left) / eval(t->right);
  return 0;
}
```

	Teste	Esperado	Recebido	

✔	`etree* et = number(123);`
	`int v = eval(et);`
	`cout << v << '\n';`
	`destroy(et);`
		123	123	✔

✔	`etree* et = add(number(123), number(-122));`
	`int v = eval(et);`
	`cout << v << '\n';`
	`destroy(et);`
		1	1	✔

✔	`etree* et = sub(mul(number(30), number(2)), div(number(10), number(5)));`
	`int v = eval(et);`
	`cout << v << '\n';`
	`destroy(et);`
		58	58	✔

✔	`etree* et = mul(mul(mul(mul(number(2), number(2)), number(2)), number(2)), number(2));`
	`int v = eval(et);`
	`cout << v << '\n';`
	`destroy(et);`
		32	32	✔

✔	`etree* et = add(number(2), sub(number(2), mul(number(2), div(number(2), number(1)))));`
	`int v = eval(et);`
	`cout << v << '\n';`
	`destroy(et);`
		0	0	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
#include "etree.cpp"

//! Computes the result of evaluating the expression tree.
int eval(const etree* et) {
  if (et->left == nullptr && et->right == nullptr) {
    return et->value;
  }
  int rl = eval(et->left);
  int rr = eval(et->right);
  int r = 0;
  switch (et->value) {
    case ADD: r = rl + rr; break;
    case SUB: r = rl - rr; break;
    case MUL: r = rl * rr; break;
    case DIV: r = rl / rr; break;
  }
  return r;
}

/*
  // private tests (1000 points each)
  {
```

Correta

Nota desta submissão: 20/20

Pergunta 5

Correta Pontuou 20 de 20

A sparse vector (also called sparse array) is an array where most elements have value 0. For space efficiency, data structures for sparse vectors only encode non-zero values. Consider the code given in `svnode.cpp` containing the definition of type `svnode` for representing sparse vectors of integer values as singly-linked lists, with the following member fields:

- `position`: the index of the represented value;
- `value`: the value itself;
- `next`: successor node — if defined (non-null) then it refers to a higher vector position, that is, if `next != nullptr` then `position < next->position`

`nullpr` is used to stand for a sparse vector containing only zeros. The following functions are defined in association:

- `svnode* build(int pos, int v, svnode* svn)`: build a new sparse vector node considering the insertion of value `v` at position `pos` in the previously existing node chain `svn`;
- `void destroy(svnode* svn)`: releases the memory allocated to `svn` and successor nodes; and
- `void print(const svnode* svn)`: prints the contents of `svn` and successor nodes (position-value pairs).

Write a C++ function `svnode* sum(const svnode* a, const svnode* b)` that creates a new sparse vector representing the sum of `a` and `b`.

You need to include `svnode.cpp` file in your code i.e. `#include "svnode.cpp"`.

You cannot use any C++ library classes or functions, including `vector`, `list`, or `string`.

Por exemplo:

Teste	Resultado

`svnode* a = nullptr;`
`svnode* b = nullptr;`
`svnode* c = sum(a, b);`
`print(a); print(b); print(c); cout << '\n';`
`destroy(a); destroy(b); destroy(c);`
	`[ ][ ][ ]`

`svnode* a = build(0, 1, nullptr);`
`svnode* b = build(0, -1, nullptr);`
`svnode* c = sum(a, b);`
`print(a); print(b); print(c); cout << '\n';`
`destroy(a); destroy(b); destroy(c);`
	`[ 0>1 ][ 0>-1 ][ ]`

`svnode* a = build(0, 1, build(10, 2, nullptr));`
`svnode* b = nullptr;`
`svnode* c = sum(a, b);`
`print(a); print(b); print(c); cout << '\n';`
`destroy(a); destroy(b); destroy(c);`
	`[ 0>1 10>2 ][ ][ 0>1 10>2 ]`

`svnode* a = nullptr;`
`svnode* b = build(0, 1, build(10, 2, nullptr));`
`svnode* c = sum(a, b);`
`print(a); print(b); print(c); cout << '\n';`
`destroy(a); destroy(b); destroy(c);`
	`[ ][ 0>1 10>2 ][ 0>1 10>2 ]`

`svnode* a = build(0, 1, build(10, 2, nullptr));`
`svnode* b = build(0, -1, build(10, 3, build(100, 4, nullptr)));`
`svnode* c = sum(a, b);`
`print(a); print(b); print(c); cout << '\n';`
`destroy(a); destroy(b); destroy(c);`
	`[ 0>1 10>2 ][ 0>-1 10>3 100>4 ][ 10>5 100>4 ]`

`svnode* a = build(0, 1, build(10, 2, build(100, 3, build(101, 4, nullptr))));`
`svnode* b = build(0, -1, build(99, 4, build(100, -3, nullptr)));`
`svnode* c = sum(a, b);`
`print(a); print(b); print(c); cout << '\n';`
`destroy(a); destroy(b); destroy(c);`
	`[ 0>1 10>2 100>3 101>4 ][ 0>-1 99>4 100>-3 ][ 10>2 99>4 101>4 ]`

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

Limpar resposta

```
1 #include "svnode.cpp"
2
3 svnode* sum(const svnode* a, const svnode* b){
```

```
svnode* sum(const svnode* a, const svnode* b){
  if(a == nullptr && b == nullptr) return nullptr;
  svnode* c = build(0, 0, nullptr);
  svnode* first = c;
  while(a != nullptr && b != nullptr){
    if(a->position == b->position){
      int val = a->value + b->value;
      if(val != 0){
        c->next = build(a->position, val, nullptr);
        c = c->next;
      }
      a = a->next;
      b = b->next;
    }
    else if(a->position < b->position){
      c->next = build(a->position, a->value, nullptr);
      c = c->next;
      a = a->next;
    }
    else{
```

	Teste	Esperado	Recebido	

✔	svnode* a = nullptr;
	svnode* b = nullptr;
	svnode* c = sum(a, b);
	print(a); print(b); print(c); cout << '\n';
	destroy(a); destroy(b); destroy(c);
		[ ][ ][ ]	[ ][ ][ ]	✔

✔	svnode* a = build(0, 1, nullptr);
	svnode* b = build(0, -1, nullptr);
	svnode* c = sum(a, b);
	print(a); print(b); print(c); cout << '\n';
	destroy(a); destroy(b); destroy(c);
		[0>1][0>-1][ ]	[0>1][0>-1][ ]	✔

✔	svnode* a = build(0, 1, build(10, 2, nullptr));
	svnode* b = nullptr;
	svnode* c = sum(a, b);
	print(a); print(b); print(c); cout << '\n';
	destroy(a); destroy(b); destroy(c);
		[0>1 10>2][ ][0>1 10>2]	[0>1 10>2][ ][0>1 10>2]	✔

✔	svnode* a = nullptr;
	svnode* b = build(0, 1, build(10, 2, nullptr));
	svnode* c = sum(a, b);
	print(a); print(b); print(c); cout << '\n';
	destroy(a); destroy(b); destroy(c);
		[ ][0>1 10>2][0>1 10>2]	[ ][0>1 10>2][0>1 10>2]	✔

✔	svnode* a = build(0, 1, build(10, 2, nullptr));
	svnode* b = build(0, -1, build(10, 3, build(100, 4, nullptr)));
	svnode* c = sum(a, b);
	print(a); print(b); print(c); cout << '\n';
	destroy(a); destroy(b); destroy(c);
		[0>1 10>2][0>-1 10>3 100>4][10>5 100>4]	[0>1 10>2][0>-1 10>3 100>4][10>5 100>4]	✔

✔	svnode* a = build(0, 1, build(10, 2, build(100, 3, build(101, 4, nullptr))));
	svnode* b = build(0, -1, build(99, 4, build(100, -3, nullptr)));
	svnode* c = sum(a, b);
	print(a); print(b); print(c); cout << '\n';
	destroy(a); destroy(b); destroy(c);
		[0>1 10>2 100>3 101>4][0>-1 99>4 100>-3][10>2 99>4 101>4]	[0>1 10>2 100>3 101>4][0>-1 99>4 100>-3][10>2 99>4 101>4]	✔

Passou em todos os testes! ✔

Solução do autor da pergunta (C):

```
#include "svnode.cpp"

//! Creates a new sparse v-ector representing the sum of a and b.
svnode* sum(const svnode* a, const svnode* b) {
  if (a == nullptr && b == nullptr)
    return nullptr;
  svnode* result = nullptr;
  svnode* curr = nullptr;
  while (a != nullptr && b != nullptr) {
    // node created in this iteration if any
    svnode* n = nullptr;

    // handle 3 possible cases
    if (a->position < b->position) {
      n = new svnode { a->position, a->value, nullptr };
      a = a->next;
    }
    else if (a->position > b->position) {
      n = new svnode { b->position, b->value, nullptr };
      b = b->next;
    } else {
      // same position for a and b
```

Correta

Nota desta submissão: 20/20

◄ T03 22/03

Ir para...

T04 29/03 ►