Departamento de Matemática da Universidade de Coimbra	
Métodos de Programação II	**Projeto 1**

Relatório do Projeto 1

TAD Inteiros de

Grande Dimensão

Trabalho Realizado por: Pedro Silva

Especificação: Em C o tipo *int* é implementado como 8 *bytes*, o que acaba por nos limitar se quisermos trabalhar com inteiros de grande dimensão. Posto isto foi proposto que implementássemos um módulo, Inteiros de Grande Dimensão (IDA), que que permita a criação e manipulação de inteiros de dimensão arbitrária, inteiros cujo limite não seja fixo mas arbitrário, só limitado pela dimensões da memória do sistema computacional. Assim sendo, para poder realizar operações entre dois inteiros, decidi defini-los como vetores que em cada célula têm um algarismo do número que pretendo operar.

Entradas: Dimensão de cada vetor e os seus elementos.

Saídas: Resultado das operações escolhidas.

Utilização: O programa começa por pedir a dimensão e os elementos dos dois vetores que pretendemos operar. De seguida entramos no menu do programa e selecionamos a operação que pretendemos. Depois de fazermos uma operação o programa retorna ao menu.

Algoritmo:

```
Ida somarVetoresComCarry(Ida v1, Ida v2) {
   Ida resultado = criarVetor(v1.tamanho); // Cria o vetor resultado com o mesmo tamanho de v1 e v2

   ajustarTamanhoVetores0(&v1, &v2); // Ajusta o tamanho dos vetores, garantindo que tenham o
mesmo tamanho

   int tamanho = v1.tamanho;
   int carry = 0;

   for (int i = tamanho - 1; i >= 0; i--) {
      int soma = v1.vetor[i] + v2.vetor[i] + carry;
      resultado.vetor[i] = soma % 10; // Calcula o algarismo da soma e armazena no resultado
      carry = soma / 10; // Calcula o carry para a próxima posição
   }

   if (carry > 0) {
      for (int i = tamanho - 1; i >= 0; i--) {
         resultado.vetor[i + 1] = resultado.vetor[i]; // Desloca os elementos do resultado para a direita
      }
      resultado.vetor[0] = carry; // Define o carry como o algarismo mais significativo
      resultado.tamanho = tamanho + 1; // Atualiza o tamanho do resultado
   } else {
      resultado.tamanho = tamanho; // Tamanho do resultado permanece inalterado
   }

   return resultado; // Retorna o vetor resultado com a soma dos vetores v1 e v2
}
```

A função começa por ajustar o tamanho dos dois vetores de entrada, se necessário, para que tenham o mesmo tamanho. Em seguida, inicia uma iteração dos vetores a partir do dígito menos significativo, à direita, para o dígito mais significativo, à esquerda. Realiza a soma dos dígitos correspondentes dos dois vetores e adiciona qualquer valor de carry, resto da soma anterior. Armazena o dígito resultante no vetor de saída e atualiza o carry para o próximo dígito. Se houver um carry não utilizado no final, um novo dígito com o valor do carry é adicionado ao início do vetor de saída.

```
Ida subtrairVetores(Ida v1, Ida v2) {
   Ida resultado = criarVetor(v1.tamanho); // Cria o vetor resultado com o mesmo tamanho de v1 e v2

   ajustarTamanhoVetores0(&v1, &v2); // Ajusta o tamanho dos vetores, garantindo que tenham o
mesmo tamanho

   int comparacao = compararVetores(v1, v2);

   if (comparacao == -1) {
      Ida temp = v1;
      v1 = v2;
      v2 = temp;
   }

   resultado.tamanho = v1.tamanho;

   int borrow = 0;

   for (int i = v1.tamanho - 1; i >= 0; i--) {
      int subtracao;
      if (i >= v2.tamanho) {
         subtracao = v1.vetor[i] - borrow; // Subtrai o algarismo de v1 e o borrow (carry)
      } else {
         subtracao = v1.vetor[i] - v2.vetor[i] - borrow; // Subtrai o algarismo de v1, o algarismo de v2 e o
borrow (carry)
      }

      if (subtracao < 0) {
         subtracao += 10; // Se a subtração for negativa, ajusta o valor adicionando 10
         borrow = 1; // Define o carry como 1
      } else {
         borrow = 0; // Não há carry
      }

      resultado.vetor[i] = subtracao; // Armazena o resultado da subtração no vetor resultado
   }

   while (resultado.tamanho > 1 && resultado.vetor[resultado.tamanho - 1] == 0) {
      resultado.tamanho--; // Remove zeros não significativos do resultado
   }

   return resultado; // Retorna o vetor resultado com a subtração dos vetores v1 e v2
}
```

Esta função também começa por ajustar o tamanho dos vetores de entrada, se necessário, para que tenham o mesmo tamanho. Em seguida, determina qual dos vetores deve ser subtraído do outro, pois não encontrei forma de lidar com números negativos. Começando pelos dígitos menos significativos, realiza a subtração dos dígitos correspondentes e considera qualquer valor de "empréstimo" do dígito anterior.
Se o resultado for negativo, adiciona 10 ao dígito e guarda empréstimo para o próximo dígito. O resultado é armazenado no vetor de saída. Se houver dígitos zero à esquerda do resultado, eles são removidos para criar um vetor de tamanho mínimo.

```
Ida multiplicarVetores(Ida v1, Ida v2) {
   int tamanho1 = v1.tamanho;
   int tamanho2 = v2.tamanho;
   int tamanhoResultado = tamanho1 + tamanho2;

   Ida resultado;

   // Inicializa o vetor resultado com zeros
   for (int i = 0; i < tamanhoResultado; i++) {
      resultado.vetor[i] = 0;
   }

   for (int i = tamanho1 - 1; i >= 0; i--) {
      int carry = 0;
      int resultadoIndex = i + tamanho2;  // Ajusta o índice de armazenamento do resultado

      for (int j = tamanho2 - 1; j >= 0; j--) {
         int produto = v1.vetor[i] * v2.vetor[j] + resultado.vetor[resultadoIndex] + carry;
         resultado.vetor[resultadoIndex] = produto % 10;
         carry = produto / 10;
         resultadoIndex--;  // Move para a próxima posição à esquerda
      }

      resultado.vetor[i] = carry;
   }

   // Encontre a posição do primeiro dígito não zero à esquerda
   int primeiroNaoZero = 0;
   while (primeiroNaoZero < tamanhoResultado && resultado.vetor[primeiroNaoZero] == 0) {
      primeiroNaoZero++;
   }

   // Se há zeros à esquerda, remova-os
   int tamanhoFinal = tamanhoResultado - primeiroNaoZero;
   for (int i = 0; i < tamanhoFinal; i++) {
      resultado.vetor[i] = resultado.vetor[i + primeiroNaoZero];
   }

   resultado.tamanho = tamanhoFinal;

   return resultado; // Retorna o vetor resultado com o resultado da multiplicação
}
```

A função inicia inicializando um vetor resultado com zeros e determina o tamanho necessário para o resultado. Em seguida inicia duas iterações aninhadas sobre os dois vetores de entrada, multiplicando dígito a dígito. O produto parcial é calculado e adicionado ao dígito correspondente do resultado, com consideração a qualquer carry anterior. O carry é calculado dividindo o produto parcial por 10. O dígito resultante é armazenado no vetor de saída e o carry é levado para o próximo dígito. O resultado é

posteriormente ajustado para remover quaisquer zeros não significativos à esquerda.

```
Ida dividirInteiro(Ida dividendo, Ida divisor, Ida *quociente, Ida *resto) {
   Ida resultado = criarVetor(dividendo.tamanho); // Inicializa o vetor resultado com o tamanho do
dividendo

   // Inicializa o quociente como zero
   *quociente = criarVetor(1);

   // Verifica se o divisor é zero (divisão por zero)
   if (compararVetores(divisor, resultado) == 0) {
      // Divisão por zero, retorna um quociente e resto de zero
      *resto = resultado;
      return resultado;
   }

   // Enquanto o dividendo for maior ou igual ao divisor
   while (compararVetores(dividendo, divisor) != -1) {
      Ida temp = divisor;
      quociente->vetor[0]++;  // Incrementa o quociente

      // Subtrai o divisor do dividendo
      temp = subtrairVetores(dividendo, temp);
      dividendo = temp;

      // Adiciona o divisor ao resultado
      resultado = somarVetoresComCarry(resultado, divisor);
   }

   *resto = dividendo;  // O resto é o valor final do dividendo
   return resultado; // Retorna o quociente
}
```

A função inicia com o quociente e o resultado definidos como zero. Verifica se o divisor é zero, divisão por zero, e, se for, retorna um quociente e um resto de zero. Em seguida, enquanto o dividendo for maior ou igual ao divisor, realiza uma iteração. Subtrai o divisor do dividendo e incrementa o quociente em um. Adiciona o divisor ao resultado. O resto final é o valor remanescente no dividendo. O resultado é retornado como o quociente.

Estruturas de Dados: Inteiros, vetores e Ida.

Particularidades: Não consegui arranjar forma para lidar com números negativos, portanto o programa apenas considera números positivos.

Exemplos:

```
Quantos elementos deseja preencher no vetor1 (max 300)? 2
Por favor, introduza os algarismos no vetor:
1
0
Quantos elementos deseja preencher no vetor2 (max 300)? 3
Por favor, introduza os algarismos no vetor:
1
3
4
Selecione a operacao que deseja realizar:
1. Subtrair Vetores
2. Somar Vetores
3. Multiplicar Vetores
4. Divisao Inteira
5. Comparar Vetores
0. Sair
Opcao: 1
O resultado da subtracao e:
1 2 4
```

```
Quantos elementos deseja preencher no vetor1 (max 300)? 8
Por favor, introduza os algarismos no vetor:
5
4
2
1
6
8
5
4
Quantos elementos deseja preencher no vetor2 (max 300)? 7
Por favor, introduza os algarismos no vetor:
5
4
1
2
3
6
5
Selecione a operacao que deseja realizar:
1. Subtrair Vetores
2. Somar Vetores
3. Multiplicar Vetores
4. Divisao Inteira
5. Comparar Vetores
0. Sair
Opcao: 3
O resultado da multiplicacao e:
2 9 3 4 4 1 4 0 2 9 9 9 7 1 0
```

```
Quantos elementos deseja preencher no vetor1 (max 300)? 6
Por favor, introduza os algarismos no vetor:
2
5
1
3
2
5
Quantos elementos deseja preencher no vetor2 (max 300)? 3
Por favor, introduza os algarismos no vetor:
2
1
2
Selecione a operacao que deseja realizar:
1. Subtrair Vetores
2. Somar Vetores
3. Multiplicar Vetores
4. Divisao Inteira
5. Comparar Vetores
0. Sair
Opcao: 5
O vetor 1 e maior que o vetor 2.
```