

Cryptopals - Relatório do desafio 1

Link: <https://cryptopals.com/sets/1/challenges/1>

1 - Introdução

O exercício 1 é um desafio que pede para construir um programa em alguma linguagem de programação que converta uma string em hexadecimal para uma string na base 64. Ambos tipos de bases são tipos de codificação de dados.

Uma string na base hexadecimal é uma palavra que pode ser pequena ou grande, em que cada caractere varia entre 0 e 15, porém os valores de 10 a 15 são representados da letra "A" até "F" (maiúsculos ou minúsculas). Sendo assim, cada caractere de uma string em hexadecimal pode ser representada por 16 caracteres, sendo eles "0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F", como exemplo temos a string "4f206469612065737461206c696e646f". Traduzindo para o texto que estamos acostumados a ler e observar, fica "O dia esta lindo".

Por outro lado temos a string na base64, que possui esse nome, pois existem 64 símbolos distintos para representar cada dígito em sua estrutura. Esses caracteres vão de "A" até "Z", "a" até "z", 0 até 9 e os símbolos + e /. Com isso temos todos eles na seguinte string: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/". Podemos usar como exemplo a mesma frase citada anteriormente que em base 64, fica "TyBkaWEgZXN0YSBsaW5kbw==".

Desse modo, temos que construir um algoritmo que transforme uma string da base hexadecimal para a base 64. Por exemplo, vimos que a string "O dia esta lindo" em hexa é igual a "4f206469612065737461206c696e646f", convertendo esta da base16 para a base64, temos "TyBkaWEgZXN0YSBsaW5kbw==".

Na descrição deste exercício no site temos como exemplo a string:

`"49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6d"`

Sendo passada para o programa e gerando o seguinte resultado:

`<SsdtlGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t>`

2 - Como funciona a conversão de hexadecimal para a base 64?

Inicialmente, precisamos entender a conversão de uma palavra para a base hexadecimal e desta para a base64. Em uma primeira análise, é importante entender a transformação de uma palavra comum para a base binária. No computador, cada caractere de uma palavra é representado por 1 byte, que são o conjunto de 8 bits. Bits são parecidos com os átomos, ou seja são a menor unidade lógica de tamanho que pode ser representada

em um computador. Estes valores podem ser representados por 2 valores diferentes: 0 ou 1. Com 1 byte podemos representar no máximo 256 caracteres, resultado da combinação de 8 bits ($2^8 = 256$). A representação de dados em um computador se baseia em 0's e 1's, esses conjuntos de bits são chamados de valores na base binária. Com isso, ao converter a string para a base binária vamos ter diversos conjuntos de 8 bits, juntando todos esses grupos, temos a palavra na base2(binária). É basicamente dessa forma, que convertemos uma palavra comum para a base binária.

A partir da análise anterior observa-se que valores na base16 podem ser representados por até 16 caracteres diferentes. Logo, temos 4 bits para representar cada valor, pois $2^4 = 16$. Dessa forma, podemos pegar a string binária de 4 em 4 elementos da esquerda para a direita, converter para algum caractere na base16 e assim sucessivamente. Quando chegar ao fim da string, temos a conversão dela para a base hexadecimal. Entretanto, precisamos manter ela na base binária para converter para a base64. Para converter da base16 para a base2, temos que converter cada dígito da base hexadecimal para a base2 com 4 bits e juntar tudo, sendo assim teremos a string da base16 na base2.

Com isso, para converter a palavra na base16 para base64, é preciso converter para a base2 e por fim passar para a base64. Como $2^6 = 64$, então temos 6 bits para representar cada caractere na base64, logo precisamos pegar de 6 em 6 bits, da esquerda para a direita, na string binária, converter para a base decimal e verificar qual caractere esse valor representa na tabela de base64. Desse modo, é só continuar o processo de conversão até chegar no final da string binária, quando terminar basta juntar todos os dígitos obtidos nesta base e teremos a string da base16 convertida para a base64.

Entretanto, quando a string na base2 não é divisível por 6 sobra uma certa quantidade de bits no final menor que 6. Assim temos que acrescentar 0's ao final da string binária até que ela seja divisível por 6. Por convenção, a cada dois 0's acrescentados, temos que acrescentar mais seis 0's para formar 8 bits (1 byte). Estes seis zeros podem ser representados por '=' que é o "padding", na base64, utilizado para terminar a conversão da string binária para a base64 quando a string binária não é divisível por 6.

3 - Desenvolvimento

No desenvolvimento do programa, optei por converter a string em hexadecimal para a base binária e por fim para base64, como descrito anteriormente. Para isto, separei ele em 9 partes:

1ª: Criação de variáveis estáticas para serem utilizadas no programa;

2ª: Pedir string em hexadecimal para usuário pelo teclado;

3ª: Verificar se a string está em hexadecimal. Se não, mostra mensagem de erro e pede novamente. Esse processo é feito até que a string informada esteja na base correta;

4ª: Remover espaços da string informada;

5ª: Converter string para a base binária;

6ª: Acrescenta 0's na string binária até que ela seja divisível por 6;

7ª: Converter string binária para a base64, na seguinte ordem:

- Converter cada 6 bits em base decimal, da esquerda para a direita;
- Verificar caractere correspondente na tabela de base64;
- Juntar este caractere com a variável que armazena a string em base64;
- Repetir os passos anteriores até chegar no final da string binária.

8ª: Colocar “padding” ao final da string em base64, caso seja necessário;

9ª: Exibir resultado da conversão na tela.

4 - Nomes e breve explicação das variáveis utilizadas

- **void askForStringHexadecimal(char *string, int size):**
 - ➔ *int lastCharacterIndex*: Variável para guardar índice do último caractere do parâmetro **string**.
- **int main():**
 - ➔ **int index**: Significa índice e tem por função na maioria das vezes acessar o índice de outras variáveis, tal como auxiliar na iteração de uma string até o final;
 - ➔ **int count**: iniciada com valor 0 e utilizada para armazenar a quantidade de zeros acrescentados ao final da string binária. Com isso, auxiliou também na parte de adicionar “padding” ao resultado final;
 - ➔ **int var**: utilizada no caso de index ou count estarem “ocupadas”;
 - ➔ **char stringHexadecimal[1024]**: Vetor com capacidade para armazenar 1024 caracteres. Serve para armazenar a string hexadecimal de entrada na aplicação;
 - ➔ **char stringBinary[1024]**: Vetor com capacidade para armazenar 1024 caracteres. Serve para armazenar a string hexadecimal convertida para a base binária;
 - ➔ **char stringInBase64[1024]**: Vetor com capacidade para armazenar 1024 caracteres. Serve para armazenar o resultado da conversão de stringHexadecimal para base64;
 - ➔ **const char valuesInHexadecimalToBinary[16][5]**: Vetor com todos os valores da base hexadecimal convertidos para a base2. Para acessar o valor hexadecimal em binário desejado, basta acessar o índice com o número em hexadecimal;
 - ➔ **char byteInBase64[6]**: armazena os bits de algum caractere na base hexadecimal;
 - ➔ **int numberOfCharInBase64**: serve para armazenar o resultado de um caractere na base64(byteInBase64) em decimal;
 - ➔ **char padding**: Guarda o caractere ‘=’ para ajuda no término da conversão da string hexadecimal para base64, caso a string na base2 da base16 não seja divisível por seis.

5 - Funções utilizadas no programa:

- Biblioteca *stdio.h*
 - ➔ **char *fgets(char *str, int n, FILE *stream):** Lê do fluxo(**stream**) de entrada uma string até que (**n-1**) caracteres sejam digitados ou encontre algum '\n' ou EOF(final de arquivo) sejam encontrados e armazena a string no endereço apontado por **str**. Podemos utilizar essa função para ler de forma segura do teclado(**stdin**), pois ela lê até n-1 caracteres evitando, assim, buffer overflows que possam causar erros de segurança ou *crashes* no programa. A ultima posição n é destinada para o caractere de final de string('\0' = NULL). Por fim, essa função retorna o argumento **str**, em caso de erros retorna **NULL**.
- Biblioteca *string.h*
 - ➔ **char *strcat(char *dest, const char *src):** Adiciona a string apontada por **src** ao final da string apontada por **dest**. Basicamente, copia **src** para o final de **dest**. Retorna um ponteiro para a string resultante **dest**.
 - ➔ **char *char strchr(const char *str, int c):** Encontra o índice da primeira ocorrência do caractere **c** na string apontada por **str**. Retorna um ponteiro para a primeira ocorrência do caractere **c** ou **NULL** caso o caractere não for encontrado.
 - ➔ **char *strncat(char *dest, const char *src, size_t n):** Adiciona **n** caracteres(índice 0 até n) da string apontada por **src** em **dest**. Possui o mesmo retorno da função **strcat**
 - ➔ **size_t strlen(const char *str):** calcula e retorna o tamanho da string apontada por **str**.
- Biblioteca *math.h*
 - ➔ **double pow(double x, double y):** Retorna o resultado de **x elevado a y**, isto é x^y .

6 - Funções construídas no programa

- ➔ **void askForStringHexadecimal(char *string, int size):** Recebe **char *string** (um ponteiro para o tipo string) e **int size** (representa a quantidade de caracteres máxima que **string** pode armazenar). Na linha 10, criamos a variável inteira **lastCharacterIndex** para armazenar o valor do último índice da **string**. Na próxima linha utilizamos a função **fgets(string, size, stdin)** para ler uma string do teclado(**stdin**) e armazenar em **string** com tamanho máximo de **size**. Se tudo der certo, a função retorna a string lida, caso dê erro retorna **NULL**. Com isso temos o **if** que verifica se a string de entrada é diferente de **NULL**, caso positivo armazenamos o índice da última posição na variável **lastCharacterIndex** e verificamos se esta variável é igual ao caractere '\n'. Se for verdade, significa que a **fgets** leu o <enter> também. Para corrigir esse problema temos que substituir essa posição pelo símbolo '\0', que se refere ao final da string, ou seja, a string termina no índice representado por este último caractere. Por fim temos a string lida do teclado na variável do tipo **char** apontada por **string**.

```

8  /*Função para pedir string hexadecimal*/
9  ▼ void askForStringHexadecimal(char *string, int size) {
10     int lastCharacterIndex;
11  ▼   if(fgets(string, size, stdin) != NULL) {
12       lastCharacterIndex = strlen(string) - 1;
13       if(string[lastCharacterIndex] == '\n')
14           string[lastCharacterIndex] = '\0';
15   }
16 }

```

Imagem 1 – `askForStringHexadecimal(char *string, int size)`

7 - Explicação do código completo da aplicação

Inicialmente nas linhas 1, 2 e 3, temos que importar algumas bibliotecas. A linha 1 importa a biblioteca, `stdio.h`, de entrada e saída padrão em C. Nas linhas 2 e 3, importamos, respectivamente, `string.h` (para trabalhar com strings) e `math.h` (para trabalhar com funções matemáticas).

Nas linhas 5 e 6, utilizamos a diretiva **#define** para definir duas macros que vão ser utilizadas ao longo do programa. Macros são semelhantes as variáveis de um programa, porém, são valores constantes(que não se modificam) ao longo da aplicação. A macro `BASE64TABLE` se refere a uma string com todos os caracteres da base64, em que cada índice de 0 até 64 se refere a um dos dígitos da base64. Isso também vale para `BASE16TABLE`, que representa os caracteres da base hexadecimal.

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4
5  #define BASE64TABLE "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
6  #define BASE16TABLE "0123456789abcdefABCDEF"

```

Imagem 2 – Importando bibliotecas e definindo macros

Na Imagem 3 temos o começo da função `int main()`. Inicialmente temos a criação de algumas variáveis do programa. Podemos observar as variáveis inteiras `index`, `count` e `var` e os vetores do tipo `char` `stringHexadecimal`, `stringBinary` e `stringInBase64`. Ambos com tamanho máximo de 1024, ou seja, podemos armazenar strings nessas variáveis com 1024 caracteres no máximo. Por fim, temos um vetor que armazena os valores de 0 até 16 na base binária de 4 bits, cada um deles é representado em um index da string em ordem crescente.

```

18 ▼ int main() {
19
20     /** Criação das variáveis que vão ser utilizadas ao longo do programa **/
21     int index, count = 0, var = 0;
22     char stringHexadecimal[1024];
23     char stringBinary[1024] = "";
24     char stringInBase64[1024] = "";
25
26 ▼ » const char valuesInHexadecimalToBinary[16][5] = {"0000", "0001", "0010", "0011",
27                                                         "0100", "0101", "0110", "0111",
28                                                         "1000", "1001", "1010", "1011",
29                                                         "1100", "1101", "1110", "1111"};

```

Imagem 3 – Declaração de variáveis

Na figura 4, utilizamos a função *printf()* para mostrar uma mensagem ao usuário informando que ele deve inserir uma string em hexadecimal. Para capturar essa string do teclado usamos a função *askForStringHexadecimal(stringHexadecimal, 1024)* que pode ser vista na imagem 1 e com mais detalhes na sessão 6, passamos para ela a variável *stringHexadecimal* e 1024, total de caracteres que ela pode armazenar. A variável *stringHexadecimal* é onde o programa irá armazenar a string digitada pelo usuário através do teclado.

```

31 » /** Pedindo string hexadecimal para usuário **/
32 printf("Informe uma string em base hexadecimal:\n\n");
33 » askForStringHexadecimal(stringHexadecimal, 1024);

```

Imagem 4 – Entrada de dados

A Imagem 5 é uma estrutura para verificar se a *string* que o usuário informou está correta, caso contrário o programa mostra uma mensagem de erro e pede a *string* ao usuário novamente até que este informe uma *string* válida na base hexadecimal. Para isso, na linha 38 utilizamos a estrutura *while*(enquanto) para verificar o valor de *var*, se o valor dessa variável for 0, significa que o usuário informou um valor incorreto e enquanto ela for igual a 0, o processo para validação da string informada será executado. Se o processo encontrar algum caractere incorreto, então o valor de *var* irá ser 0 e o programa vai pedir uma nova *string* na tela. Na linha 39 substituímos *var* por 1, para terminar o ciclo *while*, caso o processo não encontre nenhum valor falso.

O processo de validação começa na linha 40 com a estrutura de repetição *for*, um ciclo de repetição que serve basicamente para percorrer uma estrutura de dados dando a possibilidade do usuário, por exemplo, exibir cada caractere de uma *string* na tela em cada ciclo. Nessa estrutura definimos o valor da variável *index* para 0 e vamos analisar cada caractere da *string* até que chegue no caractere de final de linha '\0'. Ao final de cada laço *for*, aumentamos o valor da variável *index* em uma unidade com o termo "++" antes de *index*. Na linha 41, com a função *strchr(BASETABLE, stringHexadecimal[index])* da biblioteca *string.h*, estamos verificando se o caractere *stringHexadecimal[index]* de *stringHexadecimal* pertence à macro *BASE16TABLE* (*string* com todos os dígitos da base hexadecimal), se não pertencer, a função retorna NULL e a segunda condição nesta mesma linha tem por objetivo verificar se o caractere é diferente do dígito espaço(' '). O *&&* no meio dessas duas condições é uma operação que avalia duas ou mais condições, neste caso se todas as condições forem verdadeiras, significa que a *string* informada está incorreta, então o programa redefine o valor de *var* para 0 na linha 42, exibe uma mensagem de erro na linha

43 pedindo uma nova string e utiliza a função `askForStringHexadecimal(stringHexadecimal, 1024)` para pedir a *string* novamente. Por fim, utilizamos o comando `break` (serve para parar um ciclo de repetição) para sair do loop *for* e voltar ao *while*. O programa vai continuar no laço *while* até que o usuário digite uma string correta.

```
35  /* Verificando se stringHexadecimal esta na base correta
36  * Se não, pedimos a string novamente até ela estar correta
37  */
38  while(var == 0) {
39      var = 1;
40      for(index = 0; stringHexadecimal[index] != '\0'; ++index) {
41          if((strchr(BASE16TABLE, stringHexadecimal[index]) == NULL) && (stringHexadecimal[index] != ' ')) {
42              var = 0;
43              printf("\n[BASE_INCORRETA!] Informe a string corretamente ...\n\n");
44              askForStringHexadecimal(stringHexadecimal, 1024);
45              break;
46          }
47      }
48  }
```

Imagem 5 – Validação da string de entrada

Com a *string* definida corretamente em hexadecimal, passamos ao próximo passo, descrito na Imagem 6, que é remover os espaços do valor informado. Para isso o programa, por meio do *loop for*, analisa cada caractere de *stringHexadecimal* e verifica se é diferente de espaço, se for verdade o programa iguala o índice *count* em *stringHexadecimal* com o índice *index* dessa mesma variável e aumenta o valor de *count* em uma unidade com `++count`. Caso encontre algum espaço ele não faz esse processo, porém incrementa o valor de *index* em uma unidade com `++index`, desse modo o caractere atual (espaço) do *loop* será representado pelo valor de *count* como índice e no próximo *loop*, este caractere será substituído pelo próximo caractere da *string*. Quando chegar ao final do *loop for*, não haverá mais espaços em brancos e por conta disso a variável *count* representará a última posição de *stringHexadecimal*, então na linha 57 o programa iguala *stringHexadecimal[count]* à `'\0'`, que representa o caracterer de final de linha. Dessa forma, temos *stringHexadecimal* sem espaços.

```
50  /** Removendo espaços em branco da string em hexadecimal, caso tenha **/
51  for(index = 0; stringHexadecimal[index] != '\0'; ++index) {
52      if(stringHexadecimal[index] != ' ') {
53          stringHexadecimal[count] = stringHexadecimal[index];
54          ++count;
55      }
56  }
57  stringHexadecimal[count] = '\0';
```

Imagem 6 – Remoção de espaços em branco da string de entrada

O próximo passo, descrito na Imagem 7, é converter *stringHexadecimal* para binário e armazenar na variável *stringBinary*. Como descrito anteriormente sabemos que 1 caractere na *base16* representa 4 *bits* de memória, então o programa analisa cada caractere de *stringHexadecimal* e converte para o seu valor na *base16*. Para isso, ele percorre todos os caracteres da string com a estrutura de repetição *for*, declarada na linha 60, nela o programa começa com *index* igual a 0, tem *stringHexadecimal[index]* diferente de `'\0'` como condição de parada e a cada loop aumenta o valor de *index* em uma unidade com `++index`. O termo *stringHexadecimal[index]* simboliza o caractere representado pelo índice *index* de *stringHexadecimal*, quando o programa faz (*stringHexadecimal[index]* - `'0'`) ele esta utilizando

a parte inteira de *stringHexadecimal[index]*, pois a linguagem C trabalha com a representação inteira de qualquer caractere em ASCII, portanto ao diminuir um caractere de '0' estamos adquirindo o valor decimal deste dígito. Dessa forma, na linha 61, o programa verifica se o termo (*stringHexadecimal[index]* - '0') está entre 0 && 9. Caso positivo, o programa na linha 62 utiliza a função *strcat()* para concatenar o valor hexadecimal em binário correspondente a esse termo em *valuesInHexadecimalToBinary* com a variável *stringBinary* (armazena conversão de *stringHexadecimal* para *base2*). Se a condição na linha 61 não for verdadeira, o programa vai para o *else*, neste existem outras estruturas condicionais para verificar qual letra de "a" até "f" o caractere representa *stringHexadecimal[index]*, por fim ele utiliza *strcat()* para concatenar em *stringBinary* o valor correspondente dessa letra na base hexadecimal para a base binária.

Todo este processo descrito acima é realizado em cada loop *for* com cada letra de *stringHexadecimal*, até que chegue ao último ('\0') caractere de *stringHexadecimal*. Quando o programa chega nele, significa que o processo de conversão terminou e temos a *stringBinary* completa e pronta para ser utilizada em um próximo passo.

```
59  >>  /** Convertendo stringHexadecimal para binário e armazenando em stringBinary */
60  >>  for(index = 0; stringHexadecimal[index] != '\0'; ++index) {
61  >>      if((stringHexadecimal[index] - '0') >= 0 && (stringHexadecimal[index] - '0') <= 9) {
62  >>          strcat(stringBinary, valuesInHexadecimalToBinary[stringHexadecimal[index] - '0']);
63  >>      }else{
64  >>          if(stringHexadecimal[index] == 'a' || stringHexadecimal[index] == 'A')
65  >>              strcat(stringBinary, valuesInHexadecimalToBinary[10]);
66  >>          else if(stringHexadecimal[index] == 'b' || stringHexadecimal[index] == 'B')
67  >>              strcat(stringBinary, valuesInHexadecimalToBinary[11]);
68  >>          else if(stringHexadecimal[index] == 'c' || stringHexadecimal[index] == 'C')
69  >>              strcat(stringBinary, valuesInHexadecimalToBinary[12]);
70  >>          else if(stringHexadecimal[index] == 'd' || stringHexadecimal[index] == 'D')
71  >>              strcat(stringBinary, valuesInHexadecimalToBinary[13]);
72  >>          else if(stringHexadecimal[index] == 'e' || stringHexadecimal[index] == 'E')
73  >>              strcat(stringBinary, valuesInHexadecimalToBinary[14]);
74  >>          else if(stringHexadecimal[index] == 'f' || stringHexadecimal[index] == 'F')
75  >>              strcat(stringBinary, valuesInHexadecimalToBinary[15]);
76  >>      }
77  >>  }
```

Imagem 7 – Conversão de *stringHexadecimal* para binário

Na Imagem 8 temos uma estrutura para adicionar 0's em *stringBinary* até que a quantidade de caracteres desta seja divisível por 6. Dessa forma, podemos converter a *string* binária para a *base64*. Esse processo basicamente acontece enquanto o tamanho de *stringBinary* dividido por 6 deixar resto 0, nisso o programa utiliza a função *strcat(stringBinary, "0")* para concatenar o valor 0 em *string* com *stringBinary* e aumenta a variável *count* em uma unidade com *++count*. Essa última variável será importante para adicionar "padding" ao final da *string* convertida para *base64* que veremos mais a frente.

```
81  >>  /** Acrescentando zeros na string binária até que ela seja divisível por 6 */
82  >>  while(strlen(stringBinary)%6 != 0){
83  >>      strcat(stringBinary, "0");
84  >>      ++count;
85  >>  }
```

Imagem 8 – Adição de 0's ao final de *stringBinary*

A Imagem 9 retrata a parte mais importante da conversão da *string* na base hexadecimal para a *base64*. Agora é o momento que o programa converte *stringBinary* para a *base64*. De forma geral, o programa converte cada 6 bits de *stringBinary*, da esquerda para a direita, para a *base64* e armazena na variável *stringInBase64*. Ao final do processo temos a *string* inicial convertida para a *base64* em *stringInBase64*.

Inicialmente, na linha 90 definimos um *for* para percorrer cada 6 *bits* de *stringBinary*, da esquerda para a direita até chegar no caractere de final de linha ('\0'). Nele temos *index* igual a 0 e a condição de parada é o caractere *stringBinary[index]* até chegar em '\0' e a cada *loop*, o programa aumenta *index* em 6 unidades com *index += 6*. Na linha 93 definimos a variável do tipo *char* *byteInBase64* com 6 posições para armazenar cada 6 posições de *stringBinary*. Na próxima linha temos outro *for* que serve para armazenar os 6 bits a partir do valor de *index* pra frente em *byteInBase64*, neste pequeno *for* tem-se *var* igual a *index*, *var* menor que (*index + 6*) e a cada *loop*, *var* aumenta uma unidade com *++var*. Dentro dele a cada *loop*, o programa utiliza a função *strncat(byteInBase64, &stringBinary[var], 1)* para concatenar o caractere *stringBinary[var]* com *byteInBase64*. Ao final, teremos 6 *bits* armazenados em *byteInBase64*. Logo em seguida o programa converte essa variável para a base decimal. Para tal, ele cria a variável inteira *numberOfCharInBase64* com valor 0 e utiliza um novo *for*, na linha 99, em que inicia a variável *var* igual a 5, *var* tem que ser maior ou igual a zero e diminui o valor de *var* em uma unidade a cada *loop* com *--var*.

Antes de prosseguir precisamos entender como funciona o processo de conversão de um valor da base binária para a base decimal. Basicamente em um número binário cada *bit* tem um valor inteiro que representa ele, para identifica-lo precisamos contar cada *bit* de trás para frente começando do 0. No caso de *byteInBase64* temos 6 *bits*, logo o *bit* mais a direita terá valor 0 e o mais a esquerda terá valor 5. Seguindo essa lógica, por fim temos que fazer a operação, $\text{valor-bit} * 2^{\text{valor-do-bit}}$, com cada *bit* de *byteInBase64* e ao final basta somar todos os resultados e teremos o valor decimal de *byteInBase64*.

Na linha 100 é basicamente o processo citado acima que esta sendo realizado, o aplicação esta convertendo cada *bit* para seu valor decimal e somando tudo. Ao final deste pequeno *loop* teremos a variável *byteInBase64* convertida para a base decimal na variável *numberOfCharInBase64*. Agora, por fim, o programa na linha 104 utiliza a função *strncat()* para concatenar o caractere correspondente à *numberOfCharInBase64* na macro *BASE64TABLE* com *stringInBase64*.

Todo este processo é feito até chegar nos últimos 6 bits de *stringBinary*, quando termina a conversão desses últimos *bits* para a *base64* temos *stringHexadecimal* convertida para a *base64* em *stringInBase64*.

```

87  ▼  /** Convertendo cada 6 bits da stringBinary para decimal, da esquerda para a direita
88      * e concatenando seu valor em base64 com stringInBase64
89      **/
90  ▼  >> for(index = 0; stringBinary[index] != '\0'; index += 6) {
91
92      //Definindo uma substring com 6 bits da stringBinary, a cada loop, e armazenando em byteInBase64
93  >> >> char byteInBase64[6] = "";
94      >> >> for(var = index; var < (index + 6); ++var)
95          >> >> strncat(byteInBase64, &stringBinary[var], 1);
96  >> >>
97  >> >> //Converter byteInBase64 para decimal
98  >> >> int numberOfCharInBase64 = 0;
99  ▼  >> >> for(var = 5; var >= 0; --var) {
100     >> >> >> numberOfCharInBase64 += (pow(2, var) * (byteInBase64[5 - var] - '0'));
101     >> >> }
102
103     >> >> //Concatenar numero correspondente ao numberOfCharInBase64 na tabela de base 64 com stringInBase64
104     >> >> strncat(stringInBase64, &BASE64TABLE[numberOfCharInBase64], 1);
105     >> }

```

Imagem 9 – Conversão de stringBinary para base64

No próximo passo (Imagem 10), temos que adicionar “padding(preenchimento)” ao final de *stringInBase64*, caso seja necessário. Para isso, na linha 108 criamos uma variável de nome *padding* do tipo *char* que guarda o caractere ‘=’, na linha 109 o programa redefine o valor da variável *count* dividindo ela por 2. No *while*, enquanto *count* for maior que 0, utilizamos a função *strncat(stringInBase64, &padding, 1)* para concatenar *padding* ao final de *stringInBase64* e na linha 110 diminuímos o valor de *count* em menos uma unidade com *--count*. Ao final do processo, *stringInBase64* terá ‘=’, ‘==’ ou nenhum *padding*. Se o valor de *count* inicialmente for 0, então não será adicionado *padding* ao final da *stringInBase64*.

```

107  >> /** Colocando padding no final da conversão, caso seja necessário **/
108  >> char padding = '=';
109  >> count = count / 2;
110  ▼  >> while(count > 0){
111     >> >> strncat(stringInBase64, &padding, 1);
112     >> >> --count;
113     >> }

```

Imagem 10 – Adicionando Padding

Enfim, chegamos ao ultimo passo. Agora com *stringBinary* convertida para a *base64* em *stringInBase64* e com o *padding* devidamente colocado, podemos exibir o resultado na tela, com o comando da imagem 11 abaixo:

```

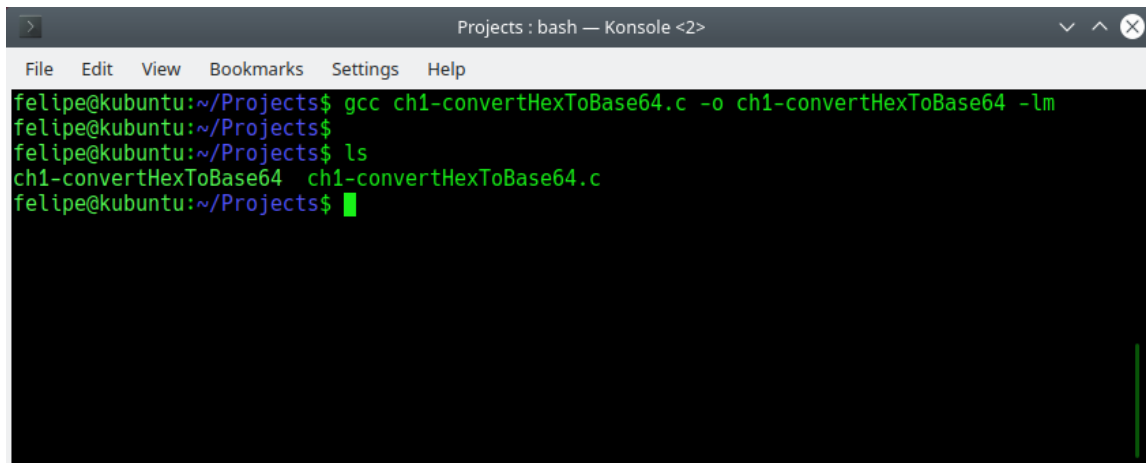
114
115     printf("\nResultado da conversão para base 64: \n\n%s\n", stringInBase64);
116

```

Figura 11 – Exibindo resultado da conversão

8 – Testando o programa

Para testar o programa, basta compilar com o comando na imagem abaixo:

A terminal window titled "Projects : bash — Konsole <2>" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is "felipe@kubuntu:~/Projects\$". The user enters "gcc ch1-convertHexToBase64.c -o ch1-convertHexToBase64 -lm", then "ls", and finally "ch1-convertHexToBase64 ch1-convertHexToBase64.c".

```
felipe@kubuntu:~/Projects$ gcc ch1-convertHexToBase64.c -o ch1-convertHexToBase64 -lm
felipe@kubuntu:~/Projects$ ls
ch1-convertHexToBase64  ch1-convertHexToBase64.c
felipe@kubuntu:~/Projects$
```

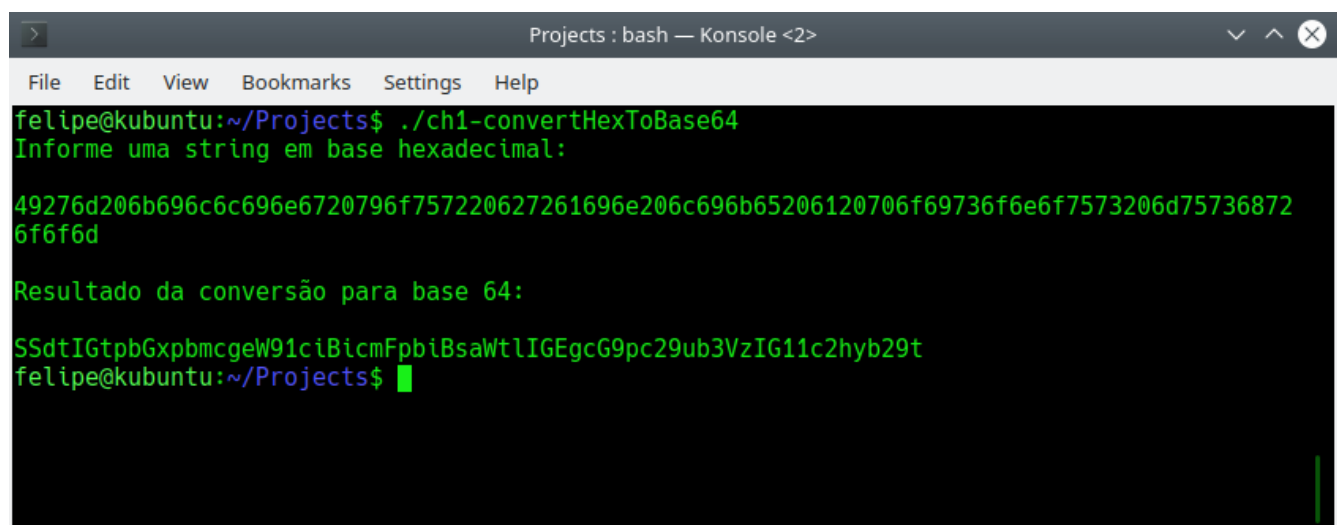
Passamos o comando **gcc** para indicar que estamos querendo compilar um programa em c/c++, passamos o arquivo (ch1-convertHexToBase64.c). A flag **-o** serve para passar o nome do arquivo de saída (ch1-convertHexToBase64) e **-lm** para compilar com a biblioteca math.h inclusa.

Executar o comando **ls** para verificar se o arquivo foi compilado corretamente, ou seja, se o arquivo de saída esta presente na pasta.

Por fim, basta rodar o programa com o comando **./nome-do-arquivo** como descrito na próxima imagem. Para testar vamos utilizar a string do enunciado deste exercício no Cryptopals:

*Entrada:*49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d

*Resultado:*SSdtIGtpbGxpbmcgeW91ciBicmFpbjBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t

A terminal window titled "Projects : bash — Konsole <2>" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is "felipe@kubuntu:~/Projects\$". The user enters "./ch1-convertHexToBase64", followed by the input string "49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d". The program outputs "Informe uma string em base hexadecimal:" and "Resultado da conversão para base 64:". The user then enters the expected output "SSdtIGtpbGxpbmcgeW91ciBicmFpbjBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t".

```
felipe@kubuntu:~/Projects$ ./ch1-convertHexToBase64
Informe uma string em base hexadecimal:
49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d

Resultado da conversão para base 64:
SSdtIGtpbGxpbmcgeW91ciBicmFpbjBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t
felipe@kubuntu:~/Projects$
```