Programação Imperativa em C e C++
Teste final, 28 de Junho de 2008

1. [5] Assinale a única alternativa correcta que completa cada uma das frases. Cada frase assinalada com uma **alternativa incorrecta desconta metade da cotação da frase** ao total do grupo.

a) [1] Após as declarações `char c=0xFF; char *p=&c;` a instrução `printf("%d", *(++p))` ...
- ☐ ... escreve sempre `0xFF`.
- ☒ ... escreve, provavelmente, um valor diferente em cada execução.
- ☐ ... dá erro de *runtime* porque a *string* de formatação devia ser `"%c"`.

b) [1] Em C, a definição de funções com o mesmo nome, mas em módulos diferentes da mesma aplicação, resulta em ...
- ☒ ... erros de ligação.
- ☐ ... erros de compilação.
- ☐ ... erros de compilação e de ligação.

c) [1] Considere o ponto de entrada de um programa C: `int main(int argc, char* argv[])`. A instrução que mostra no *standard output* o segundo argumento indicado na linha de comandos é ...
- ☐ ... `printf("%s", ((&argv)+1)+1 )`
- ☐ ... `printf("%s", argv+2 )`
- ☒ ... `printf("%s", (argv+1)[1] )`

d) [1] Em C++, dada a declaração da variável local `C1 c;` a instrução `delete &c;` pode gerar erro de *runtime* porque …
- ☐ ... o elemento apontado não está em memória automática.
- ☒ ... o elemento apontado não está em memória dinâmica.
- ☐ ... o operador `delete` não recebe um ponteiro.

e) [1] Dadas as definições:

```
class A { public: virtual int f() { return 1; }   int g() { return 3; } };
class B: public A { public: int f() { return 5; } int g() { return 4; } };
void h( A* p ) { cout << ( p->f() + p->g() )<<' '; }
```

o troço de código: `A a; B b; h( &a ); h( &b );` escreve no *standard output*...

- ☒ ... **4 8**
- ☐ ... **4 9**
- ☐ ... **4 4**

2. **[10]** Considere um sistema de cifra que consiste em rodar os bits de cada *byte* de informação para a esquerda (cifrar) ou para a direita (decifrar).

Para cifrar/decifrar os *bytes*, o sistema usa ciclicamente os valores de uma chave (sequência de valores de 0 a 7). A chave é obtida no momento da iniciação do cifrador/decifrador a partir duma *string*. Por exemplo: se a *string* dada como chave for "315", o cifrador rodará 3 bits para a esquerda o 1º,4º,7º… byte, 1 bit para a esquerda o 2º,5º,8º… byte e 5 bits para a esquerda o 3º,6º,9º… byte.

O código apresentado constitui o ponto de partida para a implementação do sistema de cifra descrito.

```
typedef unsigned char byte;   /* Unidade de informação a cifrar/decifrar */

struct StateRotate {
  unsigned chaveLength; /* número de valores da chave. */
  byte *chave;          /* Ponteiro para os valores da chave. */
  byte *valPtr;         /* Ponteiro para o valor da chave a usar no
                           próximo passo de cifra/decifra. */
};
/* variável global a usar pelo cifrador/decifrador. */
struct StateRotate stateRotate;

int initRotate( const char *key ){
  byte * p;
  unsigned i, sz = stateRotate.chaveLength = strlen( key );
  if(p = (byte *)malloc(sz) )
      for( i=0; i< sz; ++i )
            p[i] = (key[i]-'0') & 07;
  stateRotate.valPtr = stateRotate.chave = p ;
  return p != 0;
}
```

Implemente em C as funções:

a) **[2]** `byte rotateLeft(byte b, int numBits)`, que retorna o valor do *byte* `b`, depois de rodado `numBits` (valor de 0 a 7) bits para a esquerda. Por exemplo, o *byte* 11100010 em binário rodado 2 bits para a esquerda ou 6 bits para a direita fica 10001011.

```
byte rotateLeft(byte b, int numBits) { return (b << numBits) | (b >> (8 - numBits)); }
```

b) **[2]** `byte stepRotate(byte b)`, que realiza um passo de cifra, retornando o *byte* `b` cifrado de acordo com o estado da variável `stateRotate`.

```
byte stepRotate(byte b) {
    int numBits = *(stateRotate.valPtr);
    if (++(stateRotate.valPtr) == stateRotate.chave + stateRotate.chaveLength)
        stateRotate.valPtr = stateRotate.chave;
    return rotateLeft(b, numBits);
}
```

c) **[2]** `byte unstepRotate(byte b)`, que realiza um passo de decifra, retornando o *byte* `b` decifrado de acordo com o estado de `stateRotate`.

```
byte unstepRotate(byte b) {
    int numBits = (8 - *(stateRotate.valPtr)) & 0x07;
    if (++(stateRotate.valPtr) == stateRotate.chave + stateRotate.chaveLength)
        stateRotate.valPtr = stateRotate.chave;
    return rotateLeft(b, numBits);
}
```

d) **[1]** `void endRotate()`, que liberta os recursos reservados em `initRotate()`.

```
void endRotate() { free(stateRotate.chave); }
```

e) [3] `void cifraRotate(const char *key)`, que cifra, usando este sistema, os bytes lidos do *standard input* e escreve-os no *standard output*, usando a chave `key`.

```
/* Assumindo que o standard input e output estão configurados para modo binário. */
void cifraRotate(const char * key) {
    int c;
    if (initRotate(key)) {
        while ((c = getchar()) != EOF)  putchar(stepRotate(c));
        endRotate();
    }
}
```

3. [5] Para suportar o sistema de cifra apresentado no ponto anterior, defina a classe `RotateSystem`.

O código do construtor dessa classe será:

```
RotateSystem::RotateSystem( const char *key ){
  valPtr = chave = new byte[chaveLength = strlen( key )];
  for(unsigned i=0; i< chaveLength; ++i ) chave[i] = (key[i]-'0') & 07;
}
```

De acordo com a definição do construtor: Declare o construtor e os campos necessários; Declare e implemente o destrutor e todos os métodos necessários para responder aos requisitos das alíneas a), b) e c) do ponto anterior. Tenha em atenção a visibilidade e a necessidade dos métodos serem ou não estáticos.

File RotateSystem.h :

```
typedef unsigned char byte;
class RotateSystem {
    unsigned chaveLength;
    byte * chave, * valPtr;

    static byte rotateLeft(byte b, int numBits)
        { return (b << numBits) | (b >> (8 - numBits)); }

    RotateSystem(const RotateSystem & other);              // Proibir cópias.
    RotateSystem & operator=(const RotateSystem & other); // Proibir cópias.
public:
    RotateSystem(const char * key);
    ~RotateSystem() { delete [] chave; }
    byte step(byte b);
    byte unstep(byte b);
};
```

File RotateSystem.cpp :

```
byte RotateSystem::step(byte b) {
    int numBits = *valPtr;
    if (++valPtr) == chave + chaveLength) valPtr = chave;
    return rotateLeft(b, numBits);
}

byte RotateSystem::unstep(byte b) {
    int numBits = (8 - *valPtr) & 0x07;
    if (++valPtr) == chave + chaveLength) valPtr = chave;
    return rotateLeft(b, numBits);
}
```