



Nome: _____

Número: _____ Turma: _____

1. [4] Assinale a única alternativa correcta que completa cada uma das frases. Cada frase assinalada com uma **alternativa incorrecta desconta metade da cotação** ao total do grupo.

- a) [1] Dada a declaração das variáveis globais: `int b, a[2]={1,2};` a instrução `b=a[2];` ...
- ☐ ... dá origem a um erro de *linker* devido à indexação fora dos limites.
 - ☐ ... dá origem a um erro de compilação devido à indexação fora dos limites.
 - ☐ ... gera uma excepção, em tempo de execução, devido à indexação fora dos limites.
 - ☐ ... não dá origem a qualquer erro de *linker* ou de compilação.
- b) [1] Dada a declaração: `const char *a="PICC";` a única expressão com valor diferente de zero é ...
- ☐ ... `sizeof(*(a+2)) == strlen(a+2)`
 - ☐ ... `(int)*(a+2) == (int)&a[2]`
 - ☐ ... `sizeof(a[2]) == sizeof(a)+2`
 - ☐ ... `sizeof(a) == sizeof(&a[2])`
- c) [1] Dada a função: `int f(int *a, char *b) { int v = *(a++); return (b-(char*)a)+v; }` e o array: `int v[]={1,2,3};` Numa arquitectura a 32 *bits*, a chamada `f(v, (char*)v+2);` retorna...
- ☐ ... 5
 - ☐ ... -1
 - ☐ ... 9
 - ☐ ... 3
- d) [1] Sejam X e Y classes independentes em C++, cada uma com um campo de instância do tipo `int` e um destrutor virtual. `sizeof(X)` aumenta se a classe X for alterada para ...
- ☐ ... ter um método virtual.
 - ☐ ... ter um construtor por cópia.
 - ☐ ... ter mais um campo: `static Y *py;`
 - ☐ ... derivar de Y.

2. [10] Implemente as seguintes funções em C.

- a) [2] Escreva a função `stredit`, que preenche a zona de `str` que começa no carácter com o índice `idx` e termina `dim` caracteres depois com o conteúdo da *string* `data`, completando-o com espaços se for necessário. Por exemplo, se `s` for a *string* "ABCDEFGH IJ", `stredit(s,2,1,"xyz")` deixa `s` com "ABxDEF GHIJ" e `stredit(s,3,5,"xyz")` deixa `s` com "ABCxyz IJ".

```
void stredit(char * str, size_t idx, size_t dim, const char * data);
```

- b) [2] Implemente a função `memclean` que preenche com o valor 0 os *bytes* de um conjunto de blocos de memória, cada um com `bksize` *bytes*. O parâmetro `blocks` é um *array* de ponteiros terminado com `NULL` que indica os blocos de memória a limpar.

```
typedef unsigned char * byteptr;
void memclean(byteptr * blocks, size_t bksize);
```

- c) [3] Codifique a função `bomSalario` para deixar no *array* `alunos`, com `numAlunos` posições, apenas as entradas correspondentes a alunos cuja nota da disciplina com o nome "PICC" seja pelo menos 15. A função deve retornar o número de entradas restantes no *array*. Não é necessário libertar a memória das instâncias removidas.

```
size_t bomSalario(const Aluno * alunos[],
                  size_t numAlunos);
```

```
typedef struct _nota {
    const char * nomeDisciplina;
    unsigned char valores;
} Nota;

typedef struct _aluno {
    unsigned int numero;
    Nota notas[MAX_NOTAS];
    unsigned int numNotas;
} Aluno;
```

- d) [3] Desenvolva a função `ptrs2vals` que recebe um *array* de ponteiros para `num` objectos de dimensão `dim` e retorna um novo *array*, alocado dinamicamente, contendo cópias desses objectos, em vez de ponteiros. O *array* original, bem como os objectos apontados por estes, foram alojados dinamicamente e devem ser libertados.

```
void * ptrs2vals(void * * ptrs, size_t num, size_t dim);
```

3. [6] Considere a definição da classe `Array` e um programa que usa essa classe, sendo apresentado em comentário o respectivo *output*.

```
class Array {
protected:
    size_t size, cap;    // número de elementos guardados e capacidade máxima.
    int *elems;          // array de inteiros alojado dinamicamente.
public:
    Array(size_t c=10): size(0) { elems=(int*)malloc((cap=c)*sizeof(int)); }
    /* ... */
    ~Array()              { free(elems); }
    void add(int elem)     { if (size<cap) elems[size++]=elem; }
    size_t length() const { return size; }
    int remLast()          { return elems[--size]; }
};
```

Esta classe implementa um array de inteiros em que a capacidade máxima pode ser indicada no construtor.

O método `add()` acrescenta mais um elemento no fim do *array* desde que exista capacidade.

O método `remLast()` retorna e remove o último elemento, mas só pode ser chamado se existirem elementos.

```
void print(const char *name, Array a)
{
    Array aux;
    printf("%s = { ", name);
    aux = a;
    while(aux.length()>0)
        printf("%d ", aux.remLast());
    printf("}\n");
}

Array ga;

void main() {
    Array *la = new Array(5);
    la->add(10); la->add(20);
    print("la", *la);    // la = { 20 10 }
    delete la;
    print("ga", ga);     // ga = { }
    putchar('.');        // .
}
```

- a) [2] Declare e implemente o construtor por cópia e o operador de afectação da classe **Array** para que o programa funcione correctamente.
- b) [1] Até ao momento imediatamente antes de ser escrito o ponto final, quantas vezes foi chamado cada um dos construtores e destrutores? Justifique cada chamada.
- c) [2] Defina a classe derivada **DinArray** por forma a que o método `add()` acrescente sempre o elemento. Quando não houver capacidade suficiente deve ser alojado um *array* com o dobro da capacidade actual, ou com 10 elementos se a capacidade actual for zero.
- d) [1] Para testar a classe **DinArray**, o programa apresentado foi alterado apenas na primeira linha da função `main()` para: `Array *la = new DinArray(1);` mas o programa não produz o *output* verificado. Porquê?