O programa na página 2 é escrito em uma linguagem de programação que tem a mesma sintaxe e quase a mesma semântica da linguagem C, exceto pelas declarações aninhadas de funções, inspiradas em Pascal.

O arquivo `memoria.docx` traz uma representação da área de dados do programa na memória. Cada unidade de memória é representada por um retângulo, tendo acima o endereço da unidade de memória. Abaixo da unidade de memória, há uma grade para se anotar o histórico dos valores que as respectivas unidades de memória contiveram. São 16 unidades de memória ao todo na área de dados. Uma variável ocupa exatamente uma unidade de memória. Idem para um campo de registro (*struct*).

O usuário digita valores enquanto o programa está em funcionamento. Eles são obtidos pelo programa através da função `scanf` ou da função fictícia `int read()`.

As variáveis `argc` e `argv` de `main` não precisam, nem devem, ser representadas nas respostas.

(1) Para o programa da página 2:

a) Considerando que o programa contém variáveis globais, variáveis locais, variáveis locais anotadas com *static* e dados alocados dinamicamente, divida a memória em sub-áreas para os diferentes tipos de alocação de memória. Use a terminologia técnica para nomear as sub-áreas criadas.

b) Considerando que a linguagem de programação adota o escopamento dinâmico,
   - Mostre a evolução dos valores de todos os dados do programa e os resultados impressos na tela se o histórico de valores digitados pelo usuário for 13, -3 e 15.
   - Qual é o ambiente de referenciamento no bloco entre as linhas 24 e 28 quando ele é executado pela primeira vez?

c) Idem, mas considerando uma linguagem de programação com escopamento estático. Indique o escopo de cada variável ao lado do programa (use as linhas auxiliares).

d) Esse programa apresenta fuga de memória?

(2) O programa da página 3 adota uma variação ainda mais inusitada em relação à linguagem C: não há tipos. As declarações de variáveis iniciam com a palavra `var`, e as de funções, com `fun`. Para resolução, use o arquivo `memoria20.docx`. Os dados fornecidos pela linha de comando são como em C: na chamada `programa 33 4`, `argv[1]="33"` e `argv[2]="4"`. A função `atoi()` converte uma string em inteiro.

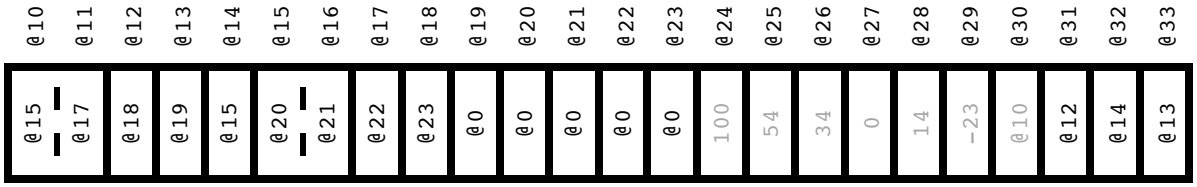
a) Divida a memória em sub-áreas conforme a questão 1, item a.

b) Considerando que a linguagem de programação adota o escopamento dinâmico,
   - Mostre a evolução dos valores de todos os dados do programa e os resultados impressos na tela se o programa foi invocado com a a expressão `programa 17 256 -1 269`.
   - Qual é o ambiente de referenciamento no bloco entre as linhas 17 e 22 quando ele é executado pela quarta vez?

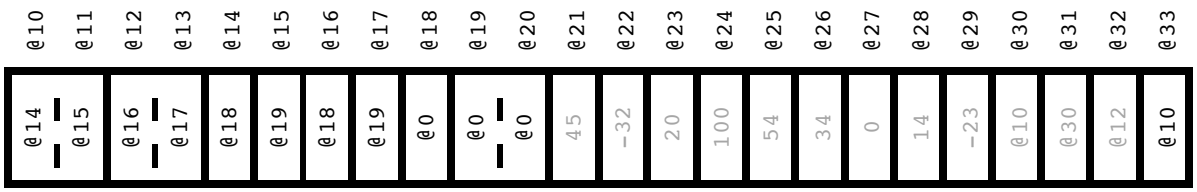
c) Idem, mas considerando uma linguagem de programação com escopamento estático e a chamada com a expressão `programa 100 -1 200 1`.

d) Esse programa apresenta fuga de memória?

(3) O desenho a seguir representa a área de memória de um programa. A pilha cresce de cima para baixo tendo sua base no endereço `@33`. O *heap* ocupa a faixa entre `@10` e `@25`. As células `@10` e `@11` correspondem a um registro com dois campos, assim como `@15` e `@16`. A execução de uma função acaba de se encerrar e a pilha recuou. O topo da pilha está em `@31`. Ilustre as duas fases do algoritmo de coleta de lixo aplicado a essa situação.



(4) Idem para a figura abaixo:



```
1  #include <stdio.h>
2
3  int a = 34, b = -10;
4
5  typedef struct _w {
6        int m;
7        struct _w *v:
8  } W;
9
10 void f() {
11       int i;
12       static int c = 100;
13
14       i = a + b;
15       if (c > 0)
16             scanf("%d", &c);
17       b = a++;
18 }
19
20 void g(W *e) {
21       int a;
22       W *d;
23
24       void h(W *i, W *k) {
25             i->m = (k->m)++;
26             i->v = NULL;
27             k->v = i;
28       }
29
30       a = b;
31       d = (W*) malloc(sizeof(W));
32
33       f();
34
35       h(d, e);
36 }
37
38 void main(int argc, char* argv[]) {
39       int b;
40       W *d;
41
42       a = 18;
43       b = 3;
44       d = (W*) malloc(sizeof(W));
45       d->m = 33;
46       d->v = NULL;
47
48       if (read() > b)
49             f();
50
51       scanf("%d", &a);
52
53       g(d);
54
55       free(d);
56
57       printf("%d", a + b);
58 }
```

```
1  #include <stdio.h>
2
3  struct single {
4        s;
5  }
6
7  struct ddouble {
8        s;
9        d;
10 }
11
12 var i = 1, z=32, p=NULL;
13
14 fun f (b) {
15       var a, d[2];
16
17       fun g(q, r) {
18             q->s = p;
19             r->s = q;
20             p = r;
21             z--;
22       }
23
24       for (a=0;a<2;a++)
25             d[a]=malloc(1);
26       g(d[0], d[1]);
27       if( i<5 && atoi(argv[i++]) > 225)
28             g(b, d[0]);
29 }
30
31 fun h() {
32       var pp = malloc(2);
33       var p = malloc(1);
34       p->s=NULL;
35       f(pp);
36       printf("%d\n", z + i);
37 }
38
39 fun main(argc, argv) {
40       var z = 12;
41       var ptr = malloc(2);
42
43       if ( i<5 && atoi(argv[i++]) <= 96)
44             f(ptr);
45
46       free(ptr);
47
48       if ( i<5 && atoi(argv[i++]) == -1)
49             h();
50
51 }
```