Gestão dinâmica da memória

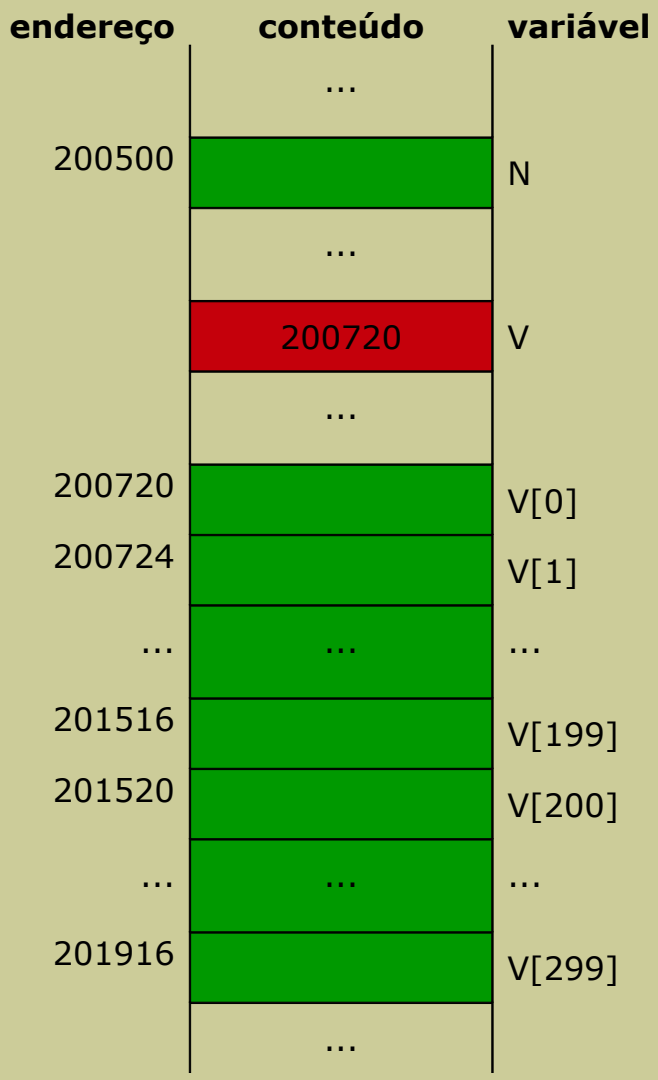
Memória Estática

Exemplo

```
#include <stdio.h>
void main()
{
  int  V[300], N, k;
  do{
     printf (“N = ?”);
     scanf (“%d”, &N);
  }while (N < 1 || N > 300);
  for (k = 0; k < N; k = k + 1) {
     printf (“Insira um inteiro: “);
     scanf (“%d”, &V[k]);
  }
}
```

Exemplo

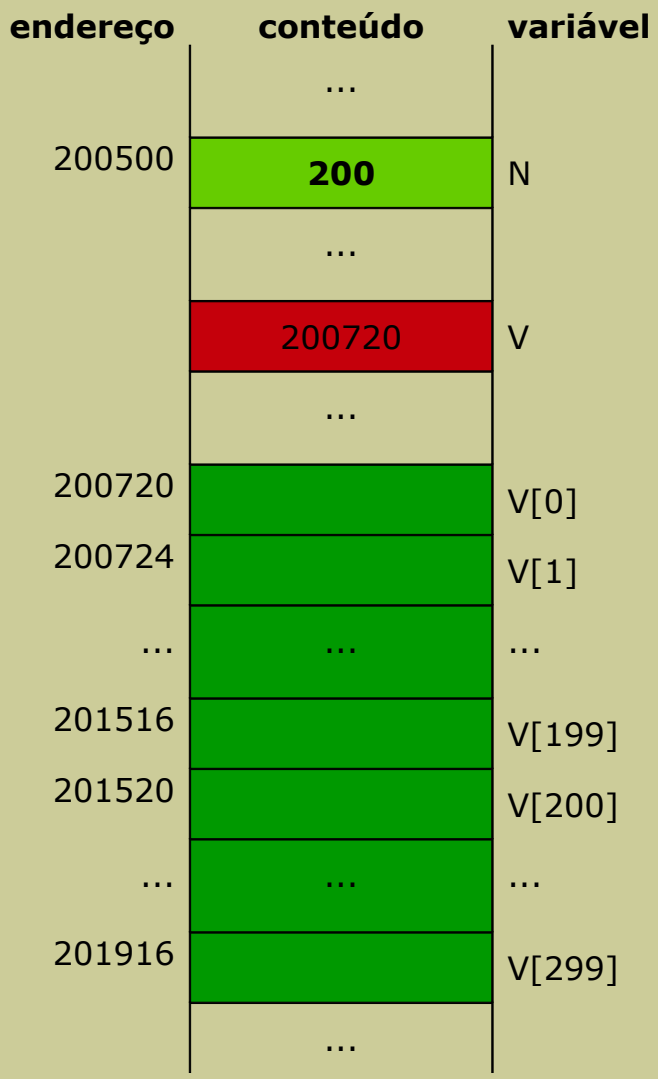
```
#include <stdio.h>
void main()
{
  int  V[300], N, k;
  do{
     printf ("N = ?");
     scanf ("%d", &N);
  }while (N < 1 || N > 300);
  for (k = 0; k < N; k = k + 1)
  {
     printf ("Insira um inteiro: ");
     scanf ("%d", &V[k]);
  }
}
```



Memória reservada pelo programa

Exemplo

```
#include <stdio.h>
void main()
{
  int  V[300], N, k;
  do{
     printf ("N = ?");
     scanf ("%d", &N);
  }while (N < 1 || N > 300);
  for (k = 0; k < N; k = k + 1)
  {
     printf ("Insira um inteiro: ");
     scanf ("%d", &V[k]);
  }
}
```

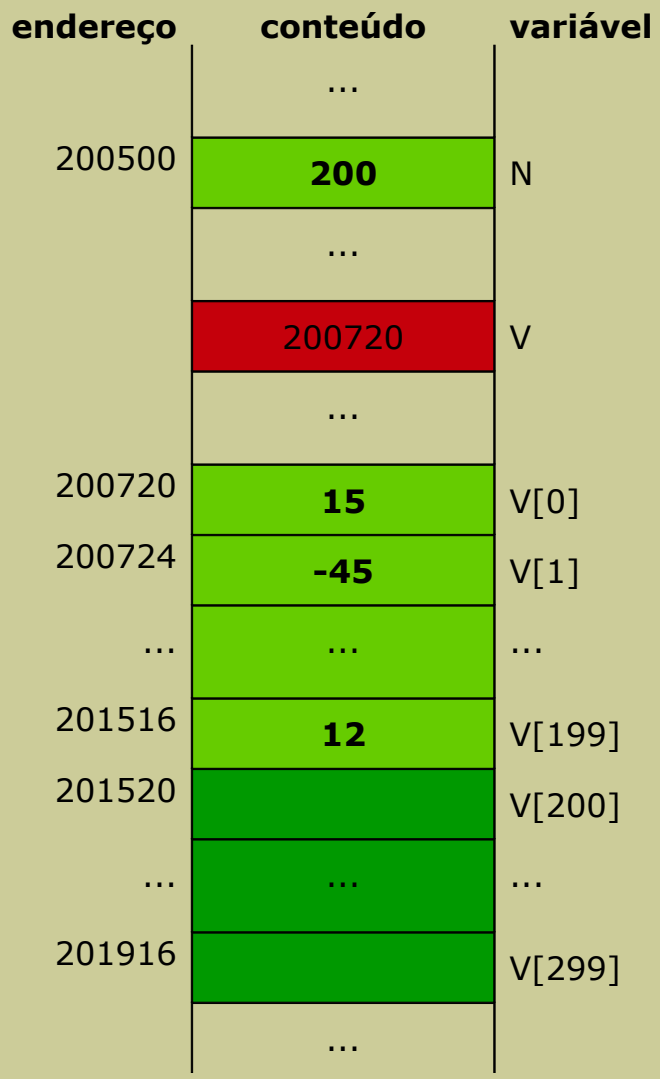


Memória reservada pelo programa

Memória reservada e usada pelo programa

Exemplo

```
#include <stdio.h>
void main()
{
  int  V[300], N, k;
  do{
     printf ("N = ?");
     scanf ("%d", &N);
  }while (N < 1 || N > 300);
  for (k = 0; k < N; k = k + 1)
  {
     printf ("Insira um inteiro: ");
     scanf ("%d", &V[k]);
  }
}
```



Memória reservada pelo programa

Memória reservada e usada pelo programa

Funções associadas à gestão dinâmica de memória

Alocação/reserva de memória

calloc

malloc

Realocação de memória alocada anteriormente

realloc

Libertação de memória alocada/realocada anteriormente

free

Biblioteca

stdlib.h

Alocação de memória com “calloc”

Sintaxe

```
void *calloc (size_t  nmemb,  size_t  size);
```

- Parâmetros da função
  - **nmemb**: número de elementos que se pretende alocar/reservar memória
  - **size**: tamanho dos elementos que se pretende alocar/reservar memória
- Devolução da função
  - o endereço (apontador) para a primeira posição do bloco, ou
  - **NULL** quando não for possível alocar memória

Sintaxe

```
void *calloc (size_t  nmemb,  size_t  size);
```

- Descrição:
  - Reserva um bloco de memória contígua com espaço suficiente para armazenar **nmemb** elementos de dimensão **size** cada elemento
  - **size_t** é o tipo usado para especificar as dimensões numéricas em várias funções
  - o tipo de retorno **void *** corresponde a um endereço genérico de memória (permite a utilização por todo o tipo de ponteiro)
  - todas as posições do bloco de memória são inicializadas com zero

Exemplo

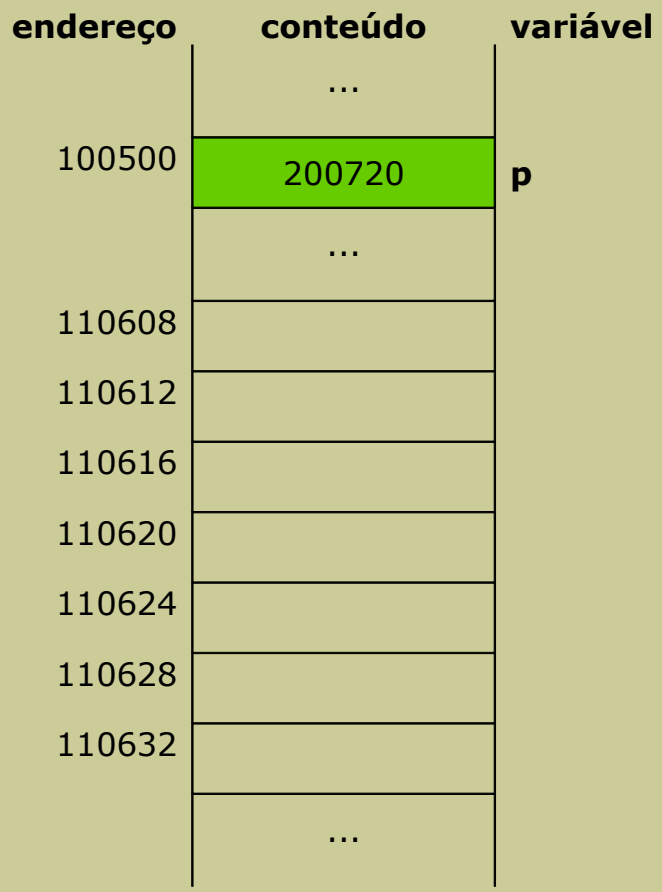
```
float  *p;
p = (float *) calloc (200, sizeof(float));
```

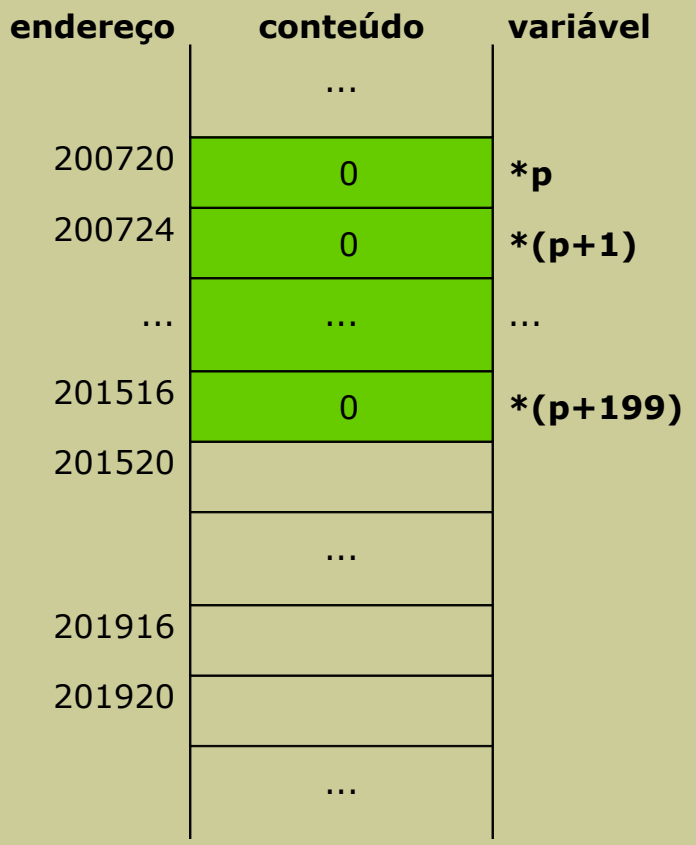
- reserva de memória para um bloco de **200** números reais
- a partir daqui, **p** pode ser tratado como um array 1D de **200** elementos (para **200** números reais)
- **p** é um ponteiro para o primeiro elemento do array
- **sizeof()** é um operador do C que devolve a dimensão (em geral, em bytes) do tipo ou variável indicado no argumento
- **(float *)** funciona como um operador de **cast** (obriga a devolver um ponteiro para uma variável do tipo real)

Exemplo


endereço
conteúdo
variável
...
100500
200720
p
...
110608
110612
110616
110620
110624
110628
110632
...
endereço
conteúdo
variável
...
200720
0
p[0]
200724
0
p[1]
...
....
...
201516
0
p[199]
201520
...
201916
201920
...
Memória reservada e usada pelo programa

Exemplo





Memória reservada e usada pelo programa

Alocação de memória com “malloc”

Sintaxe

```
void *malloc (size_t  total_size);
```

- Parâmetros da função
  - **total_size**: número total em bytes do bloco de memória a alocar/reservar
- Devolução da função
  - o endereço (ponteiro) para a primeira posição do bloco, ou
  - **NULL** quando não for possível alocar memória

Sintaxe

```
void *malloc (size_t  total_size);
```

- Descrição
  - reserva um bloco de memória contígua de dimensão **total_size** expressa em bytes
  - **size_t** é o tipo usado para especificar as dimensões numéricas em funções do C
  - o tipo de retorno **void *** corresponde a um endereço genérico de memória (permite a utilização por todo o tipo de ponteiro)
  - **calloc (n, d)** pode ser substituído simplesmente por **malloc (n*d)**
  - as posições do bloco não são inicializadas com qualquer valor

Realocação de memória com “realloc”

Sintaxe

```
void  *realloc (void *ptr,  size_t  total_new_size);
```

- Parâmetros da função
  - **ptr**: ponteiro para o bloco de memória reservado antes
  - **total_new_size:** dimensão total que se pretende agora para o mesmo bloco
- Devolução da função
  - o endereço (ponteiro) para a primeira posição do bloco redimensionado, ou
  - ***NULL*** quando não for possível alocar memória
- Descrição
  - o segundo argumento (size_t **total_new_size**) tem um significado semelhante ao da função **malloc** (size_t **total_size**)

Libertação de memória com “free”

Sintaxe

```
void  free (void *ptr);
```

- Parâmetros da função
  - **ptr**: ponteiro para o bloco de memória reservado antes, o qual foi devolvido por **malloc**, **calloc** ou **realloc**

Exemplo

Enunciado

Implementar um programa que realize as seguintes ações:

- construir um array com N elementos do tipo inteiro (N > 0)
- aumentar o array anterior em 100 elementos

Programa

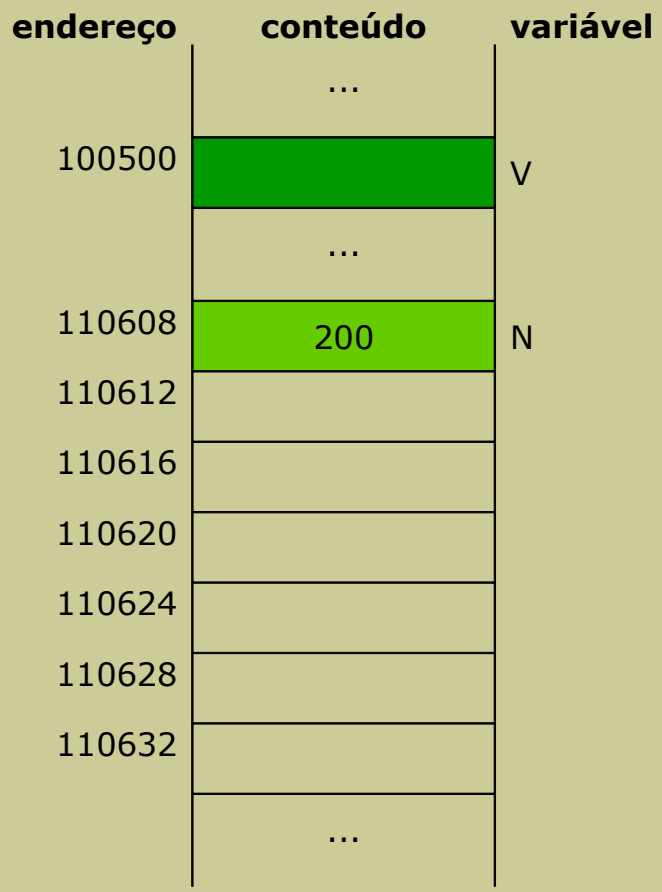
```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int  *V, N;
  do{
     printf("Insira a dimensão do vetor: ");
     scanf("%d", &N);
  }while (N < 1);
  V = (int *) malloc (N * sizeof(int));
  for (i = 0; i < N; i++){
     printf("Inserir um valor inteiro: ");
     scanf("%d", &V[i]);
  }
  V = (int *) realloc (V, (N+100) * sizeof(int));
  free(V);
}
```

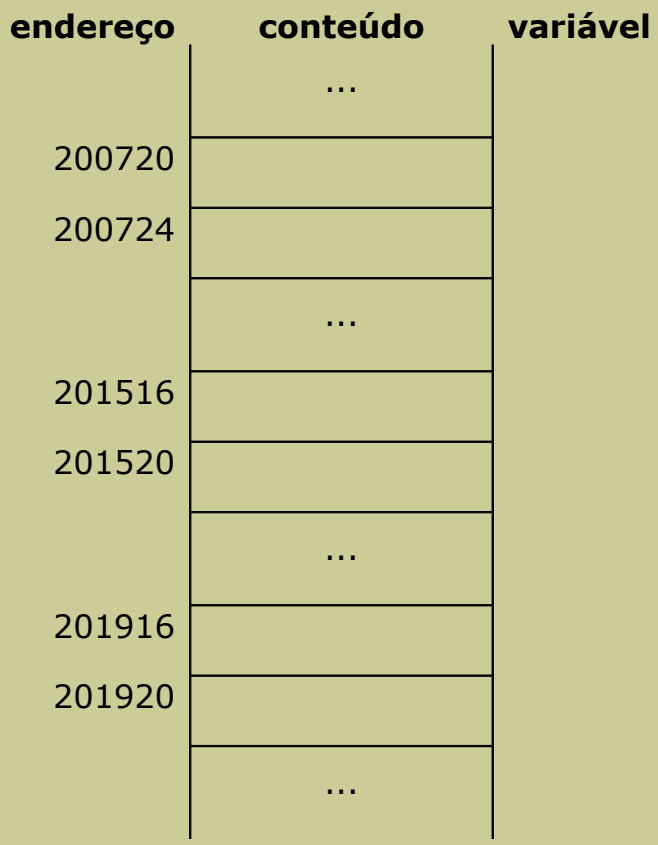
Representação gráfica

Programa

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int   *V, N;
  do{
     printf("Insira a dimensão do vetor: ");
     scanf("%d", &N);
  }while (N < 1);
  V = (int *) malloc (N * sizeof(int));
  for (i = 0; i < N; i++){
     printf("Inserir um valor inteiro: ");
     scanf("%d", &V[i]);
  }
  V = (int *) realloc (V, (N+100) * sizeof(int));
  free(V);
}
```

Representação gráfica





Memória reservada pelo programa

Memória reservada e usada pelo programa

Programa

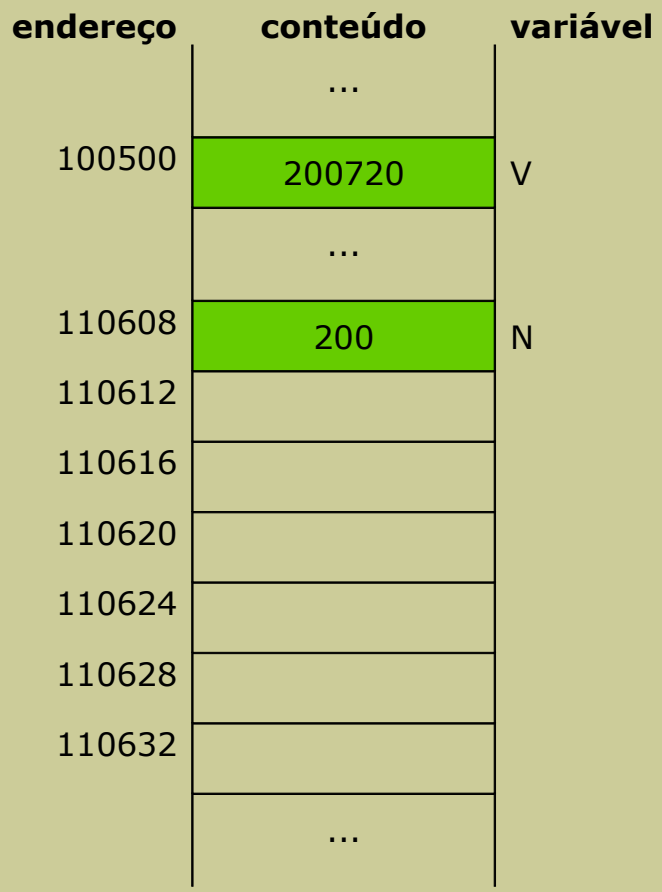
```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
  int    *V, N;
  do{
     printf("Insira a dimensão do vetor: ");
     scanf("%d", &N);
  }while (N < 1);
  V = (int *) malloc (N * sizeof(int));
  for (i = 0; i < N; i++){
     printf("Inserir um valor inteiro: ");
     scanf("%d", &V[i]);
  }
  V = (int *) realloc (V, (N+100) * sizeof(int));
  free(V);
}
```

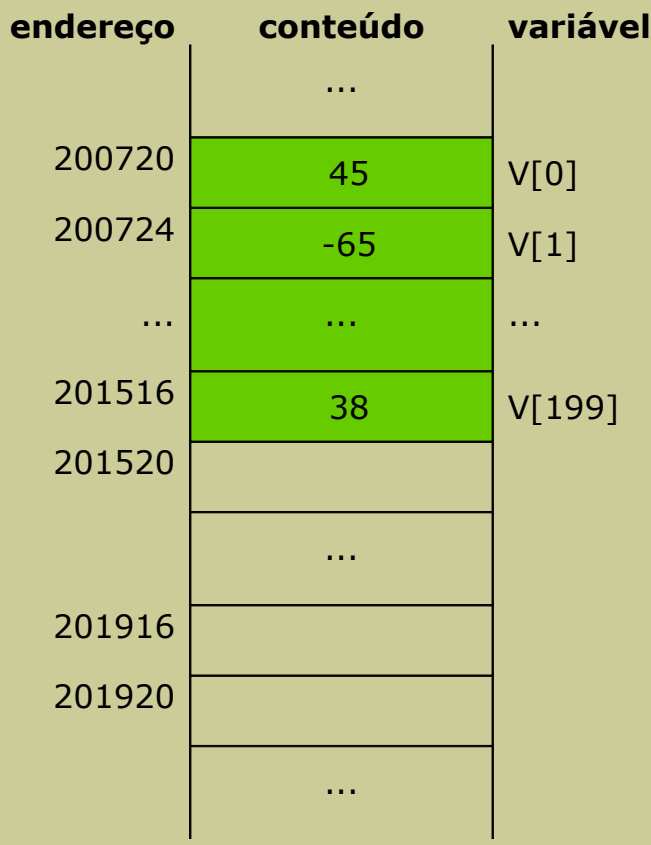
Representação gráfica

Programa

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int    *V, N;
  do{
     printf("Insira a dimensão do vetor: ");
     scanf("%d", &N);
  }while (N < 1);
  V = (int *) malloc (N * sizeof(int));
  for (i = 0; i < N; i++){
     printf("Inserir um valor inteiro: ");
     scanf("%d", &V[i]);
  }
  V = (int *) realloc (V, (N+100) * sizeof(int));
  free(V);
}
```

Representação gráfica



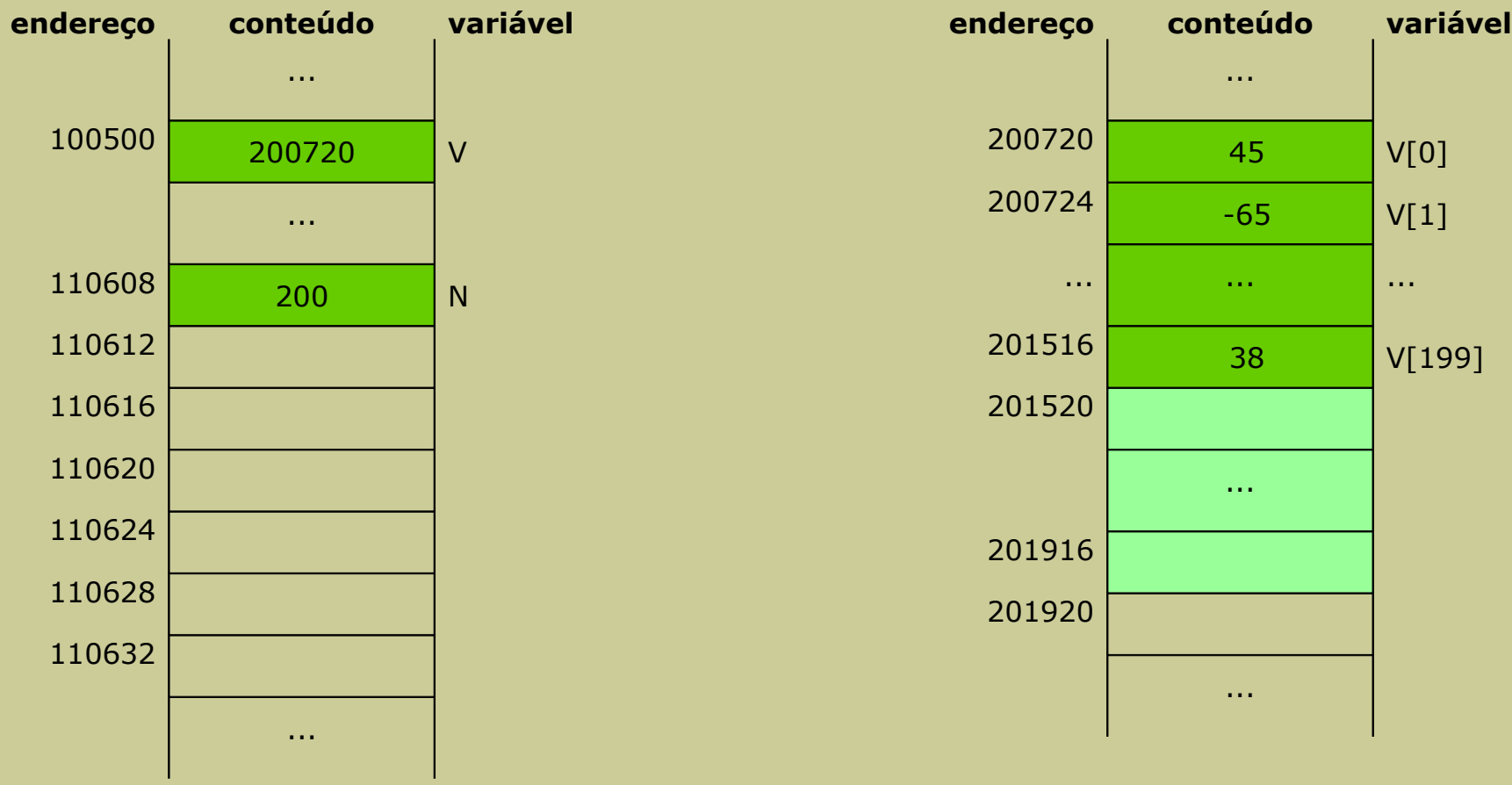


Memória reservada e usada pelo programa

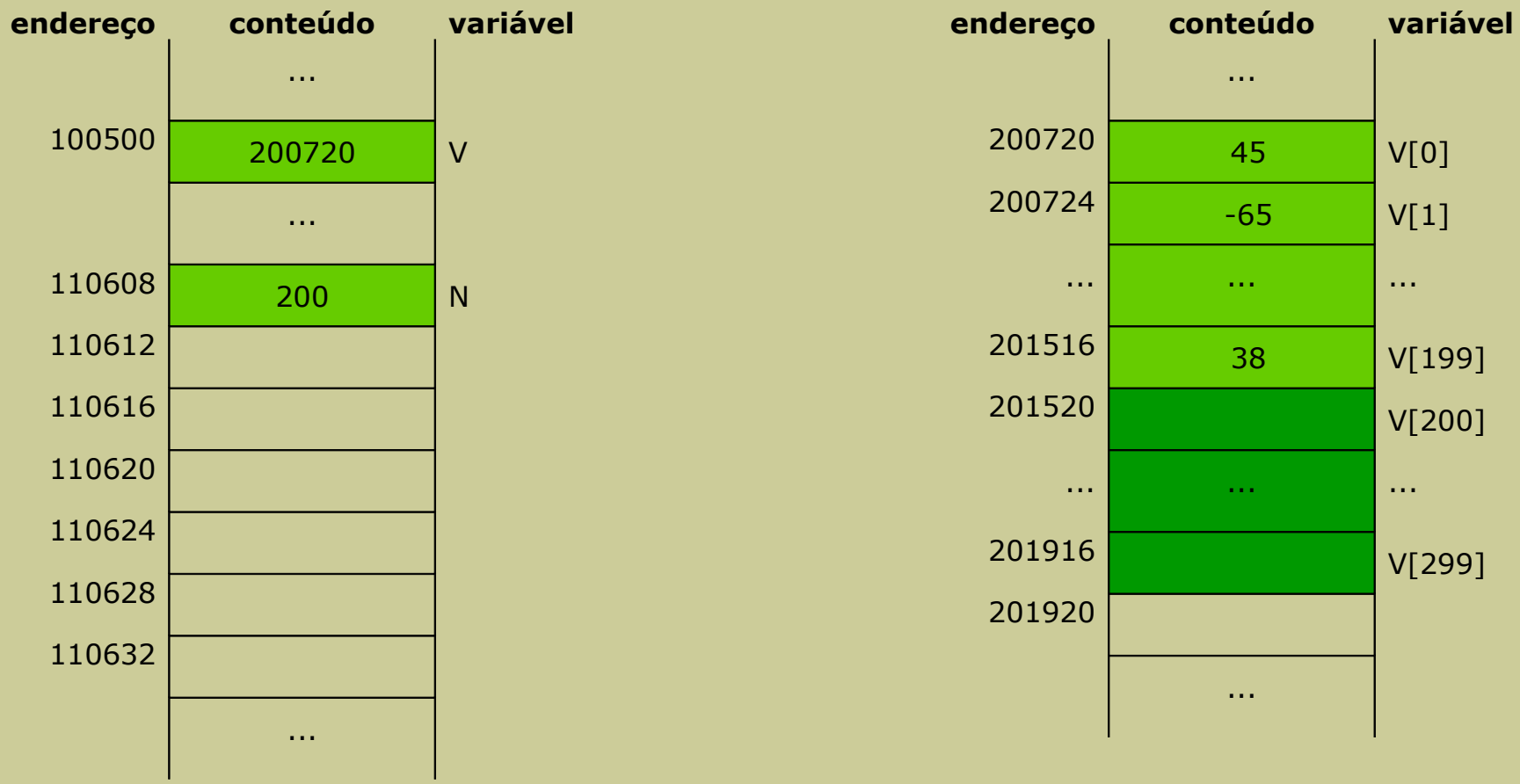
Programa

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
  int    *V, N;
  do{
     printf("Insira a dimensão do vetor: ");
     scanf("%d", &N);
  }while (N < 1);
  V = (int *) malloc (N * sizeof(int));
  for (i = 0; i < N; i++){
     printf("Inserir um valor inteiro: ");
     scanf("%d", &V[i]);
  }
  V = (int *) realloc (V, (N+100) * sizeof(int));
  free(V);
}
```

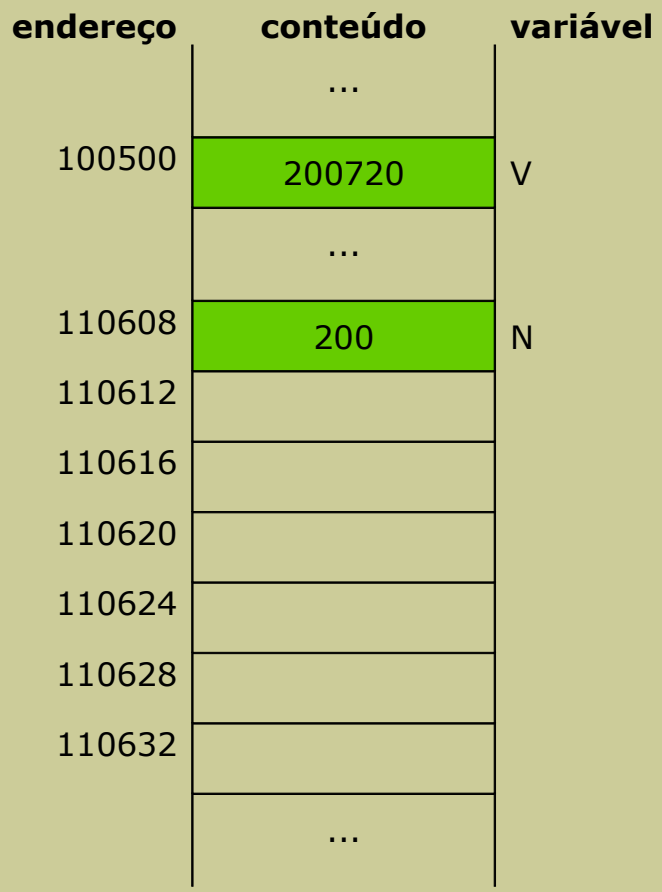
Representação gráfica (caso 1)

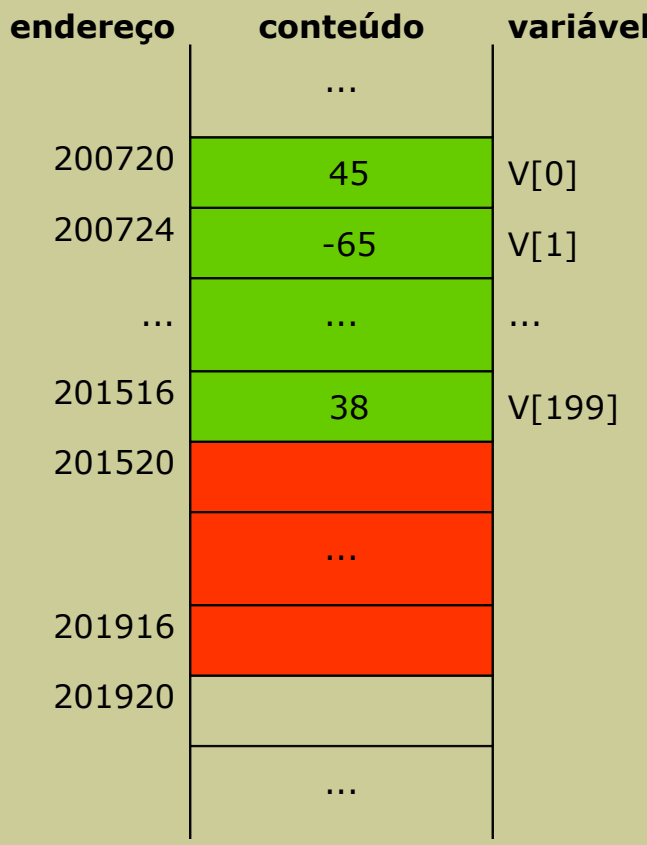


Representação gráfica (caso 1)



Representação gráfica (caso 2)

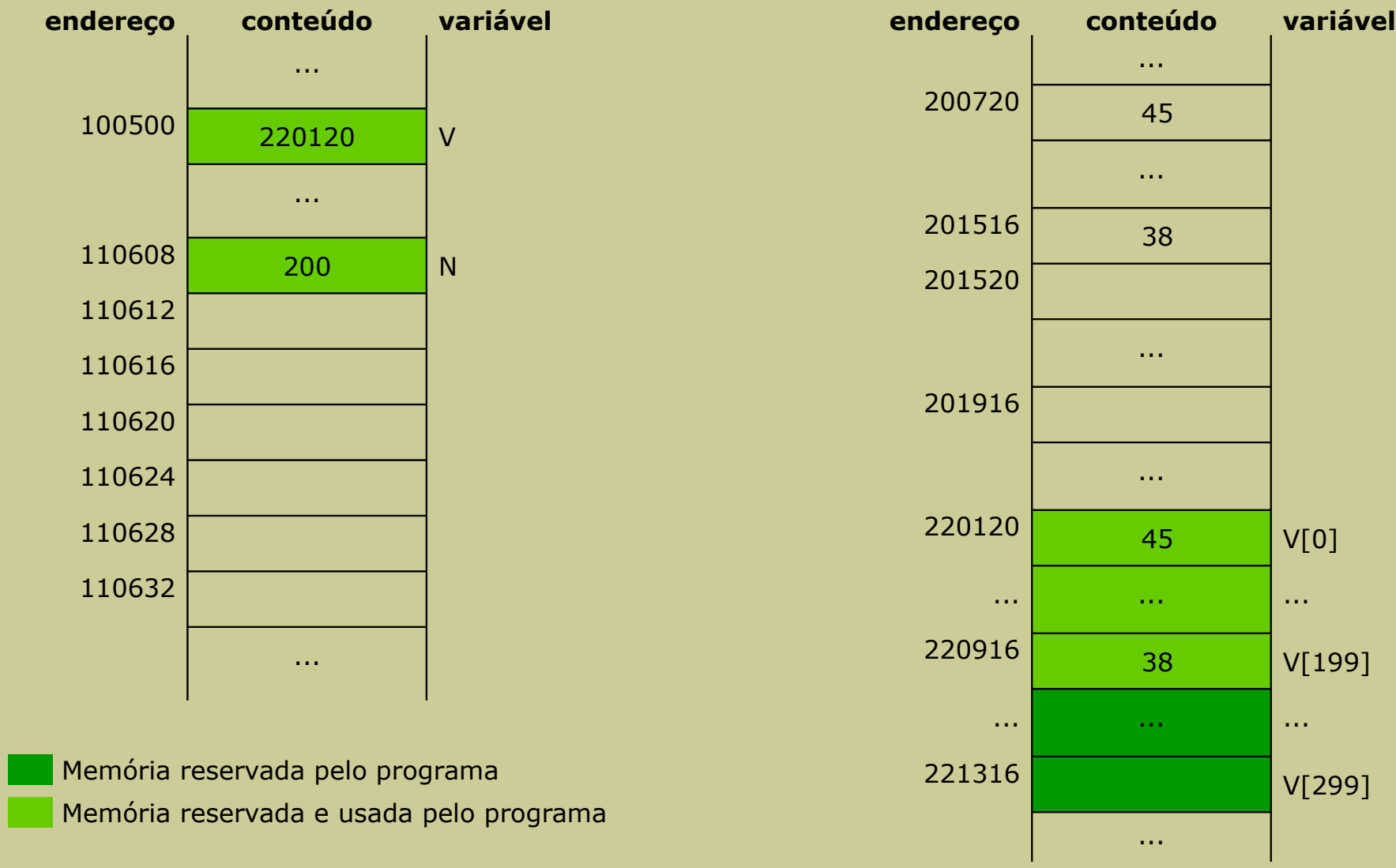




Memória reservada e usada pelo programa

Memória ocupada (ou parte dela)

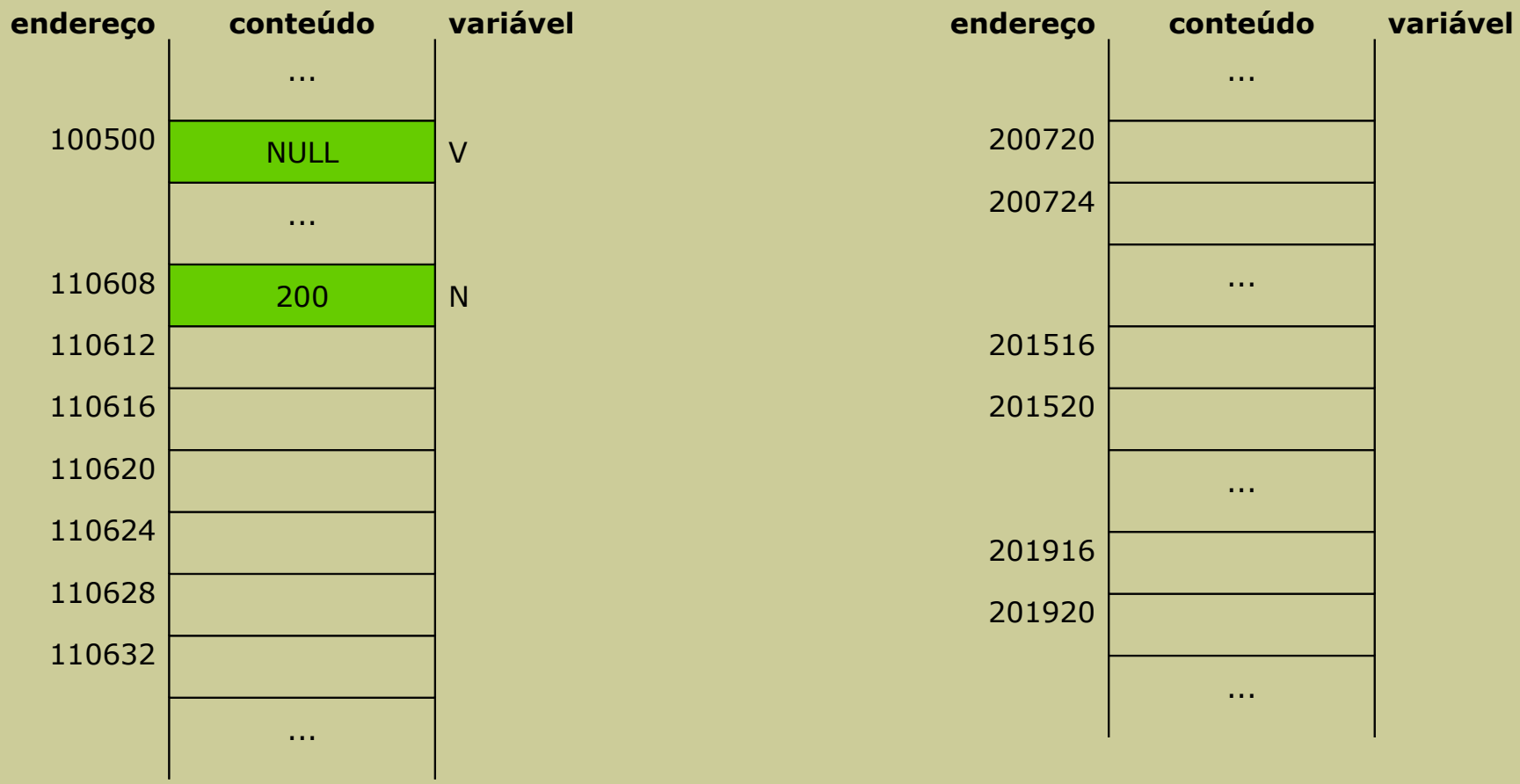
Representação gráfica (caso 2)



Programa

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
  int    *V, N;
  do{
     printf("Insira a dimensão do vetor: ");
     scanf("%d", &N);
  }while (N < 1);
  V = (int *) malloc (N * sizeof(int));
  for (i = 0; i < N; i++){
     printf("Inserir um valor inteiro: ");
     scanf("%d", &V[i]);
  }
  V = (int *) realloc (V, (N+100) * sizeof(int));
  free(V);
}
```

Representação gráfica



Memória reservada e usada pelo programa

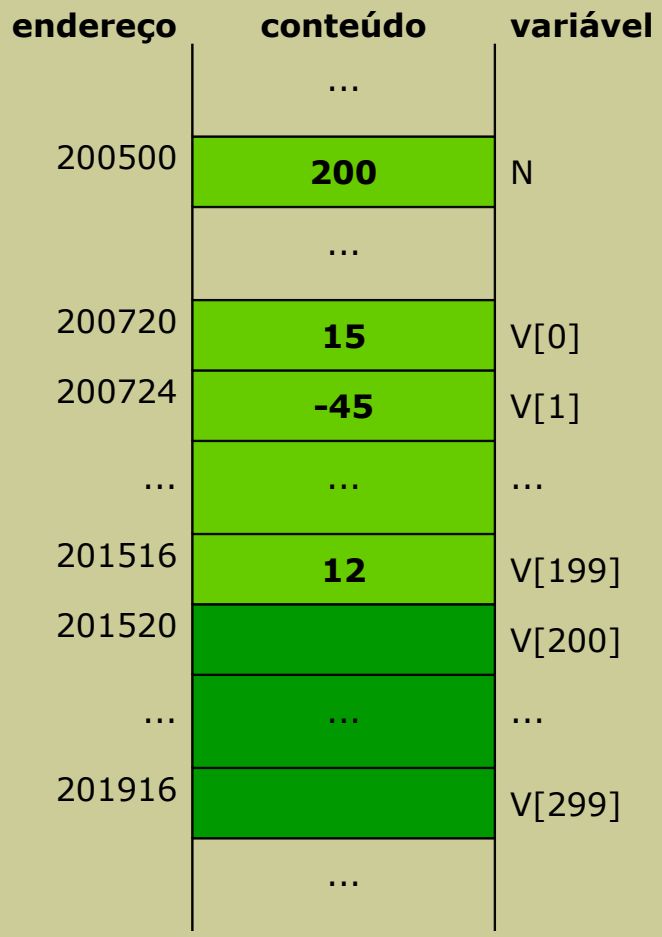
Memória Estática vs. Memória Dinâmica

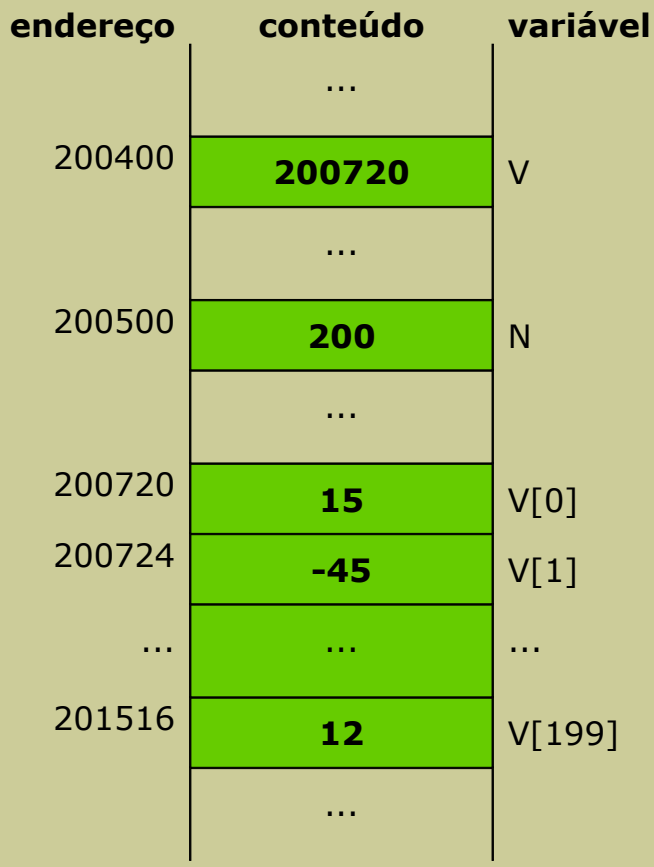
Programas

```
#include <stdio.h>
void main ()
{
  int V[300], N;
  do{
     printf (“N = ”);
     scanf (“%d”, &N);
  }while (N < 1 || N > 300);
  for (i = 0; i < N, i++){
     printf (“Insira um inteiro: “);
     scanf (“%d”, &V[i]);
  }
}
```

```
#include <stdio.h>
#include <stdlib.h>
void main () {
  int  *V, N;
  do{
     printf (“N = ”);
     scanf (“%d”, &N);
  }while (N < 1);
  V = (int*) malloc (N*sizeof (int));
  for (i = 0; i < N, i++){
     printf (“Insira um inteiro: “);
     scanf (“%d”, &V[i]);
  }
  free(V);
}
```

Representações gráficas





Memória reservada pelo programa

Memória reservada e usada pelo programa