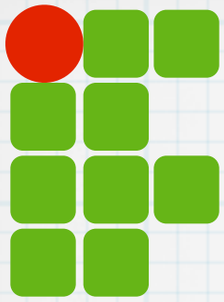
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Algoritmos

ANSI C - Gerenciamento de Memória

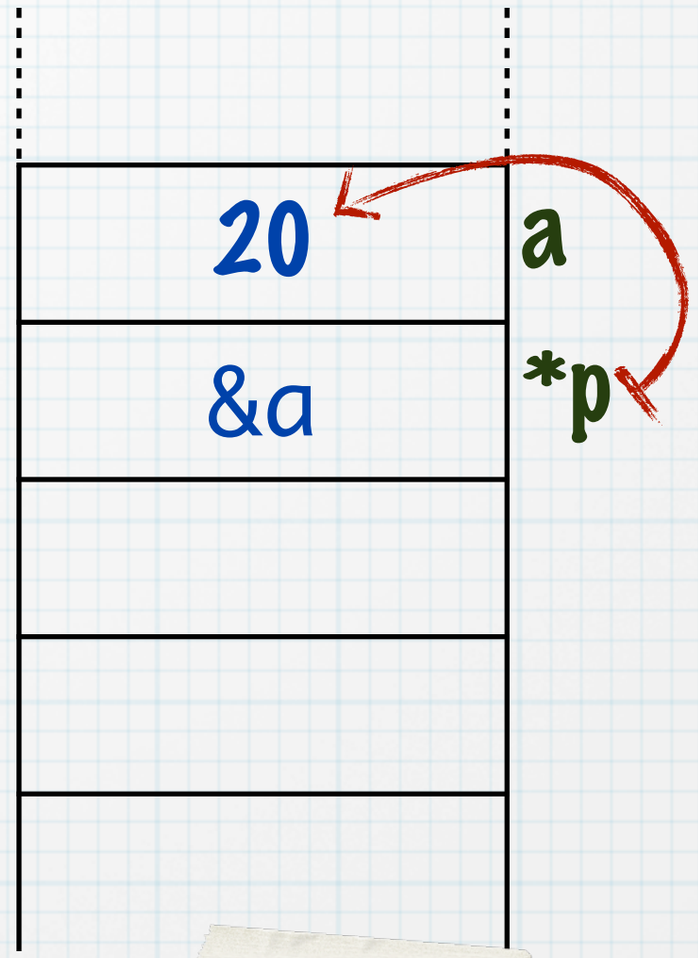


Agenda

* Introdução
* Alocação dinâmica
* Funções
  * malloc
  * free
  * calloc
  * realloc
* Exercícios

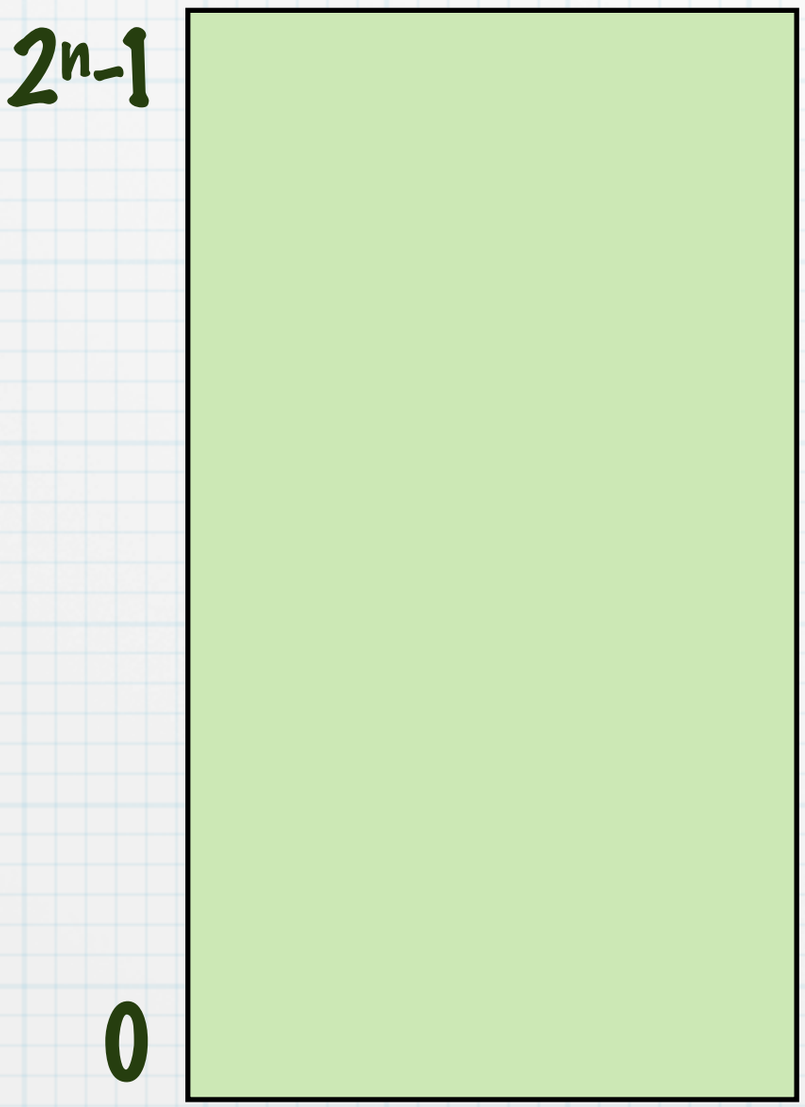
Introdução

* Quando declaramos uma variável, um espaço de memória é reservado
  * Endereço fixo
* Através de ponteiros podemos indicar qual endereço usar
  * E se quisermos usar mais memória?
    Um endereço que não seja de uma variável definida no programa

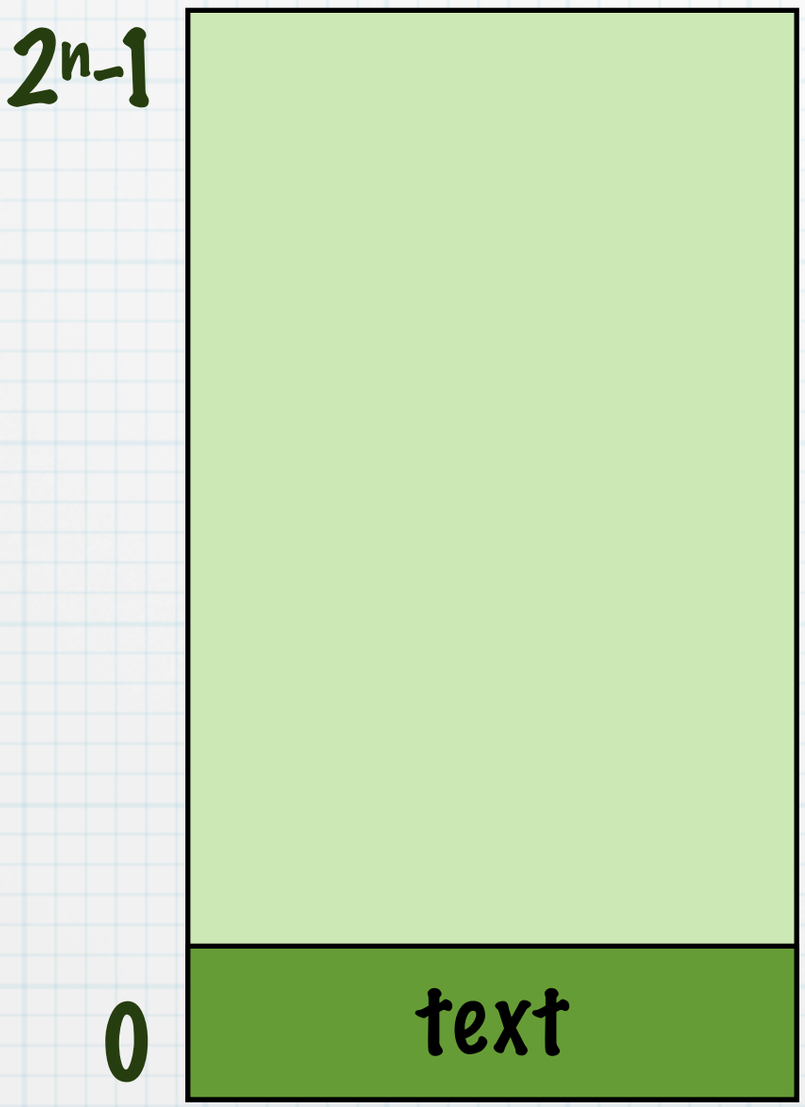


p pode apontar para QUALQUER endereço

Introdução

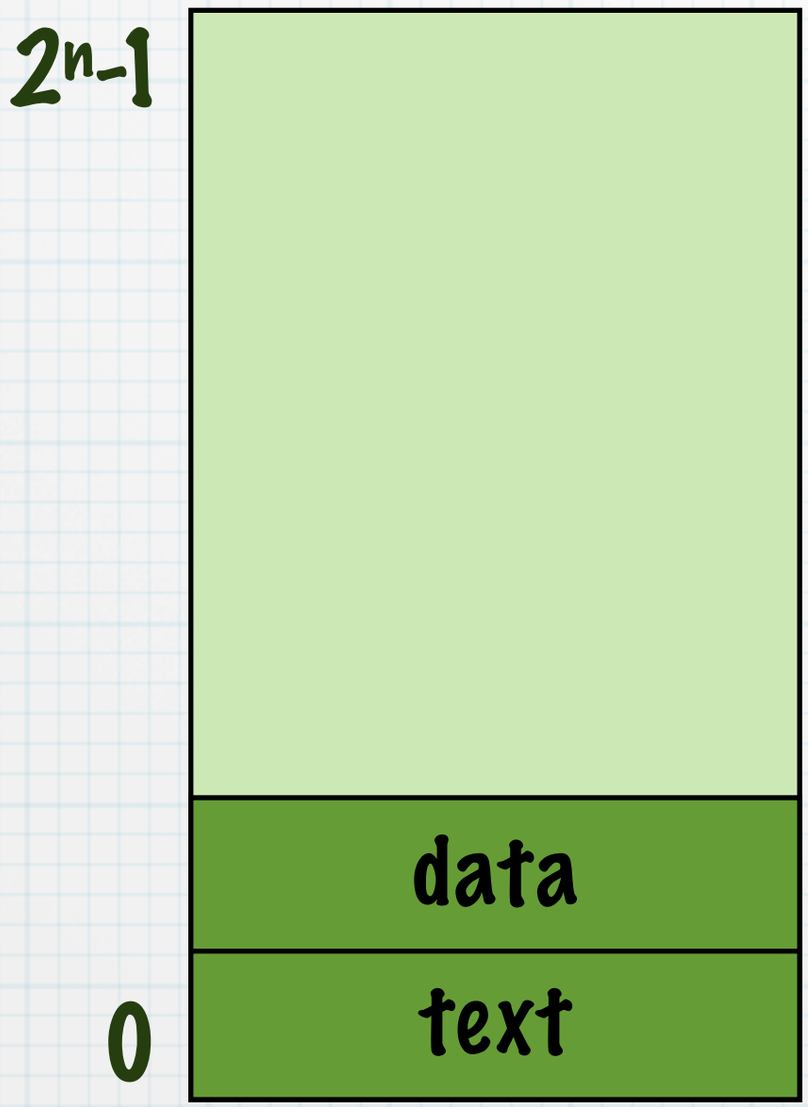


Introdução



Código do programa

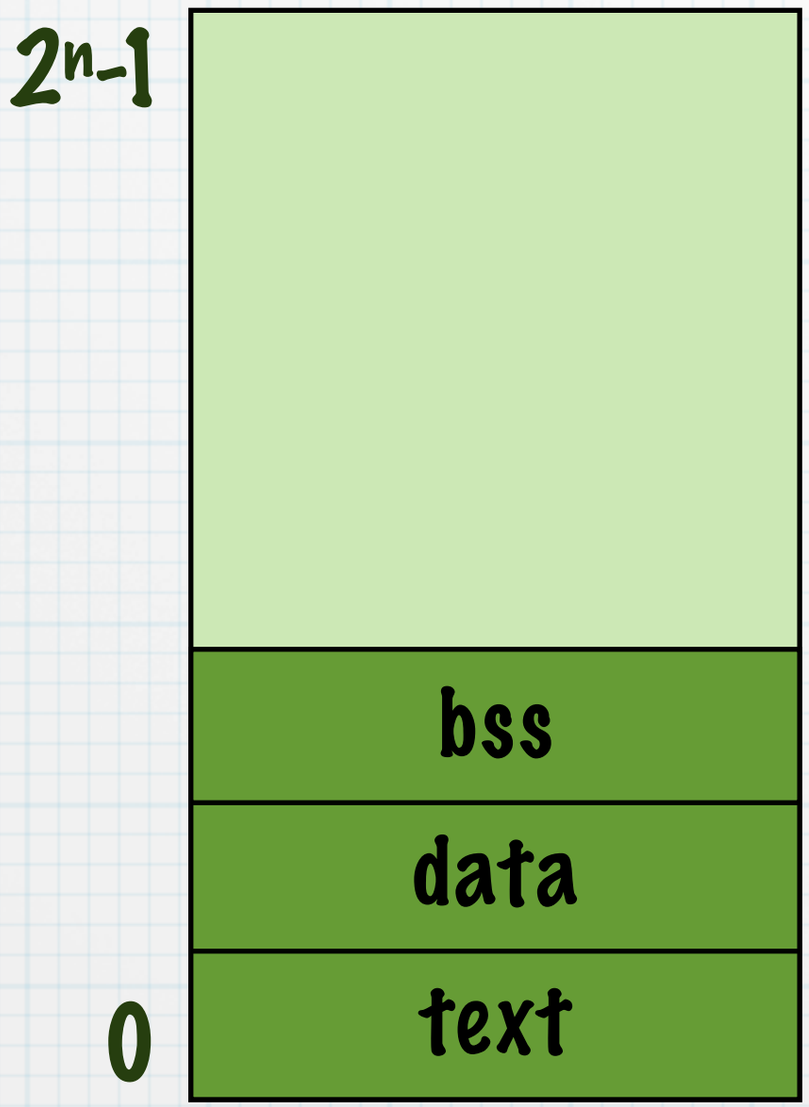
Introdução



Variáveis globais inializadas

Código do programa

Introdução

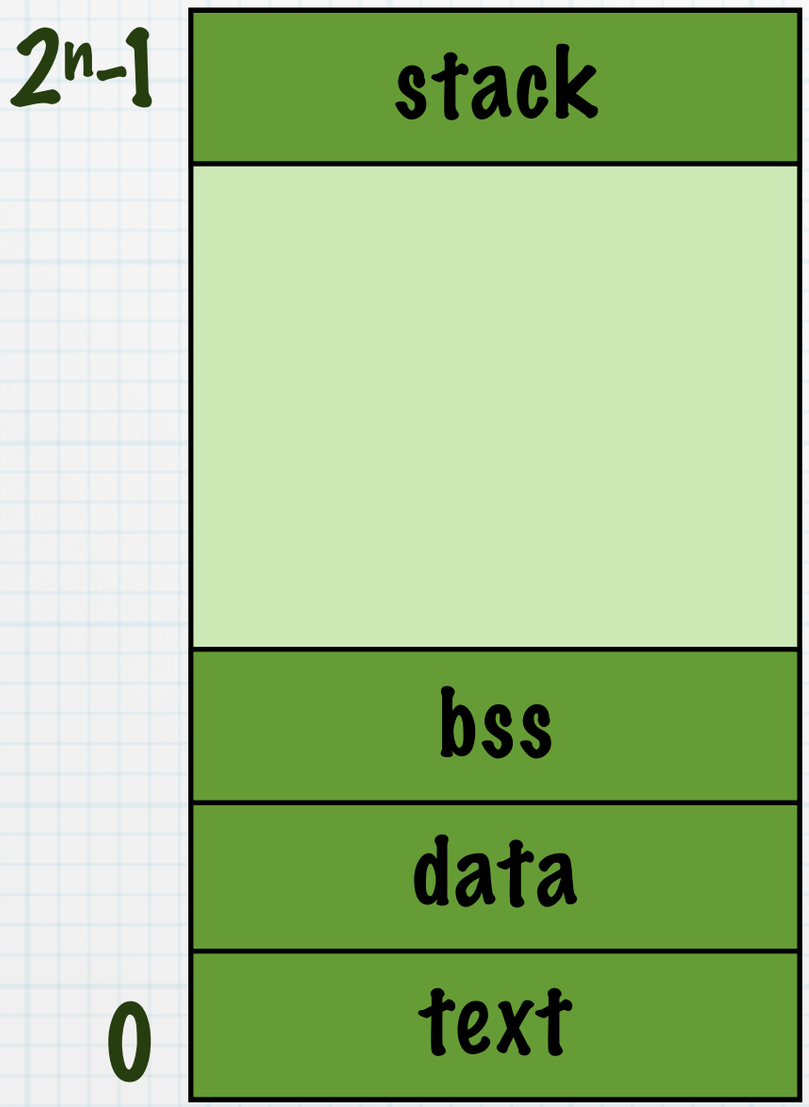


Variáveis globais não inicializadas

Variáveis globais inializadas

Código do programa

Introdução



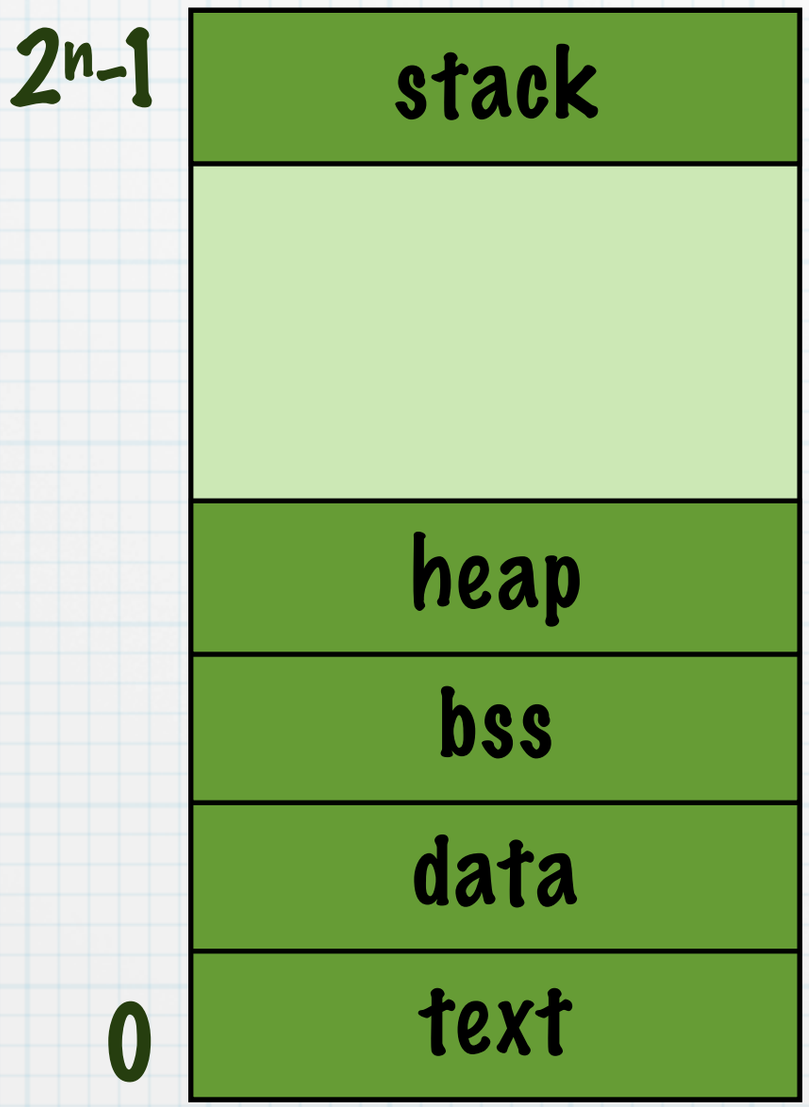
Pilha de dados locais

Variáveis globais não inicializadas

Variáveis globais inializadas

Código do programa

Introdução



Pilha de dados locais

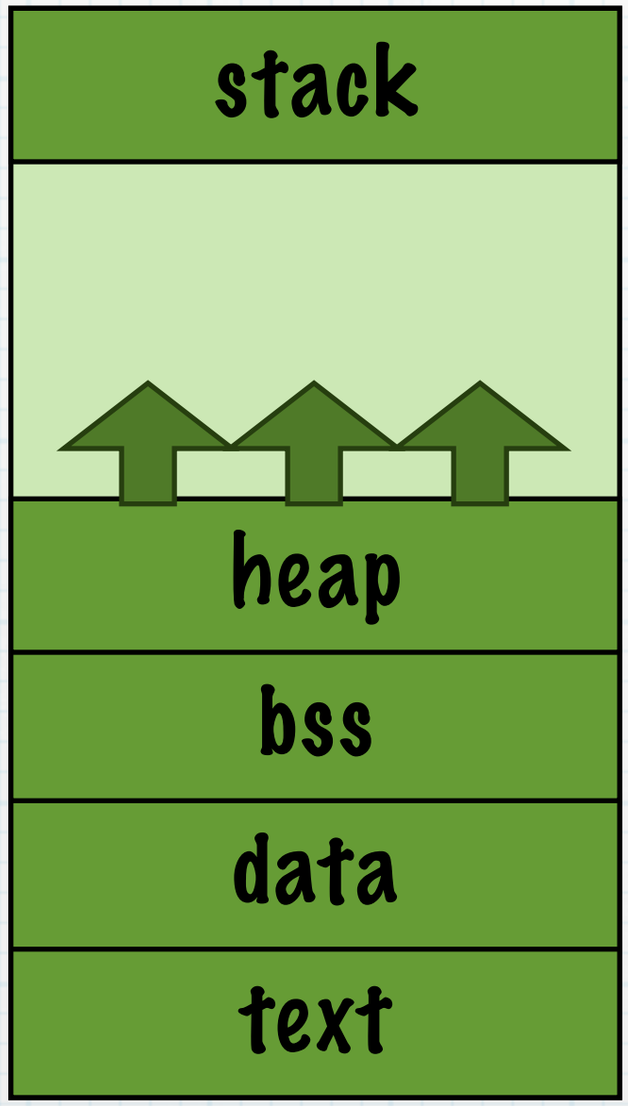
Área alocada dinamicamente

Variáveis globais não inicializadas

Variáveis globais inializadas

Código do programa

Introdução



Pilha de dados locais

Área alocada dinamicamente

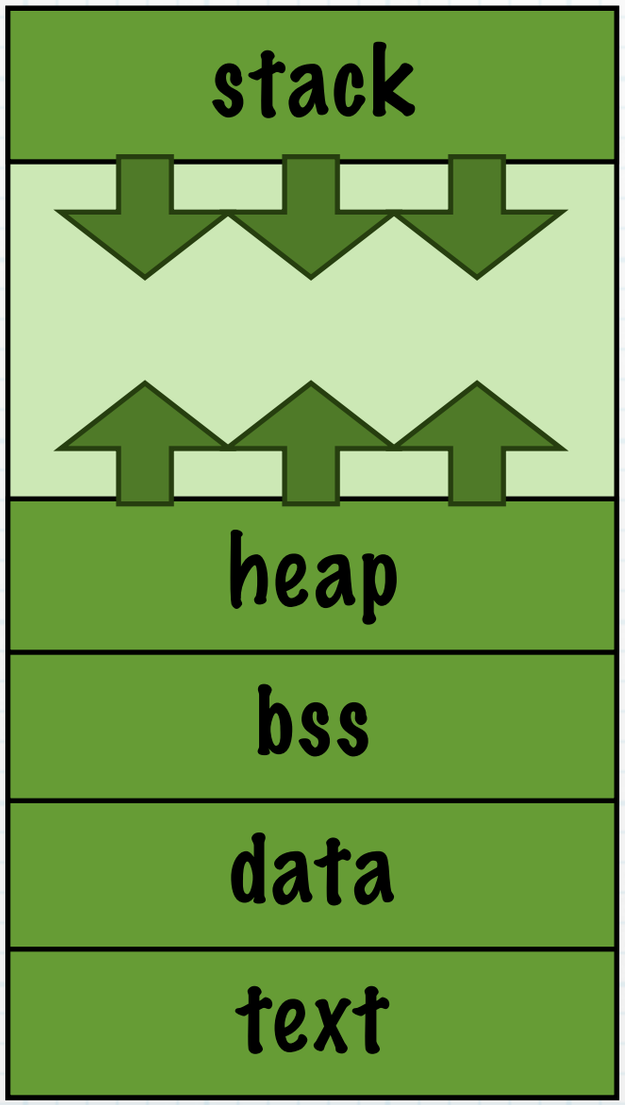
Variáveis globais não inicializadas

Variáveis globais inializadas

Código do programa

Introdução

2^n-1



Pilha de dados locais

Área alocada dinamicamente

Variáveis globais não inicializadas

Variáveis globais inializadas

0

Código do programa

Alocação dinâmica

* Porque?
  * Tamanho "desconhecido" de um array
  * Listas encadeadas (veremos mais a frente)
  * Exemplo:
* Um programa que ordena números inteiros
* Quantidade de números a ordenar desconhecida

5	12	21	29	129	321	821	...	2	123
0	1	2	3	4	5	6		n-1	n

Alocação dinâmica

* Solução 1 (sem alocação dinâmica)
  * Definir capacidade máxima do array

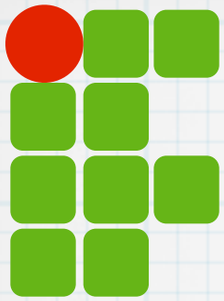
```
#define N 10000000
int numeros[N];
```

* Problemas
  * Limitamos, no programa, a quantidade de números que podemos ordenar
  * **SEMPRE** usaremos a quantidade de memória definida pela capacidade máxima

Funções de gerenciamento

* Gerenciamento de memória
  * Funções de alocação dinâmica
  * O padrão ANSI C define 4 funções para gerenciamento de memória
    * `malloc`: aloca memória
    * `calloc`: aloca e inicia com zeros
    * `realloc`: redimenciona tamanho de memória alocada
    * `free`: libera espaço alocado
* Biblioteca stdlib.h

```
#include <stdlib.h>
```



malloc - memory allocation

* Função malloc:

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

* Retorna um ponteiro para uma área de memória (void *)
  * NULL se não houver memória suficiente
* Devemos passar a quantidade de memória a ser alocada, em **bytes**
* Memória alocada na **heap**

malloc - memory allocation

```
int *numeros;
int quantidade;
numeros = (int*) malloc(sizeof(int) * quantidade);
```

cast para o tipo que realmente queremos

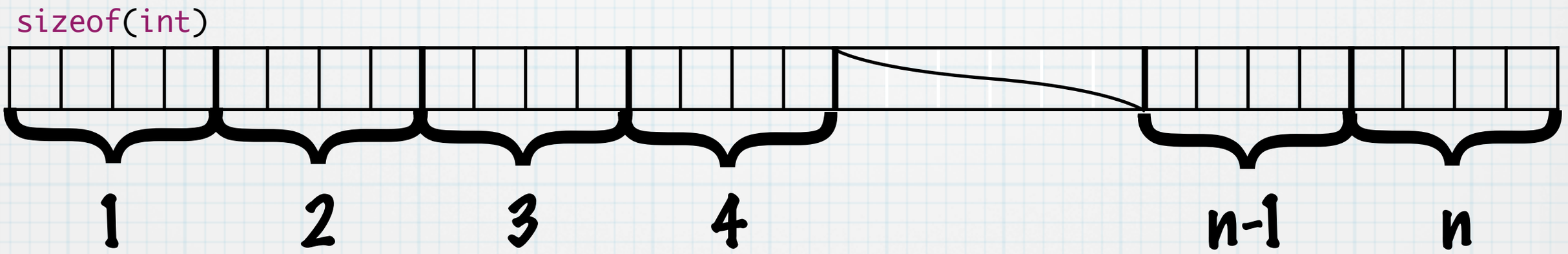
Tamanho de um inteiro, em bytes

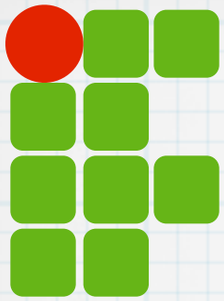
Quantidade a ser alocada

O espaço é alocado sequencialmente

Modelo da memória

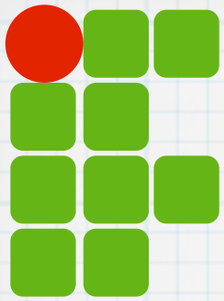
```c
int *numeros;
int quantidade;
numeros = (int*) malloc(sizeof(int) * quantidade);
```





Exemplo

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
  scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
    fprintf(stderr, "ERRO ao alocar memória\n");
    exit(1);
  }
  for (i=0; i<tamanho; i++)
    scanf("%d",&numeros[i]);
  /*...*/

  return 0;
}
```



Exemplo

Biblioteca

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
  scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
    fprintf(stderr, "ERRO ao alocar memória\n");
    exit(1);
  }
  for (i=0; i<tamanho; i++)
    scanf("%d",&numeros[i]);
  /*...*/

  return 0;
}
```

Exemplo

Biblioteca

Alocação

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
  scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
    fprintf(stderr, "ERRO ao alocar memória\n");
    exit(1);
  }
  for (i=0; i<tamanho; i++)
    scanf("%d",&numeros[i]);
  /*...*/

  return 0;
}
```

Exemplo

Biblioteca

Alocação

Verifica se houve alocação

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
  scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
    fprintf(stderr, "ERRO ao alocar memória\n");
    exit(1);
  }
  for (i=0; i<tamanho; i++)
    scanf("%d",&numeros[i]);
  /*...*/

  return 0;
}
```

Exemplo

Biblioteca

Alocação

Verifica se houve alocação

Usa memória alocada

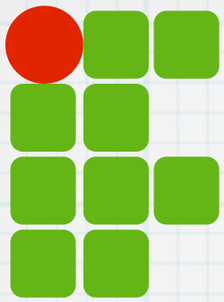
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
  scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
    fprintf(stderr, "ERRO ao alocar memória\n");
    exit(1);
  }
  for (i=0; i<tamanho; i++)
    scanf("%d",&numeros[i]);
  /*...*/

  return 0;
}
```

free - libera memória

* Depois que utilizarmos a memória alocada, devemos liberar
  * Este mesmo espaço fica disponível para novas alocações
* função free

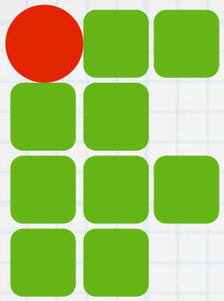
```
void free(void *ptr);
```

Ponteiro para o início de uma área de memória previamente alocada (com malloc ou calloc)



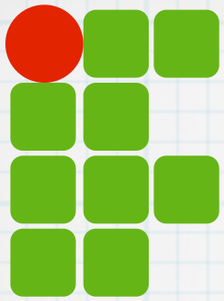
free - libera memória

* IMPORTANTE
  * Liberar após uso
  * Não usar área liberada
  * Não usar área fora da área alocada



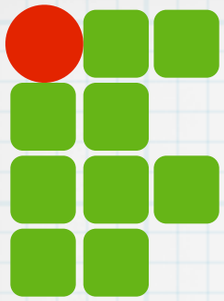
Exemplo

```c
int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
    scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
   fprintf(stderr, "ERRO ao alocar memória\n");
   exit(1);
  }
    for (i=0; i<tamanho; i++)
        scanf("%d",&numeros[i]);
    /*...*/
    free(numeros);
  return 0;
}
```



Exemplo

```c
int main(int argc, char ** argv) {
  int *numeros, tamanho, i;
    scanf("%d",&tamanho);
  numeros = (int*) malloc(sizeof(int) * tamanho);
  if (numeros == NULL) {
   fprintf(stderr, "ERRO ao alocar memória\n");
   exit(1);
  }
     for (i=0; i<tamanho; i++)
         scanf("%d",&numeros[i]);
     /*...*/
     free(numeros);
  return 0;
}
```

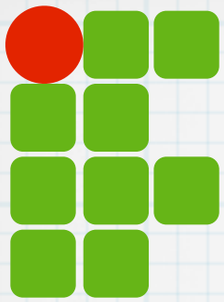


Mais alocação

* Função calloc
  * Mesma função do malloc
  * Coloca zeros (0) em toda a área alocada
  * Após seu uso também devemos liberar área

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

* Dois parâmeros
  * Quanto elementos deseja alocar
  * O tamanho, em bytes, de cada elemento



Realocação

* Redimensiona uma área de memória para um novo tamanho

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

* O retorno de realloc é um ponteiro para a área redimensionada.
* Nova área pode começar em endereço diferente do original

Retorno de função

Ponteiro para uma área de memória

Alocaçao da área de memória

```
int *impares(int *a,int tamanho, int *qtdImpares){
    int i,j,qtdI;
    int *impares;
    qtdI = 0;
    for (i = 0 ; i < tamanho ; i++)
        if (a[i]%2==1)
            qtdI++;
    impares=(int*)malloc(sizeof(int)*qtdI);
    j=0;
    for (i = 0 ; i < tamanho ; i++)
        if (a[i]%2==1)
            impares[j++]=a[i];
    *qtdImpares = qtdI;
    return impares;
}
```

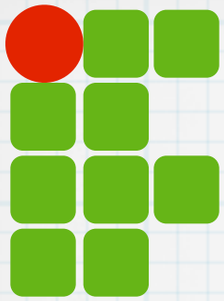
Retorno de função

Ponteiro para uma área de memória

Alocaçao da área de memória

```c
int *impares(int *a,int tamanho, int *qtdImpares){
    int i,j,qtdI;
    int *impares;
    qtdI = 0;
    for (i = 0 ; i < tamanho ; i++)
        if (a[i]%2==1)
            qtdI++;
    impares=(int*)malloc(sizeof(int)*qtdI);
    j=0;
    for (i = 0 ; i < tamanho ; i++)
        if (a[i]%2==1)
            impares[j++]=a[i];
    *qtdImpares = qtdI;
    return impares;
}
```

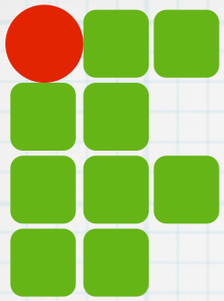
```c
int main(int argc, char ** argv) {
   int *numeros,*imp,qtdI, tamanho, i;
    scanf("%d",&tamanho);
    numeros = (int*) malloc(sizeof(int) * tamanho);
    for (i=0; i<tamanho; i++)
        scanf("%d",&numeros[i]);
    imp = impares(numeros,tamanho,&qtdI);
    free(numeros);
    free(imp);
   return 0;
}
```



Exemplo

* Otimizar área usada por uma string

```
char * removeAreaNaoUsada(char *s){
    char *nova,*origem,*destino;
    nova=(char*)malloc(sizeof(char)*(strlen(s)+1));
    origem=s;
    destino=nova;
    while (*origem!='\0') {
        *destino=*origem;
        destino++;
        origem++;
    }
    *destino=*origem;
    return nova;
}
```



Exemplo

* Otimizar área usada por uma string

```c
char * removeAreaNaoUsada(char *s){
    char *nova,*origem,*destino;
    nova=(char*)malloc(sizeof(char)*(strlen(s)+1));
    origem=s;
    destino=nova;
    while (*origem!='\0') {
        *destino=*origem;
        destino++;
        origem++;
    }
    *destino=*origem;
    return nova;
}
```

```c
int main(int argc, char ** argv) {
    char *s,*aux;
    /*...*/
    aux=s;
    s=removeAreaNaoUsada(s);
    free(aux);
    /*...*/
  return 0;
}
```

Conclusão

* Alocação dinâmica de memória é uma ferramenta poderosa para desenvolver programas em ANSI C
* Deve ser usada com cuidado
  * Não perder referências a áreas de memórias alocadas
  * Liberar memória que não é mais usada

Dúvidas?