Programação I

Alocação dinâmica de memoria

Estrela Ferreira Cruz

1

Objetivos da aula

Objetivos da aula:

Alocação dinâmica de memória:

- Apresentação do conceito;
- Principais funções de alocação de memória:
  - malloc()
  - calloc()
  - realloc();
- Libertação de memória: função free();
- Apresentação de exemplos práticos.

2

2

Alocação dinâmica de memória

Memória do computador

- Quando um programa é executado o sistema operativo coloca o código do programa e as instruções em linguagem máquina na memória do computador.
- Além das instruções, o sistema operativo também reserva memória para as variáveis que o programa utiliza.
- A memória do computador é um recurso que é partilhado pelos vários programas em execução no computador.
- Quem se encarrega de fazer a gestão da memória é o sistema operativo, que reserva uma parte da memória para armazenar programas e outra parte para armazenar dados.
- O próprio sistema operativo é um conjunto de programas e ele próprio também está a ocupar parte da memória do computador.

.

3

3

Alocação dinâmica de memória

Memória do computador

- Quando a memória do computador começa a ficar cheia, o sistema operativo pode utilizar parte do disco para servir de memória RAM. A desvantagem disso, é que o tempo de acesso ao disco é muito maior do que o tempo de acesso à memória RAM. Essa é uma das razões que leva a que os computadores fiquem lentos.
- **Cada posição de memória corresponde a um byte**.
- Cada variável, dependendo do seu tipo, ocupa um ou vários bytes, ou seja, várias posições de memória consecutivas:
  - Uma variável do tipo char geralmente ocupa 1 byte.
  - Uma variável do tipo int geralmente ocupa 4 bytes.
  - etc.

4

4

Alocação dinâmica de memória

Memória do computador

Se criarmos um programa com a seguinte instrução:

```
int x = 9;
```

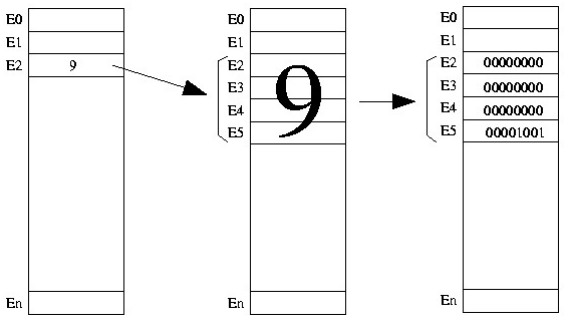
Onde fica o valor de x guardado em memoria?

- o x pode ficar armazenada numa posição de memória diferente de cada vez que o programa é executado.
- o número 9 (inteiro) não vai estar representado numa única posição de memória mas sim em 4 posições de memória consecutivas (exemplo: E2, E3, E4 e E5 (ver pag. seguinte)).
- O que é que realmente fica guardado em memória?
- Aquilo que está nessas 4 posições de memória é a representação em binário do número 9 (ver exemplo página seguinte).

5

5

Alocação dinâmica de memória



Memória do computador

- Apesar da variável x estar localizada nos endereços E2, E3, E4 e E5, diz-se que o endereço da variável x (&x) é E2. Isto é, o endereço de uma variável é o endereço do primeiro byte que esta ocupa.

.

6

6

Alocação dinâmica de memória

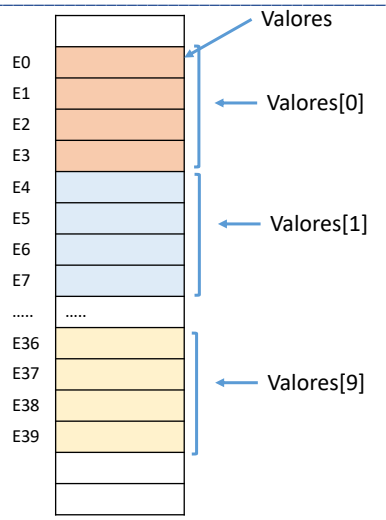
ARRAYS (Vetores)

- Tal como já vimos anteriormente, quando se declara um array, o compilador reserva um bloco consecutivo de memória que permita guardar todos os valores do array.
- O endereço do array é o endereço do primeiro byte que o array ocupa.
- O nome do array é o endereço da primeira posição do array

7

7

Alocação dinâmica de memória



Exemplo:

Suponha o array de inteiros com 10 elementos:

int valores[10];

Partindo do principio que um inteiro ocupa 4 bytes, o complicador vai reservar 40 bytes consecutivos em memória.

Por ex. do endereço E0 ao endereço E39.

Assim, teremos:

valores[0] – vai ocupar os endereços E0, E1, E2 e E3
valores[1] – vai ocupar os endereços E4, E5, E6 e E7
..
valores[9] – vai ocupar os endereços E36, E37, E38 e E39.
.

8

8

Alocação dinâmica de memória

Exemplo: int valores[10];

Sabendo que, quando nos referimos ao nome do array, estamo-nos a referir ao endereço de memória da primeira posição do array, *valores* é equivalente a *&valores[0].*

Assim, podemos ter:

```
int valores[10];
int *p=NULL;
p = valores;      // equivalente a p=&valores[0];
*p = 4;           // equivalente a dizer valores[0] = 4;
```

Se o endereço do array *valores* for 3820, ao fazermos *p = valores (ou p=&valores[0]*), p tomará o valor de 3820.

9

9

Alocação dinâmica de memória

Quando e durante quanto tempo as variáveis ocupam memória?

--

10

10

Alocação dinâmica de memória

Variáveis globais

- A declaração de **variáveis globais** indica ao compilador que é necessário atribuir em memória o espaço necessário para essas variáveis. Esta atribuição é definida na compilação e efetuada sempre que o programa é executado.
- Normalmente o espaço correspondente às variáveis globais é libertado apenas no final da execução do programa.

Variáveis locais

- Quanto às **variáveis locais** (definidas dentro de funções) o processo é idêntico. Na compilação é definido o espaço necessário para as variáveis locais às funções, no entanto a atribuição de memória é feita quando, e sempre que, a função é executada (invocada).
- Normalmente o espaço correspondente às variáveis locais é libertado no final da execução da função.

11

11

Alocação dinâmica de memória

- Existe um mecanismo de **alocação (reserva) e libertação dinâmica de memória**, que pode ser usado diretamente, quando necessário.
- A alocação dinâmica de memória, em C, é efetuada utilizando a função **malloc() (abreviatura de memory allocation)** que, tem como argumento o espaço (em nº de bytes) que se pretende reservar e devolve um apontador para a primeira posição desse espaço atribuído dinamicamente.

```
void *malloc(int number_of_bytes);
```

- A função malloc() pertence à **biblioteca stdlib.h**.
- Executa um pedido ao sistema operativo para reservar uma porção de memória (com tamanho igual ao número de bytes cujo valor é recebido como parâmetro na função).

12

12

Alocação dinâmica de memória

- A função **malloc()** retorna um apontador genérico **(void *)** para o início da porção de memória reservada que deverá conter espaço suficiente para armazenar **<number_of_bytes>** bytes.
- Se não for possível alocar essa quantidade de memória a função retorna NULL.

Exemplo: Neste exemplo temos um programa que reserva espaço para 40 bytes:

```
int main() {
char *ptr=NULL;
ptr=(char *)malloc(40);
if (ptr==NULL) {
        printf("Erro ao reservar memoria\n"); return -1;
}
strcpy(ptr,"Avenida do atlântico - Viana do Castelo");
printf ("\n%s\n", ptr);
free(ptr);
return 0;
}
```

13

13

Alocação dinâmica de memória

Como saber qual o espaço necessário para uma determinada variável?

→ A função sizeof(), como é sabido, devolve o espaço (quantidade de bytes) necessário a uma variável de um determinado tipo.

14

14

Alocação dinâmica de memória

- malloc(sizeof(int)) devolve um apontador para o espaço de memória com o tamanho necessário para armazenar uma variável do tipo inteiro.
- Poderíamos usar as seguintes instruções para reservar espaço para 100 inteiros

```
int *ip=NULL;
ip = (int *) malloc(100*sizeof(int));
```

- Devolve um apontador para o espaço correspondente à declaração de 100 variáveis do tipo inteiro, armazenadas consecutivamente, ou seja, um vetor de inteiros com 100 posições. Esta declaração é equivalente à declaração int ip[100];
- Uma forma de alocar memória para um array de n elementos de um tipo TD é a seguinte:

```
TD *a=NULL;
a = (TD *) malloc( n * sizeof(TD) );
```

15

15

Alocação dinâmica de memória

NOTAS importantes:

- O tipo do resultado da função malloc() deverá ser corretamente adaptado ao tipo para o qual se está a atribuir memória. Para tal usa-se o "cast". O cast para o tipo de apontador correto é muito importante para assegurar que a aritmética com os apontadores é efetuada de modo correto.
- O uso do operador sizeof( ) também é indispensável, mesmo que se conheça o tamanho do tipo de dados, para tornar o código independente da máquina em que é compilado. O seu uso torna o código facilmente portável.

16

16

Alocação dinâmica de memória

Notas importantes:

- A linguagem C não testa a validade de acessos fora do bloco de memória previamente declarado ou alocado.
- **É da responsabilidade do programador assegurar que esses acessos não são feitos**.
- A memória não é infinita, por isso **sempre que reservamos memória para um programa correr temos que a libertar**.

Como podemos libertar a memória reservada dinamicamente?

17

17

Alocação dinâmica de memória

- A memória não é infinita, por isso, sempre que reservamos memória temos que a libertar quando já não está a ser usada.
- Para um bom funcionamento do sistema, é obrigatório libertar o espaço de memória previamente alocado pela função malloc(). Esse é o trabalho da função free().

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

- A função free() liberta a porção de memória alocada.
- O comando free(ptr) avisa o sistema de que o bloco de memória apontado por ptr está livre. Essa porção de memória ficará disponível para novas alocações.
- A função free() pertence à biblioteca stdlib.h.

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

18

18

Alocação dinâmica de memória

Exemplo: O programa seguinte calcula a soma de dois números, usando a função malloc() para reservar memória e a função free() para a libertar.

```
int main(){
int *ptr=NULL, x=0;
ptr = (int *) malloc (sizeof (int));
if (ptr == NULL) {   // verificar se tem memória suficiente
            printf ("Ocorreu um erro ao reservar memória!\n");
            return (-1);
}
printf ("Introduza dois valores inteiros:\n");
scanf ("%d %d", ptr, &x);
(*ptr) += x;
printf ("Resultado = %d\n", *ptr);
free(ptr);    // libertar a memória
return 0;
}
```

19

19

Alocação dinâm

Exemplo 2: Fazer um programa para receber do teclado uma lista de n números, em que n seria um número introduzido pelo utilizador. No final deverá enviar para o ecrã os números superiores à média dos números introduzidos.

```
#define N 10000
main() {
   int a[N], n,i,total=0, media=0;
   printf("Quantos números quer introduzir? ");
   scanf("%d", &n);
    if( n > N ) {
        printf("O numero deve ser <= %d\n", N);
    }
   else {
         printf("Introduza os numeros\n");
         for (i=0;i<n;i++) {
               scanf("%d", &a[i]);
               total=total+a[i];
         }
         media=total/n;
         for (i=0;i<n;i++) {
              if (a[i] > media)
               printf("%d\n",a[i]);
          }
    }
}
```

20

Alocação dinâmica de memória

- Este programa tem a limitação de funcionar apenas para n < N (neste caso 10000).
- O que tradicionalmente se faz é definir N suficientemente grande e depois usar apenas parte do array.
- No entanto essa solução tem a desvantagem de desperdiçar memória. Estamos a reservar um array de 10000 inteiros e provavelmente o número n introduzido pelo utilizador vai ser apenas 10 ou 20!
- A alternativa a esta solução é requisitar a memória ao computador durante a própria execução do programa. A ideia é perguntar ao utilizador a dimensão do array. Se o utilizador introduzir 10, pedimos ao computador para nos dar um bloco de memória que permita guardar 10 inteiros. Deste modo, o array ocupa apenas o espaço que é estritamente necessário.

21

21

Alocação dinâmica

Exemplo 2: resolução do problema anterior recorrendo ao uso de alocação dinâmica de memória. Assim, o array ocupa o espaço suficiente e estritamente necessário para armazenar os valores.

```
int main() {
   int *a=NULL, n=0, total=0, media=0;
   printf("Quantos números quer introduzir? ");
   scanf("%d", &n);
   a = (int *) malloc( n * sizeof(int));
   if ( a == NULL ) {
       printf("Erro ao alocar memória\n");
       return 0;
    }
    printf("Introduza os numeros\n");
    for (i=0;i<n;i++) {
            scanf("%d", &a[i]);
            total=total+a[i];
     }
     media=total/n;
     for (i=0;i<n;i++) {
             if (a[i] > media) {
                  printf("%d\n",a[i]);
             }
      }
   free( a );  //liberta a memória
   return 0;
}
```

22

Alocação dinâmica de memória

A função calloc() é especifica para alocação de memória dinâmica para arrays.

```
void *calloc(int num_celulas, int num_bytes);
```

- num_células - é o numero de células a alocar (tamanho do array);
- num_bytes é o numero de bytes ocupado por cada célula;

- A função retorna um apontador genérico (void *) para o início da memória reservada.
- Se não for possível alocar essa quantidade de memória a função retorna o NULL.

void *calloc(int num_celulas, int num_bytes);

23

Alocação dinâmica de memória

Particularidades da função calloc()

- Todas as células de memória alocada são devidamente inicializadas;
- Quando se utiliza o endereço devolvido pela função calloc() como argumento da função free(), toda a memória das células alocada é libertada.

Exemplo: Alocar memória para um array de inteiros com 100 elementos.

```
int *api=NULL;
api=(int*) calloc (100, sizeof(int));
```

Em vez de:

```
int *api=NULL;
api=(int*) malloc (100 * sizeof(int));
```

24

Alocação dinâmica de memória

Exemplo da função calloc() : Neste exemplo, o programa recebe do teclado uma string e copia essa string para outra que ocupa apenas o espaço em memória estritamente necessário.

```
int main(void){
char frase[500], *dest=NULL;
printf("\nIntroduza uma frase -> ");
gets(frase);
dest = (char *) calloc(strlen(frase)+1, sizeof(char));
if (dest==NULL){
      printf("Out of memory\n");
      return -1;
}
strcpy(dest, frase);
printf("\nCópia da string -> %s", dest);
free(dest);
return 0;
}
```

25

Alocação dinâmica de memória

Função realloc(): pode ser usada para realocar memória.

A sua sintaxe é a seguinte:

```
void *realloc(void *p, int num_bytes);
```

- A função realloc() pode ser usada para alterar o tamanho da memória anteriormente alocado em *p (1º parâmetro) para o tamanho especificado por um novo valor (2º parâmetro).
- Na função realloc(), se o bloco previamente alocado puder ser estendido (aumentar o tamanho) para a nova dimensão, a memória adicional é reservada.
- Se não existir espaço suficiente para prolongar o bloco, é criado um novo bloco com a totalidade dos bytes necessários e os dados são copiados para a nova localização. A função retorna o novo endereço de memória.
- Se não for possível alocar o novo espaço, a função devolve o valor NULL.

```
void *realloc(void *p, int num_bytes);
```

26

26

Alocação dinâmica de memória

Exemplo de uso da função realloc().

```
void main() {
char *cidade=NULL;
cidade = (char *) malloc(6*sizeof(char));
if (cidade==NULL){
       printf("Out of memory\n");
       return -1;
}
strcpy(cidade, "Viana");           //armazena string=Viana
printf("nome da cidade=%s\n", cidade);
cidade=(char*)realloc(cidade,20*sizeof(char));
if (cidade==NULL){
   printf("Erro ao realocar\n");
}
else {
   strcat(cidade, " do Castelo"); //armazena string=Viana do Castelo
   printf("Nome final=%s\n", cidade);
}
free(cidade);
}
```

27

Alocação dinâmica de memória

Resumo: biblioteca stdlib.h.

void *malloc(int num_bytes);

void *calloc(int num_celulas, int num_bytes);

void *realloc(void *p, int num_bytes);

void free (void *ptr);

28

28

Bibliografia

- Programação Avançada Usando C, António Manuel Adrego da Rocha, ISBN: 978-978-722-546-0.
- Schildt, Herbert: C the complete Reference, McGraw-Hill, 1998.
- Algoritmia e Estruturas de Dados, José Braga de Vasconcelos, João Vidal de Carvalho, ISBN: 989-615-012-5.
- Linguagem C, Luís Manuel Dias Damas, ISBN: 972-722-156-4.
- Elementos de Programação com C - Pedro João Valente D. Guerreiro, 3ª edição, ISBN: 972-722-510-1.
- Introdução à Programação Usando C, António Manuel Adrego da Rocha, ISBN: 972-722-524-1.

29