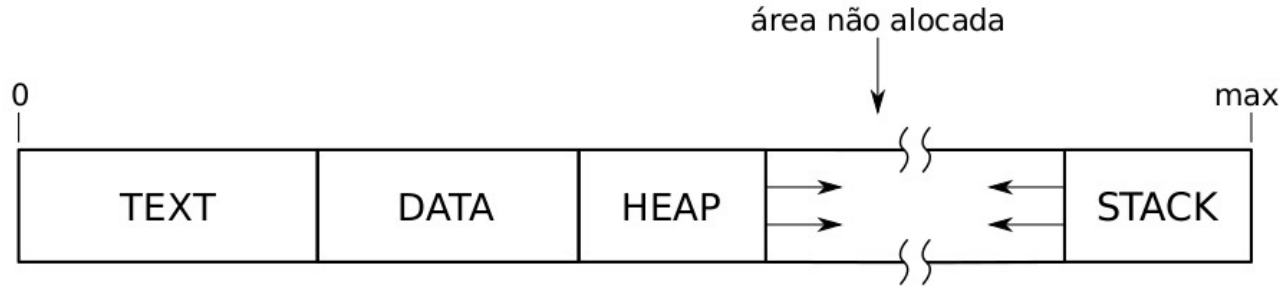
Alocação de memória

O espaço de endereços de um processo em execução é dividido em várias segmentos lógicos. Os mais importantes são:

- *Text*: contém o código do programa e suas constantes. Este segmento é alocado durante a criação do processo ("exec") e permanece do mesmo tamanho durante toda a vida do processo.
- *Data*: este segmento é a memória de trabalho do processo, aonde ficam alocadas as variáveis globais e estáticas. Tem tamanho fixo ao longo da execução do processo.
- *Stack*: contém a pilha de execução, onde são armazenadas os parâmetros, endereços de retorno e variáveis locais de funções. Pode variar de tamanho durante a execução do processo.
- *Heap*: contém blocos de memória alocadas dinamicamente, a pedido do processo, durante sua execução. Varia de tamanho durante a vida do processo.



Um programa em C suporta três tipos de alocação de memória:

- A **alocação estática** ocorre quando são declaradas variáveis globais ou estáticas; geralmente alocadas em *Data*.
- A **alocação automática** ocorre quando são declaradas variáveis locais e parâmetros de funções. O espaço para a alocação dessas variáveis é reservado quando a função é invocada, e liberado quando a função termina. Geralmente é usada a pilha (*stack*).
- A **alocação dinâmica**, quando o processo requisita explicitamente um bloco de memória para armazenar dados; o controle das áreas alocadas dinamicamente é manual ou semi-automático: o programador é responsável por liberar as áreas alocadas dinamicamente. A alocação dinâmica geralmente usa a área de *heap*.

Alocação estática

A alocação estática ocorre com variáveis globais (alocadas fora de funções) ou quando variáveis locais (internas a uma função) são alocadas usando o modificador "static". Uma variável alocada estaticamente mantém seu valor durante toda a vida do programa, exceto quando explicitamente modificada.

Exemplo:

```
#include <stdio.h>

int a = 0 ;  // variável global, aloc. estática

void incrementa(void)
{
          int b = 0 ; // variável local, aloc. automática
```

```
   static int c = 0 ; // variável local, aloc. estática

   printf ("a: %d, b: %d, c: %d\n", a, b, c) ;
   a++ ;
   b++ ;
   c++ ;
}

int main(void)
{
   int i ;

   for (i = 0; i < 5; i++)
      incrementa() ;

   return 0 ;
}
```

A execução desse código gera a seguinte saída:

```
  a: 0, b: 0, c: 0
  a: 1, b: 0, c: 1
  a: 2, b: 0, c: 2
  a: 3, b: 0, c: 3
  a: 4, b: 0, c: 4
```

As variáveis com alocação estática ("a" e "c") são alocadas e inicializadas uma única vez, portanto seus valores se preservam entre chamadas consecutivas da função "incrementa". Por outro lado, a variável com alocação automática "b" é alocada e descartada a cada execução da função, portanto seu valor não é preservado.

Alocação automática

Por *default*, as variáveis definidas dentro de uma função (variáveis locais e parâmetros) são alocadas de forma automática na pilha de execução do programa (*stack*) a cada chamada da função, sendo descartadas quando a função encerra. Isso é o que ocorre com a variável "b" do exemplo anterior.

A pilha de execução do programa normalmente é pequena (8 MB ou menos). Por isso, a tentativa de alocar variáveis locais muito grandes pode resultar em erro de compilação ou de execução (SIGSEGV - *Segmentation Fault*). Para tais situações devem ser usadas variáveis globais (estáticas) ou variáveis dinâmicas.

Se a função for chamada recursivamente, as variáveis locais e parâmetros serão novamente alocados na pilha, em áreas distintas para cada nível de recursão. Isso permite preservar os valores anteriores dos mesmos no retorno dos níveis de recursão.

O exemplo a seguir permite observar a existência de múltiplas instâncias de variáveis locais (no caso, o parâmetro "n") em chamadas recursivas:

```
#include <stdio.h>

long int fatorial (int n)
{
   long int parcial ;

   printf ("antes:  n: %d\n", n) ;

   if (n < 2)
      parcial = 1 ;
   else
      parcial = n * fatorial(n - 1) ;
```

```
   printf ("depois: n: %d, parcial: %ld\n", n, parcial) ;

   return(parcial) ;
}

int main (void)
{
   printf ("Fatorial (6) = %ld\n", fatorial (6)) ;
   return 0 ;
}
```

A execução gera o seguinte resultado:

```
  antes:  n: 6
  antes:  n: 5
  antes:  n: 4
  antes:  n: 3
  antes:  n: 2
  antes:  n: 1
  depois: n: 1, parcial: 1
  depois: n: 2, parcial: 2
  depois: n: 3, parcial: 6
  depois: n: 4, parcial: 24
  depois: n: 5, parcial: 120
  depois: n: 6, parcial: 720
  Fatorial (6) = 720
```

O padrão C99 permite a alocação automática de vetores de tamanho variável, definidos em tempo de execução. O exemplo a seguir ilustra esse conceito:

```
int my_function (int n)
{
   char name[n] ;  // aloca string com tamanho "n"

   ...
}
```

Alocação dinâmica

Na alocação dinâmica, o programa solicita explicitamente áreas de memória ao sistema operacional, as utiliza e depois as libera quando não forem mais necessárias, ou quando o programa encerrar. As requisições de memória dinâmica são geralmente alocadas na área de memória denominada *heap*.

Por default, o compilador gcc gera código que pode alocar memória até 4GB, mesmo em máquinas de 64 bits com mais memória disponível. Para gerar código executável com capacidade para alocar memória dinamicamente além desse limite, devem ser usados flags de compilação específicos, como "-mcmodel=medium" ou "-mcmodel=large".

Alocação simples

A memória pode ser alocada dinamicamente através da chamada "malloc":

```
#include <stdlib.h>
void * malloc (size_t size)
```

Esta função aloca um novo bloco com "size" bytes de tamanho e retorna um ponteiro para o início do bloco (ou 0 em caso de erro). O conteúdo desse novo bloco é **indefinido** e pode (seguramente) conter "lixo".

Exemplo de uso:

```
struct mystruct *ptr;
...
ptr = malloc( sizeof(struct mystruct) );
if (ptr == 0) abort();       // caso a alocação não tenha ocorrido
```

Liberação

A chamada "free" deve ser invocada para liberar uma área de memória previamente alocada dinamicamente:

```
#include <stdlib.h>
void free (void *ptr)
```

Esta função libera um bloco de memória previamente alocado, apontado por "ptr". **Atenção:** o ponteiro "ptr" continua apontando para o bloco liberado e por isso é aconselhável mudar seu valor para "NULL" após a liberação:

```
ptr = malloc (1024) ;
...
free (ptr) ;
ptr = NULL ;                 // não é obrigatório, mas aconselhável
```

A memória alocada por um programa é automaticamente liberada quando sua execução encerra. Por isso, o uso da chamada "free()" não é obrigatório no final do programa. Contudo, é recomendado utilizá-lo sempre, para desenvolver o hábito salutar de *sempre liberar um bloco alocado*.

Redimensionamento de área alocada

```
#include <stdlib.h>
void * realloc (void *ptr, size_t newsize)
```

Esta função redimensiona o bloco previamente alocado, apontado por "ptr", para o novo tamanho "newsize". "realoc" retorna o novo endereço do bloco, que pode ser diferente do anterior, caso tenha sido necessário mudá-lo de lugar (o conteúdo original do bloco é preservado nesse caso ou em caso de erro).

Alocação de vetor

A função "calloc()" aloca um bloco de memória de tamanho suficiente para conter um vetor com "count" elementos de tamanho "eltSize" cada um. O conteúdo do bloco alocado é **preenchido por zeros**.

```
#include <stdlib.h>
void * calloc (size_t count, size_t eltSize)
```

Exemplo:

```
float *v ;
int i ;

v = calloc( 1000, sizeof(float) ) ;  // aloca 1.000 floats

for (i=0; i < 1000; i++)             // inicializa o vetor
   v[i] = 1.0 / (i + 1) ;
```

Alocação semiautomática

A função "alloca()" provê um mecanismo de alocação dinâmica semi-automática. O bloco é alocado manualmente, mas será liberado automaticamente ao encerrar a função na qual ele foi alocado. O valor de

retorno da chamada é o endereço de um bloco de tamanho "size" bytes, alocado **na pilha** da função atual, como se fosse uma variável local.

```
#include <stdlib.h>
void * alloca (size_t size)
```

Alinhamento de memória

Os blocos alocados pelas funções descritas acima geralmente iniciam em um endereço múltiplo de 8 bytes em plataformas de 32 bits. Caso seja necessário garantir a alocação alinhada, obtendo blocos iniciando em múltiplos de 8, 16, 32, 64 bytes, etc, deve-se usar a função "memallign()".

```
#include <malloc.h>
void * memalign (size_t boundary, size_t size)
```

"memalign()" aloca um bloco de tamanho "size" cujo endereço inicial é um múltiplo de "boundary" (que deve ser 2^n). Retorna o endereço do bloco alocado, que pode ser liberado mais tarde através da função "free()". Ver a função "posix_memalign" (man 3 posix_memalign)

Exercícios

- Escreva um programa que aloque dinamicamente um vetor "v" e o preencha com "v[i] = 100*i", sendo que o número de elementos do vetor é lido do teclado. A área de memória alocada deve ser definida em função do tamanho do vetor.
- Mude o programa anterior, escrevendo funções separadas para: (a) alocar o vetor e preenchê-lo com zeros; (b) preencher o vetor; e (c) imprimir o vetor.
- Escreva um programa que aloque dinamicamente uma matriz "m" e a preencha com "m[i][j] = i+j", sendo que o número de linhas e colunas são lidos do teclado. A área de memória alocada deve ser definida em função do tamanho da matriz. Este exercício **não é tão simples quanto parece**, veja sugestões de resolução nesta página e nesta também.
- Mude o programa anterior, escrevendo funções separadas para: (a) alocar a matriz; (b) preencher a matriz; e (c) imprimir a matriz.
- Escreva um programa em C para: (a) criar uma lista encadeada simples com 1000 inteiros (com valores 1, 2, ..., 1000); (b) percorrer a lista criada, imprimindo o valor contido em cada elemento. Cada elemento da lista encadeada é um *struct* com dois campos: um valor inteiro e um ponteiro para o próximo elemento.