UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Fundamentos de Programação

CP41F

Conversão de tipos. Alocação dinâmica de memória. Recursão.

Aula 16

Prof. Daniel Cavalcanti Jeronymo

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação – 1º Período
2016.1

Plano de Aula

- Conversão de tipos.
- Alocação dinâmica de memória.
- Recursão.

Conversão de tipos

- Conversão de tipos (*type casting*)
  - Utilizado para tratar um tipo de dado como outro tipo
  - Pode resultar em perda de informação
- Implícita
  - Promoção de inteiros, promoção de double, etc
- Explícita
  - Operador de conversão

Conversão de tipos

- Implícita
  - Se necessário tipos são convertidos para supertipos (menor para maior)
- Atribuição de valores e operações sofrem promoção de inteiros:

```
char a = 30, b = 40, c = 10;
char d = (a * b) / c;
printf("%d ", d);
```

Porque não ocorre overflow em **d**?

- O mesmo ocorre em comparações:

```
double  d = 1.0;
int     i = 0;

if (d > i) printf("Maior!\n");
else printf("Menor!\n");
```

Conversão de tipos

- Implícita
  - Cuidado!

```
unsigned int maior = 10;
int menor = -1;

if (maior > menor) printf("Maior!\n");
else printf("Menor!\n");
```

Conversão de tipos

- Explícita
  - Uso do operador (tipo)
- Força o tratamento de uma variável ou expressão como um determinado tipo
- Como corrigir o exemplo anterior utilizando o operador?

Conversão de tipos

- Explícita
  - Cuidado aqui também!

O que ocorre se o conteúdo de **maior** for superior a **INT_MAX**?

```
unsigned int maior = 10;
int menor = -1;

if ((int)maior > menor) printf("Maior!\n");
else printf("Menor!\n");
```

Conversão de tipos

- Explícita
  - Forçando um erro


```
unsigned int maior = INT_MAX+1;
int menor = -1;

printf("%u\n", maior);

if ((int)maior > menor) printf("Maior!\n");
else printf("Menor!\n");
```

Conversão de tipos

- Explícita
  - O comportamento do código abaixo não é intuitivo
  - Como arrumá-lo?

```
int x = 9, y = 10;
float z = x/y;

printf("%f\n", z);
```

Conversão de tipos

- Explícita
  - Qual o problema da solução abaixo?

```
int x = 9, y = 10;
float z = (float)(x/y);

printf("%f\n", z);
```

Alocação dinâmica de memória

- Alocação dinâmica
  - Gerenciamento manual de memória na *heap*
- Heap
  - Espaço de memória dedicado para alocação dinâmica
  - Armazenamento livre

Alocação dinâmica de memória

- Bloco de memória (relembrando)
  - Já foi visto em **vetores**
  - Região contígua de memória

- Antes o gerenciamento era automático (na pilha)
- Nessa aula: manual (na heap)

Alocação dinâmica de memória

- Relembrando alguns conceitos:

  - Escopo – define a disponibilização de nomes

```c
/* "bla" - escopo de programa */
void bla() {}

/* "a" - escopo de protótipo */
void bli(int a);

/* "ble" - escopo de arquivo */
static void ble()
{
   /* "i" - escopo de bloco */
   int i;

   /* "label" - escopo de função */
   label:
   printf("algo!");
}
```

Alocação dinâmica de memória

- Relembrando alguns conceitos:
  - Tempo de vida – período de tempo em que o espaço alocado de uma variável é válido: **estático, automático e dinâmico**
  - Visibilidade – define se uma variável pode ser acessada ou não

```
/* "i" externo, visível nesse bloco */
int i = 10;

{
   /* "i" interno, visível nesse bloco */
   /* "i" externo não é visível nesse bloco */
   int i;
}

/* "i" externo é visível nesse bloco */
/* "i" interno não existe mais */
```

Alocação dinâmica de memória

- Gerenciamento automático – limitações

- Necessário conhecer o tamanho do bloco de memória a ser alocado em tempo de compilação (vetores)

- A pilha tem tamanho limitado

- Variáveis na pilha

  - Tempo de vida – automático, dependente do escopo de declaração

Alocação dinâmica de memória

- Gerenciamento manual
- Biblioteca: stdlib.h
- Funções:
  - malloc, realloc, free, calloc

Alocação dinâmica de memória

- Gerenciamento manual – alocação

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

- Aloca um bloco de size bytes na memória, retornando um ponteiro para o início do bloco. O conteúdo do bloco não é inicializado. Se size for zero, comportamento indefinido.

- Em caso de sucesso retorna um ponteiro para o início do bloco alocado. Em caso de falha retorna NULL.

Alocação dinâmica de memória

- Gerenciamento manual – realocação

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

- Redefine o bloco de memória de ptr para um novo tamanho de size bytes. Pode mover o bloco de memória para um novo endereço. Conteúdos são preservados na realocação.

- Se ptr for NULL, se comporta como malloc. Se size for zero, se comporta como free. (*ANSI C - comportamento diferente na nova especificação*)

- Em caso de sucesso retorna um ponteiro para o início do novo bloco ou NULL em caso de falha ou dealocação.

Alocação dinâmica de memória

- Gerenciamento manual – dealocação

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

- Libera (dealoca) um bloco de memória alocado por malloc ou realloc.

- Se ptr não apontar pra um endereço válido de memória, o comportamento é indefinido. Se ptr for NULL não faz nada.

Alocação dinâmica de memória

- Gerenciamento manual
  - exemplo automático vs manual (dinâmico)

- Vetor alocado na pilha (automático):

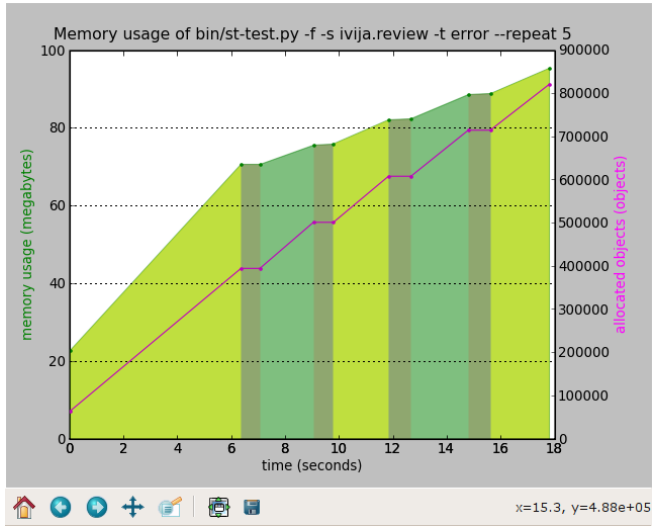
```
int vetor[10];
```

- Vetor alocado na heap (dinâmico):

```
int* vetor = malloc(10 * sizeof(int));
```

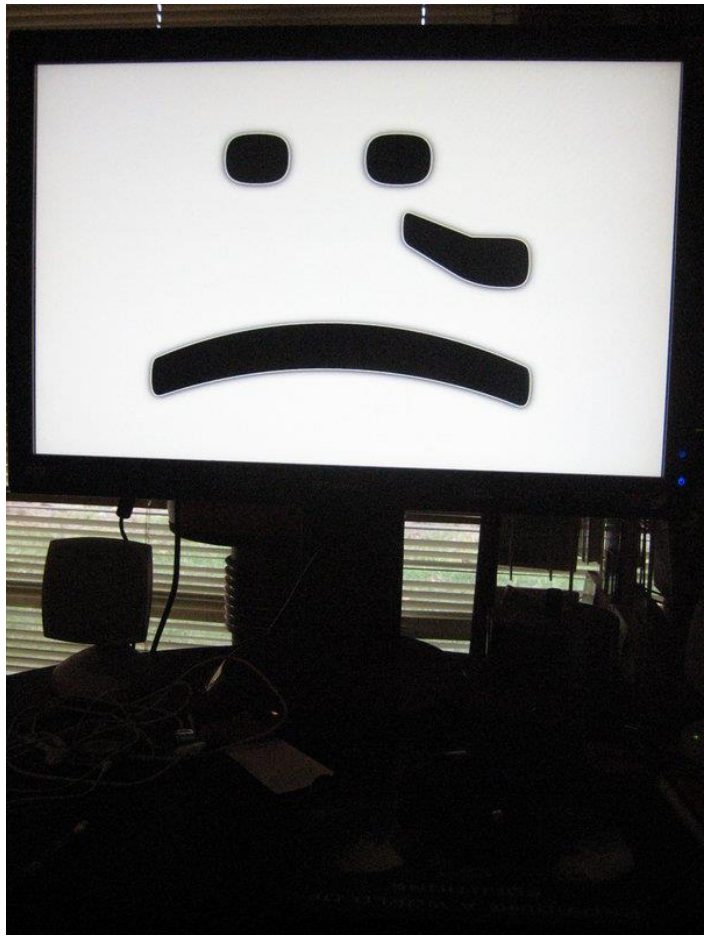
Alocação dinâmica de memória

- Gerenciamento manual
  - Memória alocada dinamicamente persiste até ser liberada
  - Problema de vazamento de memória *(memory leak)*
  - Memória alocada deve ser liberada!



Alocação dinâmica de memória

- Erros comuns:
  - *Memory leak*
  - *Null pointer*
  - *Dangling pointer*
  - *Wild pointer*
  - *Double free*



Recursão

- Recursão:
  - Laços (loops) utilizando funções que chamam a si mesmas

```
unsigned int conta(unsigned int n)
{
   printf("%u\n", n);
   return conta(n+1);
}

int main()
{
   conta(0);
}
```

Recursão

- Recursão:
  - Observando em que direção a pilha e a heap crescem

```
void overflow()
{
   int n1;
   int* n2 = (int*)malloc(sizeof(int));
   printf("%p %p\n", &n1, n2);
   overflow();
}

int main()
{
   overflow();
}
```

Recursão

- Recursão – Exercício

- Crie um programa que calcula o enésimo elemento da sequência de Fibonnaci utilizando recursividade