Universidade de Brasília

Apontadores e Alocação Dinâmica de Memória

Computação para Engenharia — Tópico 8

Daniel Guerreiro e Silva

Departamento de Engenharia Elétrica (ENE), Faculdade de Tecnologia (FT)

Roteiro

Endereços de variáveis e apontadores

Aplicações de apontadores

Alocação dinâmica de memória

Endereços de variáveis e apontadores

A memória e o programa de computador

O espaço de endereços do programa é o trecho de memória disponível para um programa. Ele se divide basicamente em:

1. área do código executável do programa;
2. área da pilha - área que armazena variáveis locais e parâmetros de funções;
3. área de dados - bem maior que a pilha e que serve para alocar variáveis que não couberam e/ou são muito grandes, além de servir para alocar novos espaços pelo programador, durante a execução.

Este tópico ensinará como definir e manipular dinamicamente esses espaços em memória.

Variáveis

Ao declararmos uma variável `x` como a abaixo:

```
int x = 100;
```

Temos associados a ela os seguintes elementos:

- Um tipo (`int`);
- Um nome (`x`);
- Um endereço de memória ou referência (0xbfd2);
- Um valor (100).

Na memória

O operador *address-of* (&)

Para acessarmos o endereço de uma variável, usamos o operador & ao lado do seu nome:

Exemplo

```
int x = 100;
cout << "Valor de x = " << x << endl;
cout << "Endereco de x = " << &x << endl;
```

Veja o exemplo completo em `x.cpp`.

Apontador

Existem tipos de dados para armazenar os endereços de variáveis.

- Uma variável declarada como um destes tipos é chamada de apontador ou ponteiro.
- Ao atribuir o endereço de uma variável a um apontador, dizemos que o mesmo aponta para a variável.

Exemplo

```
int x;
int *ap_x; //apontador para inteiros

ap_x = &x; /* ap_x aponta para x */
```

Veja o exemplo completo em `ap_x.cpp`.

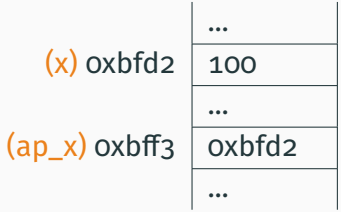
Apontador

Se escrevemos os comandos abaixo dentro do nosso programa:

```
int x = 100;
int *ap_x;

ap_x = &x;
```

Na memória temos (endereços escolhidos arbitrariamente):



Declaração de apontadores em C++

Para declarar um apontador/ponteiro utilizamos o operador unário $*$.

Exemplo

```
int    *ap_int;
char   *ap_char;
float  *ap_float;
double *ap_double;
```

Cuidado ao declarar vários apontadores em uma única linha. O operador $*$ deve preceder o nome de cada variável apontadora.

Veja os exemplos em `apontadores.cpp` e `ap_tabela.cpp`.

Fazendo acesso aos valores das variáveis referenciadas

Um endereço de variável por si só não é muito útil. Para acessarmos o valor de uma variável apontada por um endereço, também usamos o operador `*`:

- Ao precedermos um apontador com este operador, obtemos o equivalente a variável armazenada no endereço em questão.
- `*ap_x` pode ser usado em qualquer contexto que a variável `x` seria.

Exemplo

```
int x, *ap_x;
ap_x = &x;
*ap_x = 3;
```

Veja o exemplo completo em `valores.cpp`.

Aplicações de apontadores

Passagem de parâmetros por valor e referência

Como já vimos, ao passarmos argumentos para uma função, a princípio estes são copiados como variáveis locais da função. Isto é chamado passagem por valor.

- Já vimos também uma forma de passagem por referência, usando o operador `&` ao lado do tipo do parâmetro na declaração.

Exemplo

```
void troca(int& x, int& y) {
  int aux;
  //troca de valores entre variaveis
  aux = x;
  x = y;
  y = aux;
}
```

Passagem de parâmetros por valor e referência

Há uma segunda forma de realizar passagem de argumentos por referência, que consiste em passarmos como argumento o endereço da variável, no lugar de seu valor.

- Ou seja, o mecanismo corresponde a passarmos apontadores para as variáveis que queremos alterar na função.

Retomando: passagem de argumentos por referência

Para indicarmos que será passado o endereço do argumento, usamos o mesmo tipo que usamos para declarar um ponteiro:

```
tipo nome (tipo *parâmetro1, tipo *parâmetro2, …, tipo *parâmetroN) {
    comandos;
}
```

Passagem de argumentos por referência

Para acessarmos o conteúdo da variável apontada pelo ponteiro, usamos o operador *

```
void troca(int *end_x, int *end_y) {
  int aux;
  aux = *end_x;
  *end_x = *end_y;
  *end_y = aux;
}
```

Exemplo: troca_pointer.cpp

```cpp
#include <iostream>
using namespace std;

void troca(int *end_x, int *end_y) {
  int aux;
  //troca de valores entre var.
  aux = *end_x;
  *end_x = *end_y;
  *end_y = aux;
}

int main() {
  int a = 100, b = 200;

  troca(&a, &b);
  cout << "a = " << a << ", b = " << b << endl;
  return 0;
}
```

Repare que, ao chamar a função, é necessário passar os endereços de `a` e `b`.

Apontadores e vetores

A variável que representa um vetor é, por definição, um apontador constante para o primeiro elemento do vetor.

- A operação de indexação corresponde a deslocar este apontador ao longo dos elementos alocados ao vetor.
- Isto pode ser feito de duas formas:
  - Usando o operador de indexação: `v[4]`.
  - Usando aritmética de endereços: `*(v+4)`.

Apontadores e vetores

Esta dupla identidade entre apontadores e vetores é a responsável pelo fato de vetores serem sempre passados por referência e pela inabilidade da linguagem em detectar acessos fora dos limites de um vetor.

Veja o exemplos em `ap_e_vetor.cpp` e `cadeias.cpp`.

Alocação dinâmica de memória

Alocação dinâmica de memória

Além de reservarmos espaços de memória com tamanho fixo na forma de variáveis locais, podemos reservar espaços de memória de tamanho arbitrário e acessá-los através de apontadores.

- Desta forma podemos escrever programas mais flexíveis, pois nem todos os tamanhos devem ser definidos aos escrever o programa.
  - Por exemplo, a memória necessária pode depender da entrada dada pelo usuário.

A alocação e liberação destes espaços na área de dados (heap) da memória é feito por meio de dois operadores em C++:

`new`: Aloca um espaço de memória.

`delete`: Libera um espaço de memória.

Operador `new`

```
ponteiro = new tipo;
ponteiro = new tipo[numero_de_elementos];
```

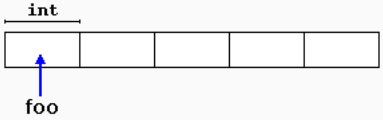
O operador aloca a memória necessária e retorna um ponteiro para o início do bloco.

Operador `new`

Exemplo: um vetor de 5 inteiros

```
int *foo;

foo = new int[5];
```



- O sistema aloca espaço para 5 elementos do tipo `int` e retorna um ponteiro para o primeiro elemento da sequência, que é atribuido a `foo`.
- Como já visto, o primeiro elemento apontado por `foo` daí pode ser acessado com a expressão `foo[0]` ou `*foo`.
- O segundo elemento pode ser acessado com a expressão `foo[1]` ou `*(foo+1)`, e assim por diante...

Outro exemplo - operador `new`

Analise o exemplo do arquivo `new.cpp` e o esquema abaixo, que ilustra a situação na memória ao executá-lo (endereços escolhidos arbitrariamente):

	...
(p) oxafd2	oxbf00
	...
(*p ou p[0]) oxbf00	0
(*(p+1) ou p[1]) oxbf01	1
(*(p+2) ou p[2]) oxbf02	2
	...
(*(p+99) ou p[99]) oxbf63	99
	...

Operador `delete`

Uma vez que não seja mais necessário, podemos liberar o uso de um bloco de memória, permitindo que este espaço seja reaproveitado depois ou por outro programa:

```
delete ponteiro;
delete[] ponteiro;
```

- O primeiro comando é para liberar espaço de um único elemento alocado com o comando `new`, enquanto o segundo comando libera espaço alocado para vetores
- Deve ser passado para o comando `delete` exatamente o mesmo ponteiro retornado pelo operador `new`.

Operador `delete`

Exemplo: alocando e liberando um vetor de 5 inteiros

```
int *p;
p = new int[5];
delete[] p;
```

Veja o exemplo completo em `delete.cpp`.

Exemplo: alocando e liberando memória de matrizes

No caso de matrizes ou vetores multidimensionais, devemos alocar um vetor de apontadores (um apontador por linha) e depois um vetor de elementos para cada linha.

- Na hora de liberar o espaço, temos também que aplicar a operação `delete` para cada linha.

Veja o exemplo em `transpostadinamico.cpp`.

Exemplo final: notasdinamico.cpp

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

void leia_notas(double nota[], int n){
  for (int i = 0; i < n; i++) { //leitura das notas
    cout << "Nota do aluno " << i+1 << ": ";
    cin >> nota[i];
  }
}
//calcula a media do vetor v
double media(double v[], int n){
  double m = 0.0;
  for (int i = 0; i < n; i++) //calculo da media
      m += v[i];
  m /= n;
  return m;
}
//calcula desv. padrao do vetor v de media med
double desvpad(double v[], int n, double med){
  double dv = 0.0;
  for (int i = 0; i < n; i++)
      dv += (v[i]-med)*(v[i]-med); //calculo do desvio padrao
  dv = sqrt(dv/n);
  return dv;
}
```

Continuação: notasdinamico.cpp

```cpp
int main() {
  double *notas, mu, sigma;
  int tamanho;

  cout << "Numero de alunos: ";
  cin >> tamanho;

  notas = new double[tamanho]; //alocacao dinamica do vetor
  leia_notas(notas, tamanho); //chama procedimento que le em notas

  mu = media(notas, tamanho); //chama funcao que calcula media
  sigma = desvpad(notas, tamanho, mu); //chama funcao que calcula desv. padrao

  delete[] notas; //liberacao da memoria usada pelo vetor;
  cout  << "Nota media = "
        << fixed
        << setprecision(2)
        << mu
        << "\nDesvio padrao = "
        << sigma
        << endl;

  return 0;
}
```

Dilbert


JOB INTERVIEW
DO YOU HAVE ANY SALES EXPERIENCE?
NO, BUT I . . .
OKAY. WHAT-EVER.
THERE'S NO BASE PAY. YOU ONLY GET PAID ON COMMISSION.
AND YOU'LL NEED A SPECIAL LAPTOP FOR THIS JOB.
YOU CAN BUY IT FROM OUR COMPANY WITH A 5% EMPLOYEE DISCOUNT.
YOU'RE HIRED.
YES! AND MY FRIENDS TOLD ME I WOULD NEVER FIND A SALES JOB IN THIS WEAK ECONOMY!
BY THE WAY, WHAT DOES THE COMPANY SELL?
WE SELL LAPTOPS TO IDIOTS.
E-mail: SCOTTADAMS@AOL.COM
© 2001 Scott Adams, Inc. Dist. by UFS, Inc.
www.dilbert.com