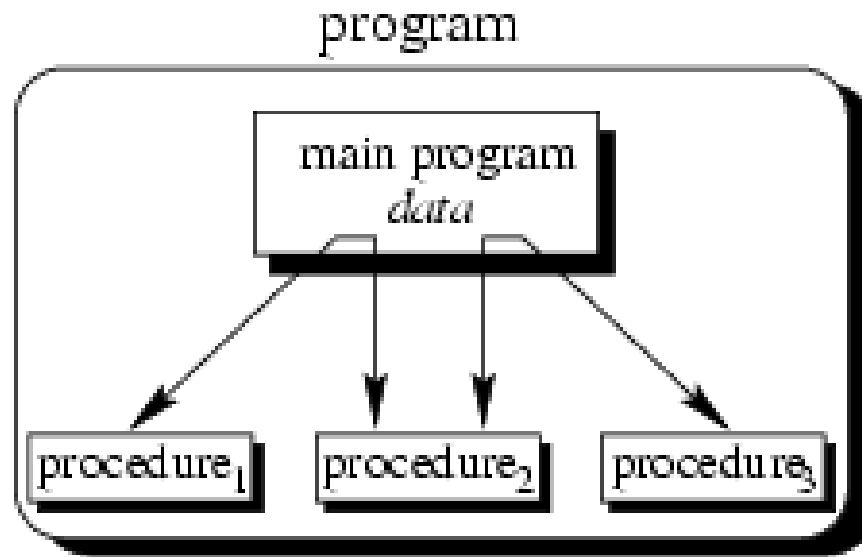
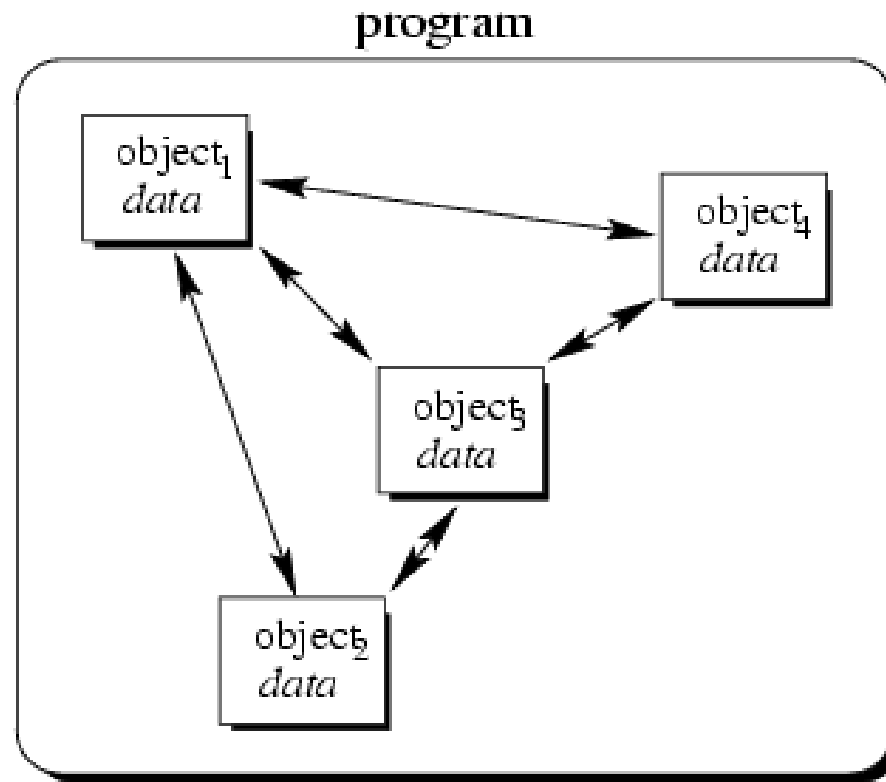


# Conceito de Procedural



- O programa principal coordena as chamadas para os procedimentos e entrega os dados apropriados como parâmetros.

# Conceito Orientada a Objetos



- Objetos do programa interagem enviando mensagens uns aos outros

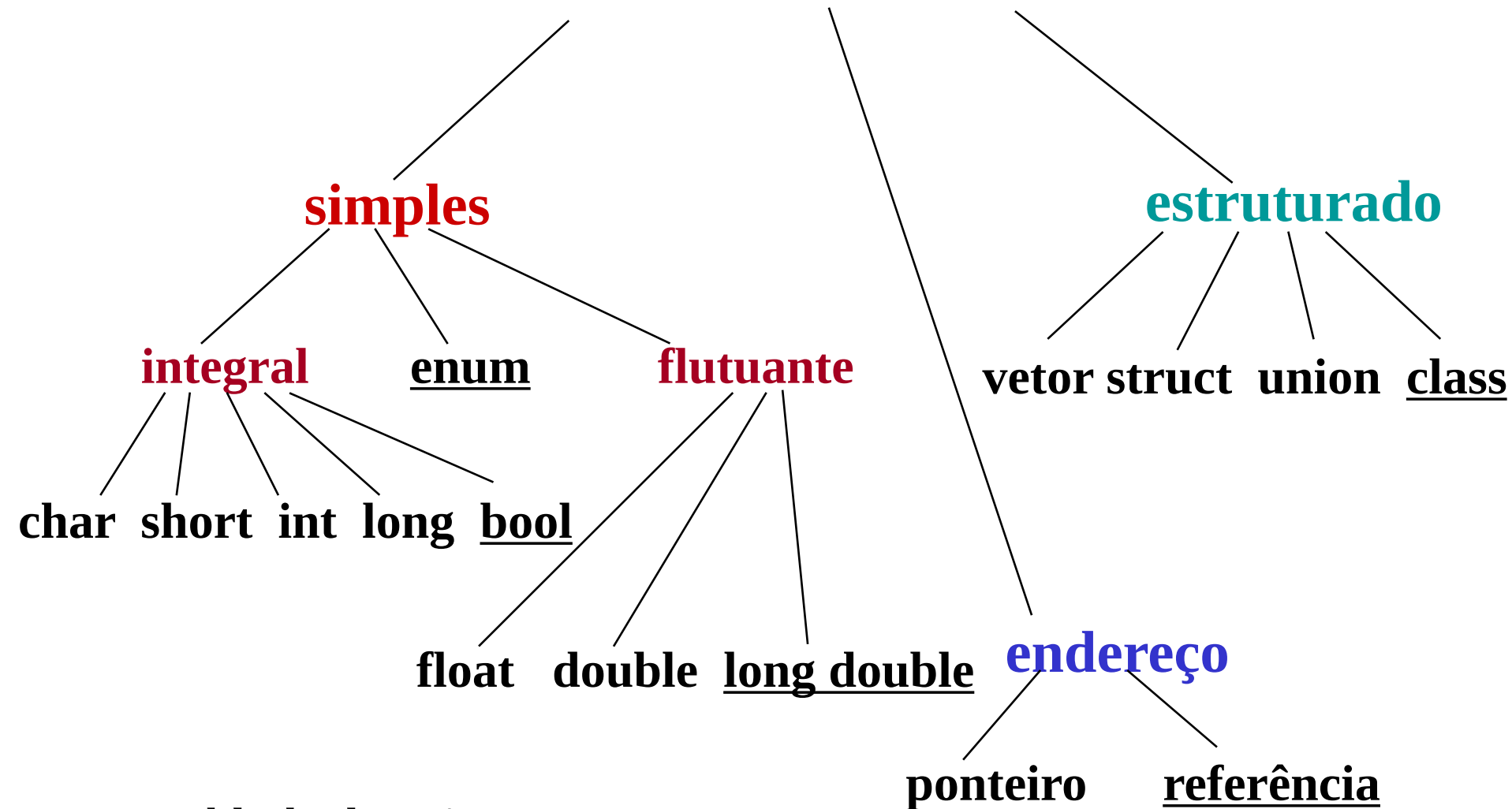
# C ++

- Suporta Abstração de Dados
- Suporta POO
  - Encapsulamento
  - Herança
  - Polimorfismo
- Suporta programação genérica
  - Containers
    - Pilha de caracteres, int, double etc
  - Algoritmos genéricos
    - sort(), copy(), search() em qualquer container Stack / Vector / List

# Ponteiros, dados dinâmicos, e tipos de referência

- Comentário sobre ponteiros
- Variáveis de referência
- Alocação dinâmica de memória
  - Operador **new**
  - Operador **delete**
  - Alocação de memória dinâmica para matrizes

# C ++ Tipos de dados



Tipos sublinhados não presentes em C

# Lembre-se que. . .

char str [8];

- **str** é o endereço base do vetor.
- Nós dizemos que **str** é um ponteiro porque o seu valor é um endereço.
- É um **ponteiro constante**, porque o valor de **str** em si não pode ser alterado por atribuição. Ele “aponta” para a posição de memória de um char.

6000

|         |     |     |      |     |     |     |     |
|---------|-----|-----|------|-----|-----|-----|-----|
| ‘O’     | ‘l’ | ‘á’ | ‘\0’ |     |     |     |     |
| str [0] | [1] | [2] | [3]  | [4] | [5] | [6] | [7] |

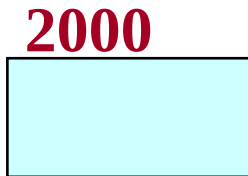
# Endereços na memória

- Quando uma variável é declarada, uma memória suficiente para armazenar um valor desse tipo de dado é alocado em um local de memória não utilizado. Este é o endereço da variável

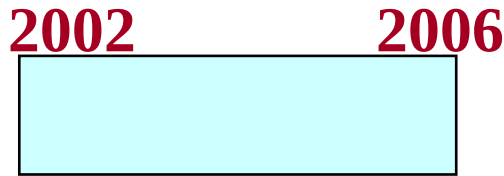
```
int x;
```

```
float nro;
```

```
char ch;
```



**x**



**nro**



**ch**

# Obtenção de endereços de memória

- O endereço de uma *variável não-vetorizada* pode ser obtido usando o **operador de endereço (&)**

```
int x;  
float nro;  
char ch;
```

2000

**x**

2002

**nro**

2006

**ch**

```
cout << "Endereço de x é" << &x << endl;
```

```
cout << "Endereço do número é" << &nro << endl;
```

```
cout << "Endereço do ch é" << &ch << endl;
```



# O que é uma variável de ponteiro?

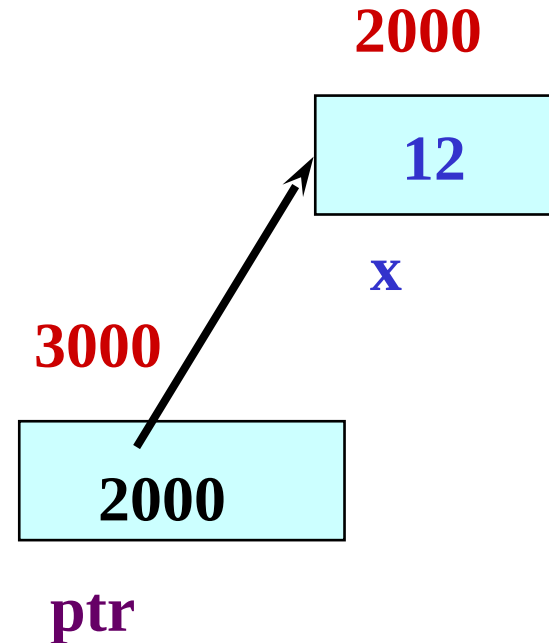
- Uma variável de ponteiro é uma **variável cujo valor é o endereço de um local na memória**
- Para declarar uma variável ponteiro, você deve especificar o tipo de valor que o ponteiro irá apontar, por exemplo:

```
int* ptr; // ptr irá apontar para o endereço de um int
```

```
char* q; // q irá apontar para o endereço de um char
```

# Usando uma variável de ponteiro

```
int x;  
x = 12;  
  
int* ptr;  
ptr = &x;
```



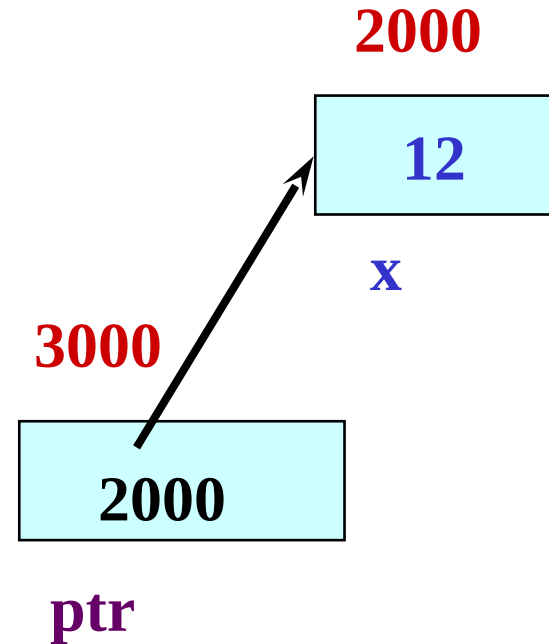
**NOTA:** Porque ptr contém o endereço de x,  
dizemos que ptr “aponta para” x

# **\***: Operador que desreferencia

```
int x;  
x = 12;
```

```
int* ptr;  
ptr = &x;
```

```
cout << *ptr;
```

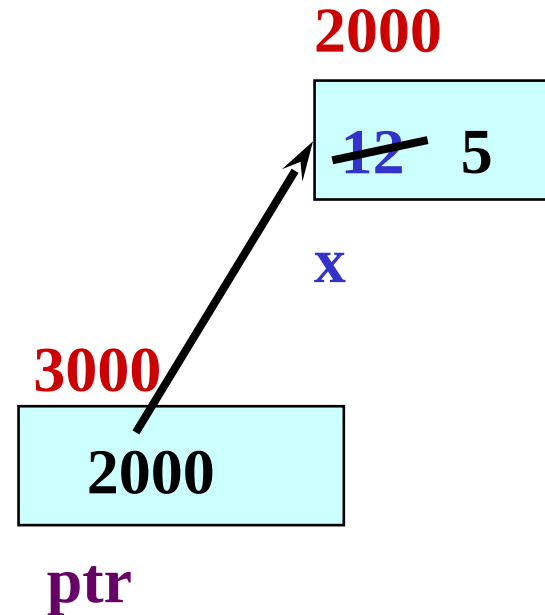


**NOTA:** Desreferencia = Não trata o ponteiro como referência mas sim trata o seu valor

O valor apontado por **ptr** é denotada por  $*ptr_1$

# Usando o Operador que desreferencia

```
int x;  
x = 12;  
  
int* ptr;  
ptr = &x;  
  
*ptr = 5;
```



**// Altera o valor no endereço que  
ptr aponta para 5**

# Auto-Test de Ponteiros

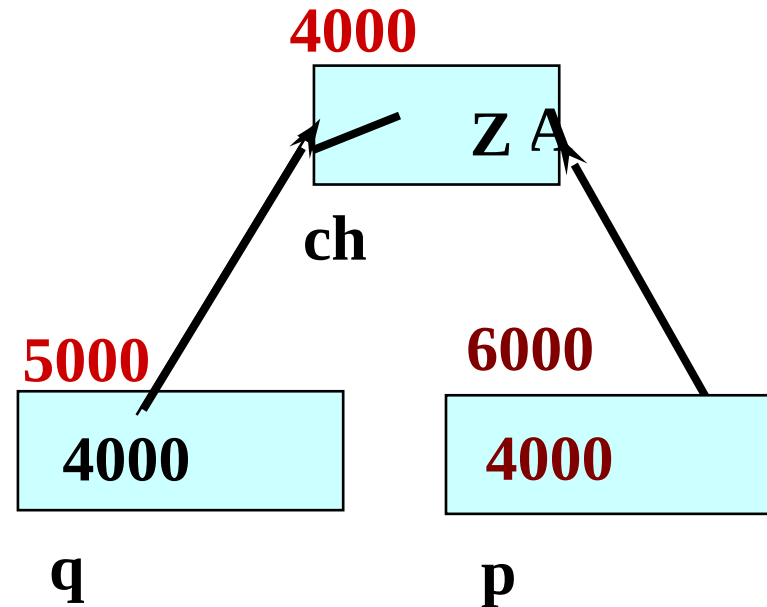
```
char ch;  
ch = 'A';
```

```
char* q;  
q = &ch;
```

```
*q = 'Z';
```

```
char* p;
```

```
p = q;
```



// RHS tem valor 4000

// agora p e q apontam para ch

# Usando um ponteiro para acessar os elementos de uma String

```
char msg [] = "Olá";
```

```
char* ptr;
```

```
ptr = msg;
```

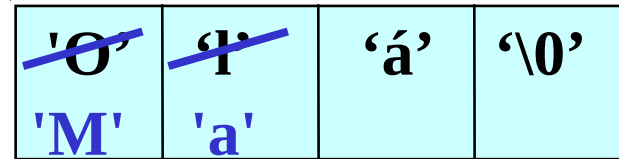
```
*ptr = 'M';
```

```
ptr++;
```

```
*ptr = 'a';
```

msg

3000



3001

ptr

# Exercício 2

Dado o seguinte trecho de programa:

```
int i=3, j=5;
```

```
int *p, *q;
```

```
p = &i;
```

```
q = &j;
```

Qual é o valor das seguintes expressões ?

a) `p == &i;`

b) `*p - *q`

c) `**&p`

d) `3 * *p / (*q) + 7`

# Variáveis de referência (C++)

Variável de referência = *alias/apelido para outra variável*

- Contém o endereço de uma variável (como um ponteiro)
- Não há necessidade de realizar qualquer desreferenciação (ao contrário de um ponteiro) não precisa do \*
- Deve ser inicializada quando é declarada


```
int x = 5;
```

```
int &z = x;    // z é um outro nome para x
```

```
int &y;        // Erro: referência deve ser inicializado
```

```
cout << x << endl; -> imprime 5
```

```
cout << z << endl;    -> imprime 5
```



```
z = 9;        // mesmo que x = 9;
```

```
cout << x << endl;    -> imprime 9
```

```
cout << z << endl;    -> imprime 9
```



# Porque variáveis de referência?

- São principalmente usados como parâmetros de função
- Vantagens do uso de referências:
  - você não tem que passar o endereço de uma variável
  - você não tem que desreferenciar a variável dentro da função chamada – não precisa usar o \*

# Exemplo Variáveis de referência

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

// prototipos de funcoes
void p_troca (int *, int *);
void r_troca (int&, int&);

int main (void){
    int v = 5, x = 10;
    cout << v << " " << x << endl;
    p_troca(&v, &x);
    cout << v << " " << x << endl;
    r_troca(v, x);
    cout << v << " " << x << endl;
    return 0;
}
```

```
void p_troca(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

```
void r_troca(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

# Exemplo 2 – Referência &

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

void calculaRaizes(float a, float b, float c, float &x1,
float &x2) {
    float delta = b * b - 4 * a * c;
    if (delta > 0) {
        x1 = (-b + sqrt(delta))/2 * a;
        x2 = (-b - sqrt(delta))/2 * a;
    }
}

int main() {
    float a, b, c; // caso de teste: 1, -5, 6
    float x1, x2; // resultado: 2 e 3
    cout << "Coeficientes da equacao do 2o grau: ";
    cin >> a >> b >> c;
    cout << endl;
    calculaRaizes(a, b, c, x1, x2);
    cout << "Raizes x1: " << x1 << " e " << x2 << endl;
    return 0;
}
```

# Exercício 2

- Escrever um procedimento chamado calculaVolumeArea que permita calcular a **área e o volume** de um paralelepípedo. Escreva também um programa principal para obter as dimensões do usuário, invocar esse procedimento e exibir o resultado na tela.
  - $\text{Area} = 2 * A * L + 2 * A * P + 2 * L * P$
  - $\text{Volume} = A * L * P$
- Escrever um procedimento com o mesmo nome (sobrecarga de calculaVolumeArea) mas que receba a altura e o raio de um cilindro e calcule **a área e o volume** do mesmo.
  - Area:  $2 * (\pi * R^2 + 2 * \pi * R * h)$
  - Volume:  $\pi * R^2 * h$

# Alocação dinâmica de memória

Em C e C ++, três tipos de memória são utilizados por programas:

- **Memória estática** - onde as variáveis globais e estáticas vivem
- **Memória heap** - dinamicamente alocados em tempo de execução  
- memória "administrada" e acessada usando ponteiros
- **Memória de pilha** - usado por variáveis automáticas e parâmetros de funções

## Static Memory

Global Variables  
Static Variables

Heap Memory (or free store)  
Dynamically Allocated Memory  
(Unnamed variables)

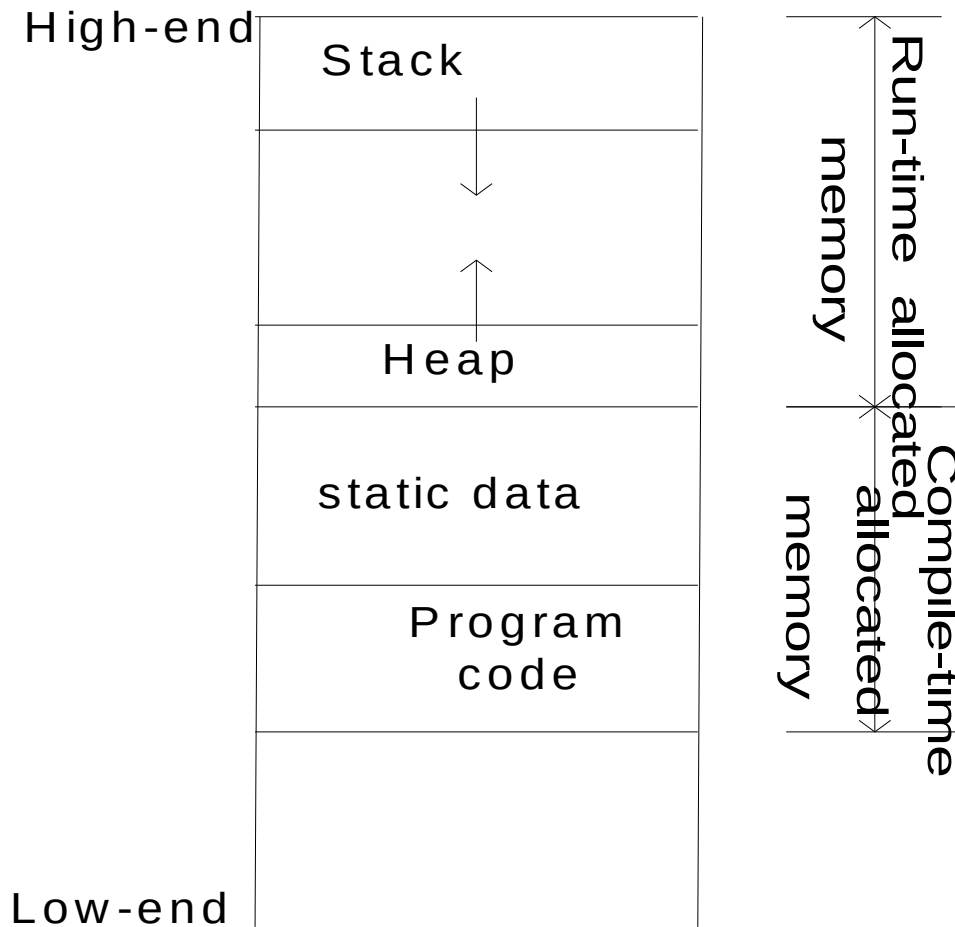
## Stack Memory

Auto Variables  
Function parameters

# 3 Tipos de dados do programa

- **DADOS ESTÁTICOS**: Atribuídos no momento da compilação
- **DADOS DINÂMICOS**: explicitamente alocado e desalocado durante a execução do programa, instruções C ++ escritos por programadores usando operadores **new** e **delete**
- **DADOS AUTOMATICOS**: Criado automaticamente na entrada da função, reside no escopo de ativação da função e, é destruído ao retornar a função

# Memória Dinâmica Diagrama de alocação



# Alocação dinâmica de memória

- *Em C*, Funções como `malloc()` e `free()` são usados para alocar dinamicamente a memória da **heap**.
- *Em C ++*, Isto é conseguido utilizando os operadores **new** e **delete**
- **new** é usado para alocar memória durante o tempo de execução
  - retorna um ponteiro para o endereço onde o objeto foi armazenado
  - sempre retorna um ponteiro para o tipo que segue o **new**



# Sintaxe Operador **new**

**new DataType**

**new DataType [IntExpression]**

- Se a memória está disponível, em uma área chamada heap **new** aloca o objeto solicitado ou matriz, e retorna um **ponteiro** para o endereço da memória alocada.
- Caso contrário, o programa termina com mensagem de erro.
- O objeto alocado dinamicamente existe até que o operador de exclusão (delete) destrua ele.

# Operador new

```
char* ptr;
```

```
ptr = new char;
```

```
*ptr = 'B';
```

```
cout << *ptr;
```

2000

5000

ptr

5000

'B'

**NOTA:** Os dados dinâmicos não tem nome de variável

# O ponteiro **NULL**

- Há um ponteiro constante chamado de “ponteiro nulo” denotado por NULL
- NULL não é endereço de memória 0.
- **NOTA:** É um erro excluir (delete) a referência a um ponteiro cujo valor é NULL. Tal erro pode causar o encerramento do programa, ou um comportamento irregular. É o trabalho do programador verificar isso.

```
while (ptr != null) {  
    ...    // ok para usar *ptr aqui  
}
```

# Sintaxe Operador **delete**

**delete ponteiro**

**delete [] ponteiro**

- O objeto ou vetor atualmente apontado pelo ponteiro é **desalocado**, e o valor do ponteiro é indefinido e a memória é devolvida à memória livre.
- **Boa prática:** definir o ponteiro para a memória liberada para NULL – não é feito automaticamente
- Os colchetes são usados com o operador delete para desalocar um vetor alocado dinamicamente.

# Operador delete

```
char* ptr;
```

```
ptr = new char;
```

```
*ptr = 'B';
```

```
cout << *ptr;
```



```
delete ptr;
```

2000

5000

ptr

**NOTA:**

**delete** desaloca a  
memória apontada por ptr

# Exercício 3

Crie um programa que

- a. Declare um ponteiro para inteiro
- b. Crie uma variável inteira alocada dinamicamente usando new, armazenando a referência no ponteiro acima
- c. Atribua o valor 3 a essa variável usando o ponteiro do item a)
- d. Imprima o valor da variável e o endereço de memória onde essa variável se localiza
- e. Libere essa variável antes de finalizar o programa usando delete.

# Exercício 4

Considerando a estrutura:

```
struct Vetor {  
    float x;  
    float y;  
    float z;  
};
```

Crie um programa que

- Declare um ponteiro para um struct Vetor
- Crie uma estrutura struct Vetor, alocada dinamicamente (new)
- Atribua os valores 1.5, 4.2 e 7.9 aos atributos x, y e z dessa estrutura utilizando o operador ->
- Imprima o valor dos atributos x, y e z da estrutura acima
- Imprima o endereço de memória onde a estrutura se localiza na memória
- Libere essa estrutura antes de finalizar o programa usando delete.

# Exercício 5

Crie um programa que permita o usuário digitar as médias finais de todos os alunos de uma determinada disciplina e posteriormente calcule e exiba:

- a. Média da classe
- b. A menor nota
- c. A maior nota

Antes de digitar as médias o programa deve solicitar a quantidade de alunos da classe e criar um vetor com a quantidade exata de alunos.

Utilize as funções `new` e `delete` para criar e liberar esse vetor.