

Chapter 4. Pointers

Programming Concepts in Scientific Computing
EPFL, Master class

September 27, 2024

Pointers and the Computer Memory

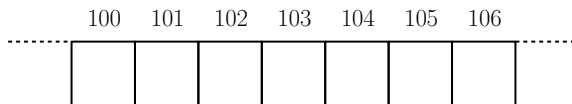
Addresses

```
int x = 1;  
int y = 2;  
std::cout << &x << "\n";
```

Pointers and the Computer Memory

Addresses

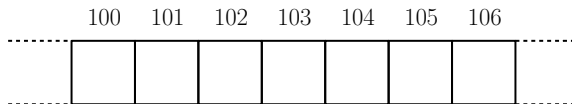
```
int x = 1;  
int y = 2;  
std::cout << &x << "\n";
```



Pointers and the Computer Memory

Addresses

```
int x = 1;  
int y = 2;  
std::cout << &x << "\n";
```



Debug this program (breakpoint)

```
(gdb) x/2wx &x
```

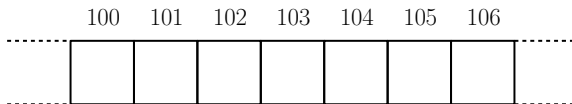
```
(gdb) x/2wx &y
```

Want to know more ? \Rightarrow (gdb) help x

Pointers and the Computer Memory

Addresses

```
int x = 1;  
int y = 2;  
std::cout << &x << "\n";
```



Debug this program (breakpoint)

```
(gdb) x/2wx &x
```

```
(gdb) x/2wx &y
```

Want to know more ? \Rightarrow (gdb) help x

Pointers and the Computer Memory

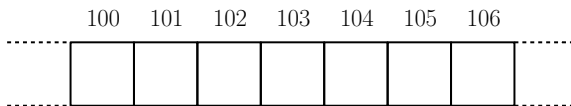
Addresses

```
int total_sum = 10;
```

Pointers and the Computer Memory

Addresses

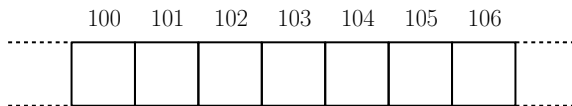
```
int total_sum = 10;
```



Pointers and the Computer Memory

Addresses

```
int total_sum = 10;
```



Getting an address: &

```
std::cout << &total_sum << "\n";
```


Pointers and the Computer Memory

Pointer Variables

Using the *star* in types:

```
int *p_x;
```

Pointers and the Computer Memory

Pointer Variables

Using the *star* in types:

```
int *p_x;
```

Example of use

```
// x stores an int precision number  
int x = 3;  
// p_x stores the address of an int  
int *p_x = &x;
```

Try it with CLion !

```
int x = 3;
```

```
int *p_x = &x;
```

```
std::cout << p_x << std::endl;
```

Try it with CLion !

```
int x = 3;  
int *p_x = &x;  
  
std::cout << p_x << std::endl;
```

Debug this program (breakpoint)

Try it with CLion !

```
int x = 3;  
int *p_x = &x;  
  
std::cout << p_x << std::endl;
```

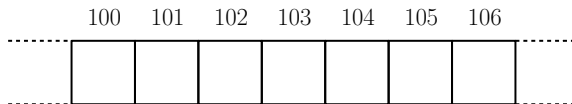
Debug this program (breakpoint)

Hit this command in gdb:

(gdb) x/3wx &x

Try it with CLion !

What is the memory structure ?



String of characters

Declare an array of characters:

```
char name[250] = "yopla";
```

String of characters

Declare an array of characters:

```
char name[250] = "yopla";
```

However, I can write:

```
char *ptr = name;
```


String of characters

Declare an array of characters:

```
char name[250] = "yopla";
```

However, I can write:

```
char *ptr = name;
```

because an array of characters is actually a pointer!

Aliasing/de-reference

```
int y = 3;  
int *p_x = &y;
```

Aliasing/de-reference

```
int y = 3;  
int *p_x = &y;
```

```
// This changes the value of y  
*p_x = 1;
```

Pointers and the Computer Memory

Addresses

```
int x = 1;  
int y = 2;  
*(&y + 1) = 3;
```

What does this do ?

Main: argv structure

Considering this code:

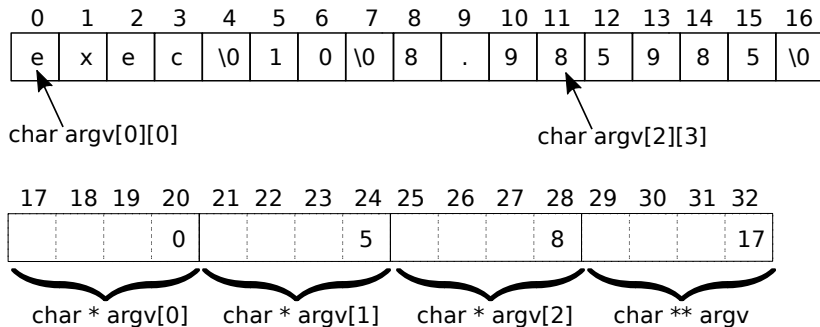
```
int main(int argc, char ** argv){  
    int p = atoi(argv[1]);  
    double z = atof(argv[2]);  
}
```

If I launch the executable like this:

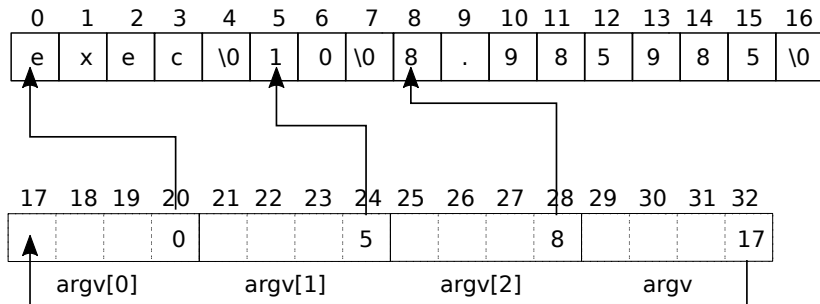
```
> ./exec 10 8.985985
```

What is the memory structure in that case ?

Main: argv structure



Main: argv structure



Warnings on the Use of Pointers

What is the problem with this code ?

```
int *p_x;
```

```
*p_x = 1;
```


Warnings on the Use of Pointers

```
// p_x stores the address of a int  
// not yet specified  
int *p_x;  
  
// trying to assign 1.0 in an unspecified  
// memory location  
*p_x = 1;
```

Dynamic Allocation of Memory

```
int *x = new int;
```

Dynamic Allocation of Memory

```
int *x = new int;
```

```
*x = 10;
```

Dynamic Allocation of Memory

```
int *x = new int;
```

```
*x = 10;
```

```
delete x;
```

Dynamic Allocation of Memory

Vectors

```
double *x = new double[10];  
double *y = new double[10];
```

Dynamic Allocation of Memory

Vectors

```
double *x = new double[10];  
double *y = new double[10];  
  
for (int i = 0; i < 10; i++) {  
    x[i] = double(i);  
    y[i] = 2.0 * x[i];  
}
```

Dynamic Allocation of Memory

Vectors

```
double *x = new double[10];  
double *y = new double[10];  
  
for (int i = 0; i < 10; i++) {  
    x[i] = double(i);  
    y[i] = 2.0 * x[i];  
}
```

```
delete[] x;  
delete[] y;
```

Matrices

```
int rows = 5, cols = 3;  
double **A = new double *[rows];
```


Matrices

```
int rows = 5, cols = 3;  
double **A = new double *[rows];  
  
for (int i = 0; i < rows; i++) {  
    A[i] = new double[cols];  
}
```

Matrices

```
int rows = 5, cols = 3;  
double **A = new double *[rows];
```

```
for (int i = 0; i < rows; i++) {  
    A[i] = new double[cols];  
}
```

```
// you can access the values of the array with  
A[2][4] = 5;
```

Matrices

```
int rows = 5, cols = 3;
double **A = new double *[rows];

for (int i = 0; i < rows; i++) {
    A[i] = new double[cols];
}

// you can access the values of the array with
A[2][4] = 5;

// At the end: deallocate the memory
for (int i = 0; i < rows; i++) {
    delete[] A[i];
}
delete[] A;
```

ROW MAJOR format

```
double *p_a = new double[rows * cols];
```

ROW MAJOR format

```
double *p_a = new double[rows * cols];
```

```
double **A = new double *[rows];  
for (int i = 0; i < rows; i++) {  
    A[i] = &p_a[i * cols];  
    A[i] = p_a + i * cols;  
}
```

ROW MAJOR format

```
double *p_a = new double[rows * cols];
```

```
double **A = new double *[rows];  
for (int i = 0; i < rows; i++) {  
    A[i] = &p_a[i * cols];  
    A[i] = p_a + i * cols;  
}
```

```
// you can access the values of the array with  
A[2][4] = 5;  
// or with  
p_a[2 * cols + 4] = 5;
```

ROW MAJOR format

```
double *p_a = new double[rows * cols];
```

```
double **A = new double *[rows];  
for (int i = 0; i < rows; i++) {  
    A[i] = &p_a[i * cols];  
    A[i] = p_a + i * cols;  
}
```

// you can access the values of the array with

```
A[2][4] = 5;
```

// or with

```
p_a[2 * cols + 4] = 5;
```

// At the end: de-allocate the memory

```
delete[] A;
```

```
delete[] p_a;
```

ROW MAJOR format

- ▶ ROW MAJOR: C, C++
- ▶ COLUMN MAJOR: Matlab and Fortran

Tips

- ▶ Pointer Aliasing: e.g. coding

- ▶ $C = A \cdot B$

- ▶ $A = A \cdot B$

Tips

- ▶ Pointer Aliasing: e.g. coding

- ▶ $C = A \cdot B$

- ▶ $A = A \cdot B$

- ▶ Dynamic Allocation: check non-null pointer:

```
int *p_x = new int;  
assert(p_x != NULL);
```

Tips

- ▶ Pointer Aliasing: e.g. coding

- ▶ $C = A \cdot B$

- ▶ $A = A \cdot B$

- ▶ Dynamic Allocation: check non-null pointer:

```
int *p_x = new int;  
assert(p_x != NULL);
```

- ▶ Every new Has a delete

Pointers

Take away message

- ▶ **pointer**: variable storing an address (of another variable)
- ▶ **Type *ptr**: pointers are typed (`int*`, `double*`, ...)
- ▶ **&var**: get the address of *var* (`int* ptr = &a;`)
- ▶ ***ptr**: access the pointed variable (`*ptr = 1;`)
- ▶ **C-array**: a pointer to the first item of the array (`int vec[256];`
~ `int *vec;`)
- ▶ **char **argv**: array of array of characters
- ▶ **Dynamic allocation**: *long life* variables should be **created**
and **destructured** using **new&delete**

```
double *var = new double;  
// ...  
delete var;
```

```
double *vec = new double[1000];  
// ...  
delete[] vec;
```