11. Pointers

June 22, 2023

1 Pointers

How many of you know about pointers already?

: yes   : no

1.1 1. Introduction to pointers

What is a pointer? A pointer is an address to a memory location. What is a reference? A reference is an alias (another name) of a variable/object.

[ ]:

```
#include <iostream>
int apples = 10;
int *ptr_to_apples;
ptr_to_apples = &apples;
std::cout << "address = " << ptr_to_apples << std::endl;
std::cout << "apples = " << *ptr_to_apples << std::endl;
```

Image source: http://www.btechsmartclass.com/cpp-programming/CPP-Pointers.php

We can declare pointer for each data type:

[ ]:

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

Pointer declaration:

[ ]:

```
int *ptr;
```

Dereferencing (retrieving the data):

[ ]:

```
b = *ptr;
```

Null pointer? Use nullptr C++11

[ ]:

```
int *ptr_to_apples = nullprt;
```

1.2 2. Using pointers

1.2.1 2.1 Pointer arithmetics

We can increment or decrement pointers, thus providing access to the next or to the previous memory address. This is very useful when using arrays.

[ ]:

```
#include <iostream>
int apples_per_person[5] = {1 ,2 ,3 ,4 ,5};
int *ptr_to_apples = &apples_per_person[0];
std::cout << *ptr_to_apples << std::endl;
ptr_to_apples++;
std::cout << *ptr_to_apples << std::endl;
ptr_to_apples+=10;
std::cout << *ptr_to_apples << std::endl;
```

Let's expand on this. Can we have a pointer to a pointer?

: yes : no

Image source: https://www.boardinfinity.com/blog/c-pointers/

1.2.2 2.2 Calling by reference and calling by value

2.2.1 Calling a function with pointers

[ ]:

```
#include <iostream>
void recipe(int *ptr_to_apples){
    *ptr_to_apples += 1;
}
int apples = 10;
int *ptr_to_apples = &apples;
recipe(ptr_to_apples);
//recipe(&apples);
std::cout << apples << std::endl;
```

2.2.2 Calling by reference

[ ]:

```
#include <iostream>
void recipe(int &apples){
    apples += 1;
}
int apples = 10;
recipe(apples);
std::cout << apples << std::endl;
```

2.2.3 Calling by value

[ ]:

```
#include <iostream>
void recipe(int apples){
    apples += 1;
}
int apples = 10;
```

```
recipe(apples);
std::cout << apples << std::endl;
```

1.2.3 2.2 Dynamic memory allocation

So far we have been using the stack to store our arrays. How about allocating/deallocating in the heap using dynamic memory allocation? The new operator allocates the memory and returns the address to the newly allocated memory chunck: pointer-variable = new data-type; . The delete operator releases the memory that the variable points to.

[ ]:

```
int *vali = new int(39);
// ...code...
delete vali;

int *vali = new int[10];
// ...code...
delete[] vali;
```

Can we have pointers to containers? Absolutely!

[ ]:

```
std::vector<int> *v = new std::vector<int>(10);
v->at(2); //Retrieve using pointer to member
v->operator[](2); //Retrieve using pointer to operator member
v->size(); //Retrieve size
vector<int> &vr = *v; //Create a reference
vr[2]; //Normal access through reference
delete v;
```

1.3 3. Now let’s answer some questions

Answer the questions using these symbols:

[ ]:

```
float a = 5;
float & b = a;
```

: b is a reference   : b is a value

[ ]:

```
void foo(int *a) {
print(a);
}
```

: in the function print(), a is a pointer   : in the function print(), a is dereferenced and is a value

---

[ ]:

```
int *p = new int(6) ;
int *a;
```

```
a = p;
```

: a is now equal to 6   : a is a pointer

[ ]:

```
int *p = new int(6) ;
int a;
a = *p;
```

: a is now equal to 6   : a is a pointer

[ ]:

```
struct HouseCosts{
    double energy, heating, water, services;
};

double totalCost(HouseCosts *hc){
    auto sum = (*hc).energy + hc->heating + hc->water + hc->services;
    return sum;
}

HouseCosts hc;
hc.energy = 11.11;
hc.heating = 101.11;
hc.water=22.2;
hc.services=33.33;

auto total_costs = totalCost(&hc);
```

1.4 4. Pointers in classes

[ ]:

```
class HouseCosts{
    public:
    double *energy, *heating;
    int months;
    HouseCosts(int months);
    ~HouseCosts();

    double totalCost(int m);
};

HouseCosts::HouseCosts(int m){
    months = m;

    energy = new double[months];
    heating = new double[months]
}
```

```
HouseCosts::~HouseCosts(){
    delete[] heating;
    delete[] energy;
}

double HouseCosts::totalCost(){
    auto sum = energy + heating + water + services;
    return sum;
}
```

1.5 5. Pointers to objects

[ ]:

```
class HouseCosts{
    public:
    double *energy, *heating;
    int months;
    HouseCosts(int months);
    ~HouseCosts();

    void addtomyHousehold(HouseCosts &rhs);
};

HouseCosts::HouseCosts(int months){
    this->months = months;
    //(*this).months = months;

    energy = new double[this->months];
    heating = new double[this->months]
}

HouseCosts::~HouseCosts(){
    delete[] heating;
    delete[] energy;
}

void HouseCosts::addtomyHousehold(HouseCosts &rhs){
    if(this->months == rhs.months)
        for(auto m:energy){
            this->energy[m] += rhs.energy[m];
            this->heating[m] += rhs.heating[m];
        }
    else
        throw(-1);
}
```