We can also use pointer to declare an array, which called dynamic array.

Firstly you declare a pointer:

int \*p;

Now you have a pointer which supposes to point to an integer type address. but now we haven’t assign the address to the pointer. So it can’t do anything now. You can also initial the pointer to point to null by doing this:

Int \*p = nullptr; or int \*p = NULL; or int \*p = 0;

If you want to declare an array, you need the key word “new”, new follow with the type of data you want to store. When you call “new”, the compiler will select a location and set the location for the type you declare follow “new”

Example:

Int \*p = new int;

Now the pointer p holds an address which is the head(first element) of the array, you can access the location simply by the same way you do for array.

P[0] = 5;

p[7] = 10;

p[3] =7;

…

By this way you didn’t declare the size of the array, you just have the address of first element, so if you want to use this kind of declaration, remember to create a counter to count the size.

How about declare a dynamic array with specific size?

Just add the size with [ ] follow the instruction:

Int \*p = new int[80]; // This will declare an integer array with size 80

Let’s see some real example on the next page.

int main()

{

int \*c = nullptr;

cout << " c : \t" << c << endl;          // c :   **00000000** (because c point to null)

cout << " &c : \t" << &c << endl; // &c :  **00FCFE04** (The address of the pointer c itself)

c = new int;                             // now the system assign an address into c (suppose to store int value)

cout << " &c : \t" << &c << endl;        // &c :  **00FCFE04** (The address of pointer c itself is the same)

cout << " c : \t" << c << endl;          // c :   **01429E30** (The address of first element in the dynamic array)

cout << " &c[0] : \t" << &c[0] << endl;  // &c[0] : **01429E30** (The same, it's the address of first element)

cout << " &c[1] : \t" << &c[1] << endl;  // &c[1] : **01429E34** (The address of the second element)

c[0] = 5;                                // assign the int value to the address of first element

c[1] = 10;                               // assign the int value to the address of second element

cout << " c[0] : \t" << c[0] << endl;    // c[0] : **5** (print the first element, the same like normal array)

cout << " c[1] : \t" << c[1] << endl;    // c[1] : **10** (print the second element, the same like normal array)

}

Let’s see another example:

int main()

{

int A = 10;                           // Declare int variable A = 10

int \*a = &A;                          // Declare a pointer a which point to address of A

int \*b = a;                           // Declare a pointer b which holds the same content of pointer a

cout << "&A  \t" << &A << endl;       // &A      00A5F890

cout << "a  \t" << a << endl;         // a       00A5F890

cout << "&a  \t" << &a << endl;       // &a      00A5F884

cout << "b  \t" << b << endl;         // b       00A5F890

cout << "&b  \t" << &b << endl;       // &b      00A5F878

cout << "A  \t" << A << endl;         // A       10

cout << "\*a  \t" << \*a << endl;       // \*a      10

cout << "\*b  \t" << \*b << endl;       // \*b      10

}

Pointer is a very important concept which you will need in Data Structure. If you can’t truly understand, feel free to come to any SI sessions

I’m glad to help you anytime ~