CSC230

Intro to C++ Lecture 2

Storage classes

- Define the scope (visibility) and life-time of variables and functions
  - auto
  - register
  - static
  - extern
  - mutable

Storage classes

- **auto**
  - Only for local variables inside functions

```
int func(){
   int a = 10;
   auto b;
}
```

- **register**
  - Should be stored in a register, not RAM (no guarantee)

```
int func(){
   int a = 10;
   register int b;
}
```

Storage classes

- **Static**
  - **local** variable: stay in memory during program life-time
  - **Global** variable: data scope is the file itself, not visible to other files
  - Member of class: one copy for all objects of that class

```
#include <iostream>
void func();
static int globalCount = 10; // global static variable
int main()
{
  while(globalCount--)
      func();
  return 0;
}
void func()
{
  static int localCount = 0; // local static variable
  localCount++;
  std::cout << "gloabCount: " << globalCount << "\n" ;
  std::cout << "localCount: " << localCount << std::endl;
}
```

Storage classes

- extern
  - The global variable or the function is defined in a **different file**

```
extern int a = 10;
extern func();
```

- **mutable**
  - In object, a mutable member can be overriden by a constant function
  - More details later

Operators

Arithmetic	+	-	*	/	%	++	--
Relational	==	!=	>	<	>=	<=	
Logical	&&	| |	!				

Bitwise operators

A = 0011 1100
B = 0000 1101

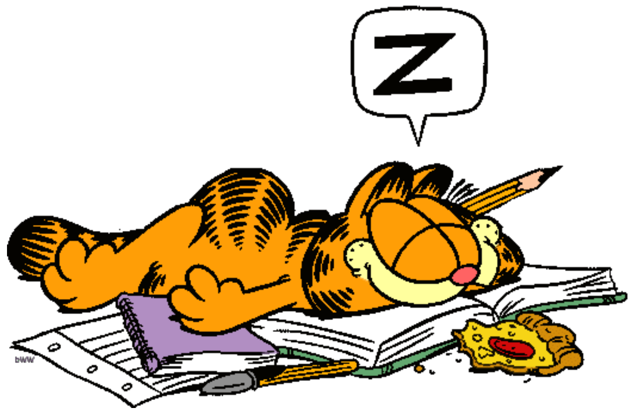
&	Binary AND	A & B = 0000 1100
|	Binary OR	A | B = 0011 1101
^	Binary XOR	A ^ B = 0011 0001
~	Binary ones complement	~A = 1100 0011
<<	Binary Left shift	A << 2 = 1111 0000
>>	Binary Right Shift	A >> 2 = 0000 1111

Other important operators

Cast	int(3.14) returns 3
. ->	Refer to the member of class, structure, union
&	int a; &a is the address of the variable a.
*	If variable a stores a memory address, *a points to that address.

C++ flow control

if
if… else
switch
for loop
while loop
do… while loop
nested loops



```
while(nappingTime){

   if(hungry)
     break;
   else
     continue;

}
```

Functions

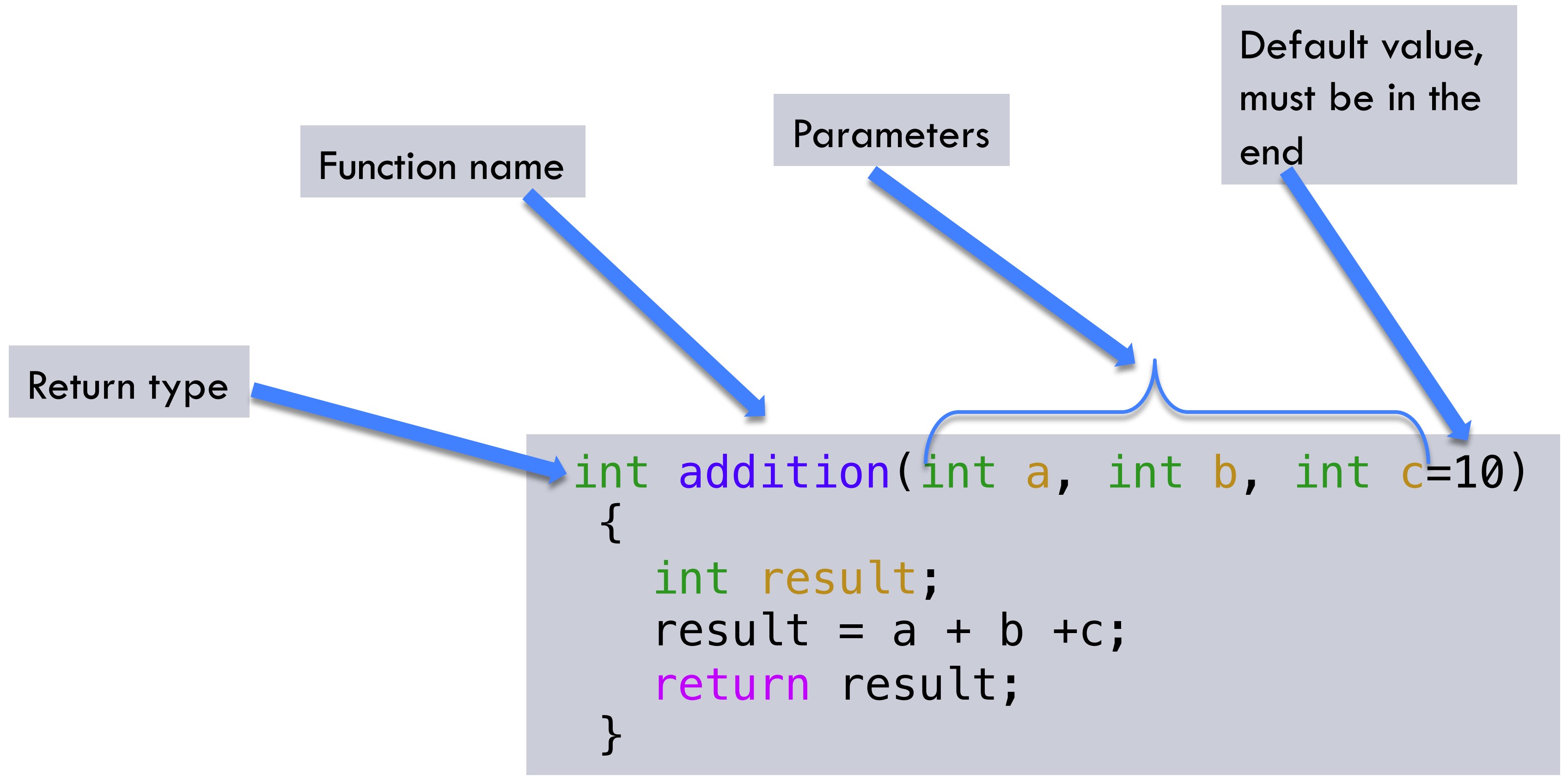
```
#include <iostream>

void func()
{
  std::cout << "Hello, pilot!" << "\n" ;
}

int main()
{
  func();
  return 0;
}
```

function: defined outside class

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

class Greeting
{
public:
  void func()
  {
    std::cout << "Hello, pilot!" << "\n" ;
  }
};
```

Method/class function: defined inside class

Functions



```
int addition(int a, int b, int c=10)
 {
   int result;
   result = a + b +c;
   return result;
 }
```

addition(1, 3, 34)
Or
addition(2, 5)

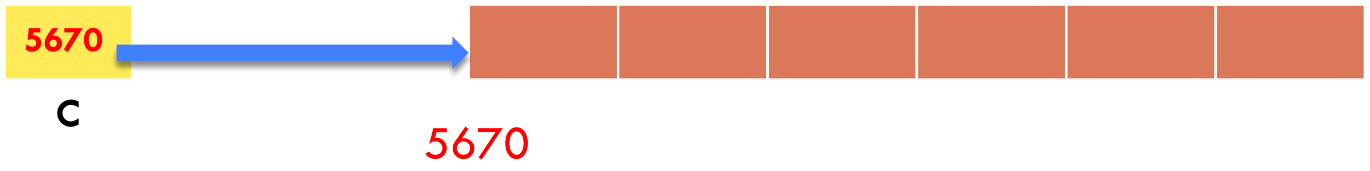
Functions

double cos(double)
double sin(double)
double log(double)
double pow(double, double)
double sqrt(double)
int abs(int)
double fabs(double)
double floor(double)
int rand()

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

int main ()
{
    double d = 30.74;

  cout << "sin(d) :" << sin(d) << endl;
  cout << "floor(d) :" << floor(d) << endl;
  cout << "pow( d, 2) :" << pow(d, 2) << endl;

  return 0;
}
```

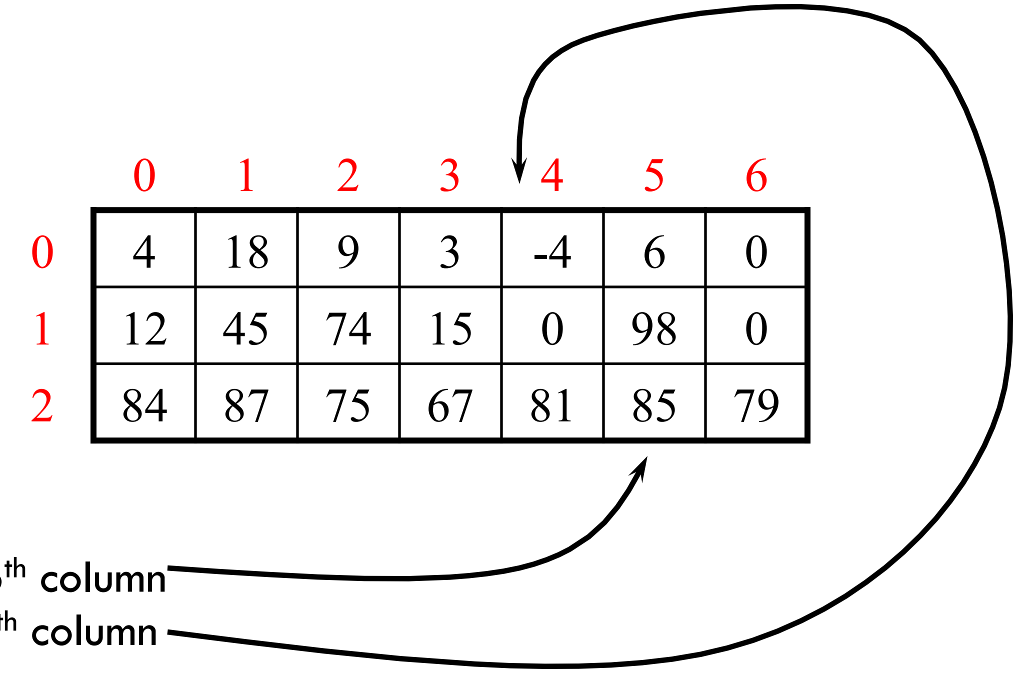
Arrays

```
double a[5] = {3.0, 4.0, 4.3, 6.0, 9.0};
double b[] = {1.0, 2,0, 3.0};
int *c = new int[6];

int m[10];
int n[10][20];
```



Multidimensional arrays

```
int rows = 3;
int columns = 7;
int array[rows][columns]
```



	0	1	2	3	4	5	6
0	4	18	9	3	-4	6	0
1	12	45	74	15	0	98	0
2	84	87	75	67	81	85	79

`array[2][5]`	3rd value in 6th column
`array[0][4]`	1st value in 5th column

Processing 2-d array

```
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
       array[i][j] = 0;
    }
}
```

Row-by-row processing

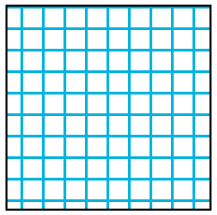
```
for (int j = 0; j < columns; j++) {
    for (int i = 0; i < rows; i++) {
       array[i][j] = 0;
    }
}
```

column-by-column processing

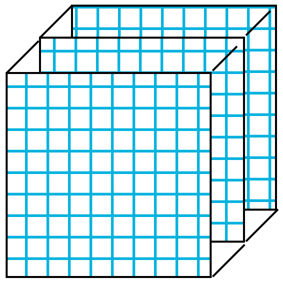
Array layout

An array can be declared with multiple dimensions.

2 Dimensional



3 Dimensional



```
double coord[100][100][100];
```

```
char test[3][2];
…     // initialization
```

Inside memory, the elements of array test are contiguously stored, row-by-row.

C	**E**	**F**	**A**	**B**	**I**
test[0][0]	test[0][1]	test[1][0]	test[1][1]	test[2][0]	test[2][1]

2-D array as parameter

- When passing a two-dimensional array as a parameter, the base address (starting address) of the array is passed.
- The two-dimensional array is stored in row-major order (row-by-row)
- The function must know the dimensions of the array, how?
    - The number of column must be specified

address of element (r, c) = base address of array
+ r*(number of elements in a row)*(size of an element)
+ c*(size of an element)

```
void init(int twoD[][col], const int row) {
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++)
      twoD[i][j] = -1;
  }
}
```

string type

- **C-type**
  - one-dimensional array of characters which is terminated by a null character '\0’
  - char greeting[6] = {'H', 'e', 'l', 'l', 'o', '**\0**'};
  - char greeting[] = "Hello";

C-type String

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

int main ()
{
   char str1[20] = "Hello, CSC230";
   char str2[20];
   // copy str1 into str2
   strcpy( str2, str1);
   cout << "strcpy( str2, str1) : " << str2 << endl;
   return 0;
}
```

cstring: defined in c

C++ type String

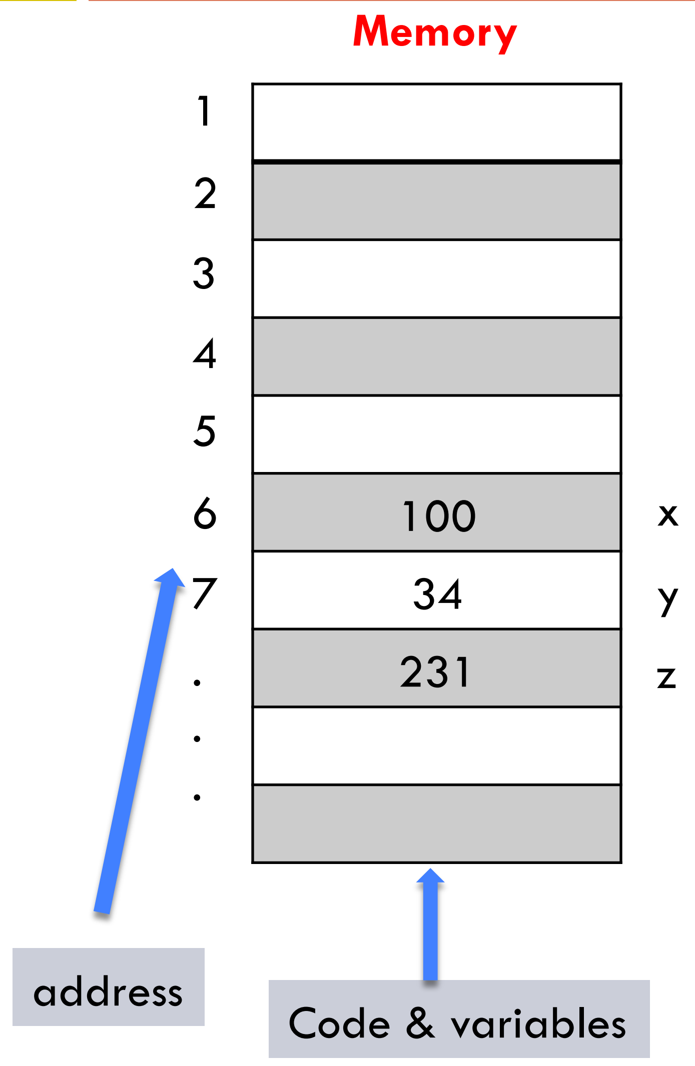
```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
  string str1 = "Hello, ";
  string str2 = ”CSC230";
  string str3;
  int  len ;
  // concatenates str1 and str2
  str3 = str1 + str2;
  cout << "str1 + str2 : " << str3 << endl;
  len = str3.size();
  cout << "str3.size() :  " << len << endl;

  return 0;
}
```

string: defined in c++

String functions

strcpy(s1, s2); Copy s2 to s1
strcat(s1, s2); s2 is appended to the end of s1
strlen(s1); return the length of s1
strcmp(s1, s2); if s1 = s2, return 0; if s1 < s2, return negative value; if s1 > s2, return positive value
strchr(s1, ch); return a pointer to the first ch in string s1
strstr(s1, s2); return a pointer to the first s2 in string s1

Pointers



x is an int variable.
Q: Where is x in memory ?

A: &x

Q: How to save x's memory address?

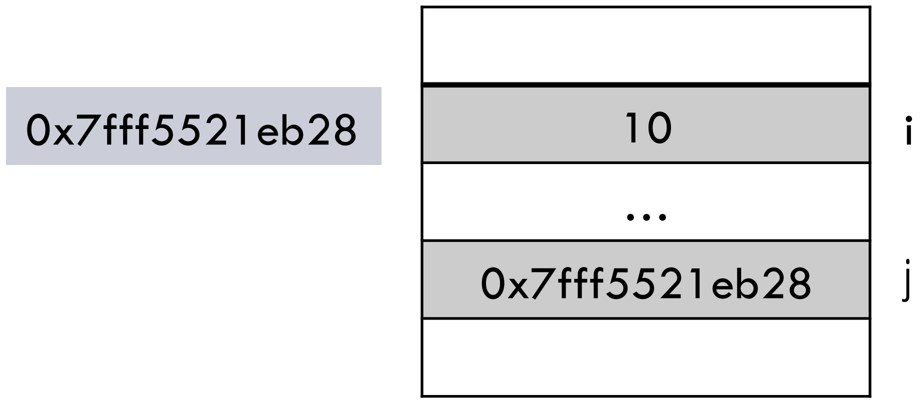
A: int *addr;
addr = &x;

A **pointer** is a variable that contains the address of another variable.

- addr is a pointer to int.
- &x is NOT a pointer, it is an address

Pointer example

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

int main ()
{

   int i = 10;
   int *j = &i;
   cout << i << "\t" << &i << "\t" << j << "\t" << *j << endl;

   return 0;
}
```



```
$ ./a.out
10      0x7fff5521eb28   0x7fff5521eb28   10
```

The unary operator * is the *indirection* or *dereferencing* operator; when applied to a pointer, it accesses the object the pointer points to.

More pointer examples

```
int x = 1, y = 2, z[10];
int *ip;              // ip is a pointer to int

ip = &x;              // ip now points to x
y = *ip;              // y is now 1
*ip = 0;              // x is now 0
ip = &z[0];           // ip now points to z[0]
```

Why pointer?

Want to use a function to swap two values.

```
void swap(int x, int y)
{
   int temp;
   temp = x;
   x = y;
   y =temp;
}
```

```
swap(a, b);
```

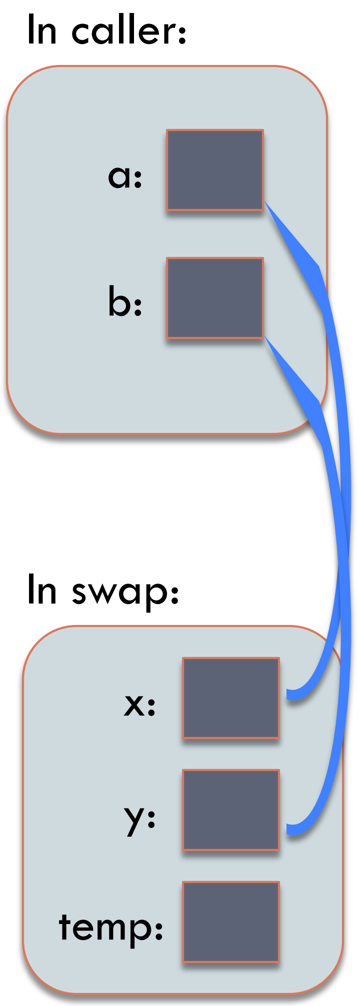
✗

```
void swap(int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y =temp;
}
```

```
swap(&a, &b);
```

✓

How it works inside?



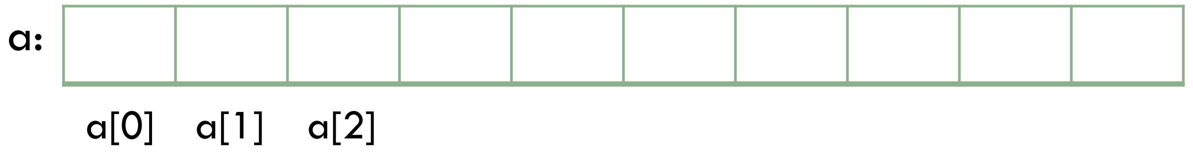
```
void swap(int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y =temp;
}
```

```
swap(&a, &b);
```

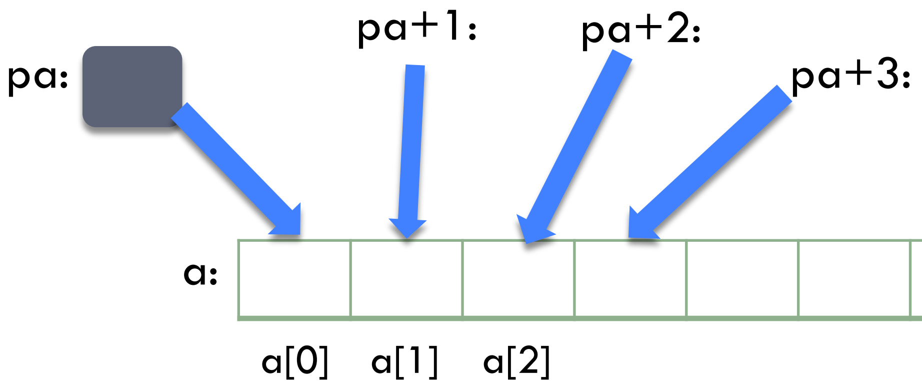
Pointers and Arrays

Pointer has a strong relationship with array.

```
int a[10];
```



```
int *pa;
pa = &a[0];
```



In fact, a[0] == a, variable a has the starting address of the whole array.
pa = a; and pa = a[0]; are equivalent.

a[i], *(a+i), pa[i], *(pa+i) are equivalent.

Pointers and Arrays Example

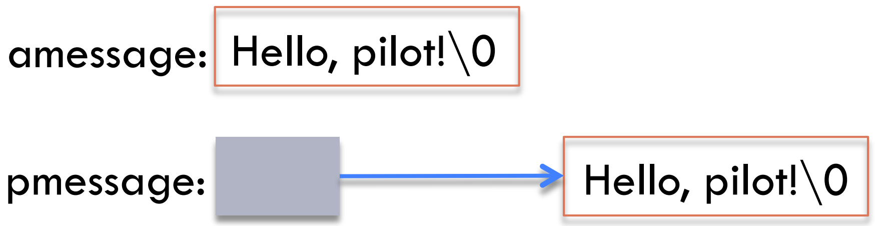
```
/* strlen: return length of string s */
int strlen(char *s)
{
  int n;
  for(n=0; *s != '\0'; s++)
    n++;
  return n;
}
```

Pointers and Arrays Example

```
/* strlen: return length of string s */
int strlen(char *s)
{
  char *p     = s;

  while (*p != '\0')
    p++;
  return p - s;
}
```

Characters Pointers

```
char amessage[]= "Hello, pilot!";
char *pmessage = "Hello, pilot!”;
```



Pointer to pointer

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

int main ()
{

   int i = 10;
   int *j = &i;
   int **k = &j;
   cout << &i << "\t" <<k << "\t" << **k << endl;

   return 0;
}
```

0x7fff5521eb28	10	i
	…	
0x7fff5521eb20	0x7fff5521eb28	j
0x7fff5521eb18	0x7fff5521eb20	k