Pointers sample Code in C++

Index: Link to sample code

Note: Sample code is enclosed inside multiline comment tag /* */

```
#include<iostream>
using namespace std;
void sample(int [], int);
void swap(int*, int*);
int fun(int a);
void passfun(int a, int (*fptr)(int));
int main()
{
	//Pointer basics
/*	int var;
	var = 5;

	int *ptr=NULL;
	ptr= &var; // '&' address of Operator

	cout<<var<<endl;
	cout<<ptr<<endl;
	cout<<&ptr<<endl;
	char c;
	c='C';
	char *ptrc;
	ptrc=&c;
	cout<<c<<endl;
	cout<<ptrc<<endl<<endl;
*/

//	dereferencing pointer
/*
	int var;
	var = 5;
	int *ptr;

	ptr= &var;
	cout<<var<<endl;
	cout<<*ptr<<endl;
	//'*' dereferencing operator refers the value of the variable pointed to by
	the pointer
	*ptr = 10;
	cout<<var<<endl;
	cout<<*ptr<<endl;
*/

//Constant Pointer and Pointer to constant
/*
	const int ca = 100, cb = 200; // constant variables
	int c = 300;
	int b=20;
	const int *ptr_ca = &ca; // pointer to a constant variable
	int *const cptr = &c; // constant pointer

	//*ptr_ca = 10; // Error
	//cptr = &b; // Error
	cout<<c<<endl;
	*cptr = 10; // Okay
	cout<<c;
	cout<<*ptr_ca<<endl;
	ptr_ca = &cb; // Okay

	cout<< *ptr_ca;
	return 0;
```

```
*/

//Pointer and Array
/*
	int a[5]={1,3,5,7,9};
	cout<<a<<endl;
	// name of the array <a> represents address of the first element of the
	//array
	cout<<&a[0]<<endl<<endl; // result will be equals to cout<<a;

	cout<<a[0]<<endl; //value of the first element of the array
	cout<<*a<<endl; //value of the first element of the array

	cout<<a+1<<endl;//Pointer arithmatic or arithmatic operation on address
	cout<<&a[1]<<endl;

	//a= a+1;  // error : array name a is equivalent to *const a  whose address
	//can not be changed

	int *p; //Pointer variable
	p =a;
	p++;  // p= p+1; // Pointer Arithmetic
	cout<<p<<endl; // address of a[1]

	cout<<*(a+1)<endl; // * operator has higher precedence over + operator
	cout<<a[1]<<endl;
*/

//Pointers with Functions
/*
	int a=4, b=7;
	cout<<"1";
	cout<<"a="<<a<<endl;
	cout<<"b="<<b<<endl;
	swap(&a,&b);  //definition
	cout<<"3";
	cout<<"a="<<a<<endl;
	cout<<"b="<<b<<endl;
*/

//Allocating and Deallocating Memory Dynamically
/*
	int *ptr = new int; // new operator to allocate memory
	*ptr =5;
	cout<<*ptr<<endl<<ptr<<endl;

	int *aptr= new int[3];// new operator to create dynamic array
	aptr[2]=4;
	cout<<aptr[2];

	delete ptr; // delete operator to deallocate memory
	delete [] aptr; // delete [] operator to deallocate dynamic array
	//cout<<*ptr;
	ptr =NULL; // set pointers to NULL after deallocating heap memory area
	aptr=NULL;
*/
```

```
// Function Pointers or Pointers to Function
//fun_ptr is a pointer to function fun()

      int (*fun_ptr)(int) = fun; // function prototype of fun() is compulsory

    /* The above line is equivalent of following two lines
       void (*fun_ptr)(int);
       fun_ptr = &fun;
    */

    // Invoking fun() using fun_ptr [Function Pointer]
    int b= (*fun_ptr)(10);
    cout<<b<<endl;

      //Function can be passed as parameter inside another function
       passfun(5, fun); // definition


//Common Error oriented scenarios in Pointers
/*    int *p;
      *p =8;
      cout<<*p; // dereferencing without initializing a pointer is an error
*/
//dangling pointer
      int *p, *q;
      int a =5;
      p=&a;
      q=p;
      cout<<*q;
      p=NULL;
      cout<<*q; // this will work

      int *p = new int;
      int *q = new int;
      *p =5;
      q=p;
      cout<<*q<<endl;
      delete p;
      cout<<*q; // q will become a dangling pointer

//redeclaration of pointer without reference it with new pointer variable or
deleting occupied memory
      int *p1 = new int;
      p1 = new int; //the address to old location is gone!

    return 0;

}
int fun(int a)
{
    return a*a;
}
void passfun(int a, int (*fptr)(int))
{
      cout<<a*(*fptr)(a);

}

void sample(int* aa, int size)
//void sample(int aa[5])
```

```
{
	for (int i=0 ; i<size;i++)
	cout<<*(aa+i)<<endl;
}
void swap(int *a, int *b)
{
	int c;
	c=*a;
	*a=*b;
	*b=c;
	cout<<"2";
	cout<<"a="<<*a<<endl;
	cout<<"b="<<*b<<endl;
}
```