CS 121 – 09

Midterm review (2)

Topics:

* Pointer
* Inheritance
* Recursive function

Pointer:

Create a pointer points to a variable:

int main()

{

int a = 9;

//create a pointer \*ptr which points to the variable a.

//print the value store in the address of a by the pointer you //create above

}

Create a pointer points to an array:

int main()

{

int a[5] = { 10, 20, 30, 40, 50 };

//create a pointer which points to the head of array a

//print the second element of the array by the pointer \*ptr

//create a for loop to print the whole array

}

Dynamic array:

Create a dynamic array:

int main()

{

//create a int type dynamic array with size 10;

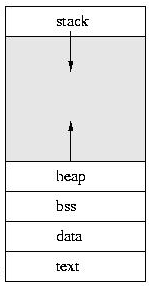
//assign value 100 inside every elements of the array by pointer \*arr

//create a loop to print out all elements in the array

//deallocate the memory of dynamic array arr

}

When a program is loaded into main memory, there is a basic structure like this:



Inheritance:

Here is the given class:

class Rectangle

{

public:

void setLength(int l)

{

length = l;

}

void setWidth(int w)

{

width = w;

}

int get\_area()

{

return length\*length;

}

protected:

int length;

int width;

};

Create a class called Cuboid which inherit class Rectangle (public inherit)

Class Cuboid should contain a private member “int hight” and two public

functions “setHight” which used to set the value of hight and “get\_volume”

which used to return the int volume of the cuboid

class Cuboid : public Rectangle

{

public:

private:

};

Now you have the class Rectangle and Cuboid, try to complete the task below:

int main()

{

//declare a Cuboid type variable called c1

//set the length of c1 = 10

//set the width of c1 = 20

//set the hight of c1 = 30

//print the volume of c1 by the public member function

//create a pointer which points to the address of c1 (pointer called ptr)

//using the pointer ptr to accesss the public member "setHight" of c1 to

change the hight to 3

//print the volume of c1 by public member function (access by the pointer)

}

Recursive function:

* Consider and define the base case ( usually use if statement )
* At least one parameter, the parameter must be changed when the function calls itself

Tracing the output of the following recursive functions:

int f(int n)

{

if (n == 1) return 1;

else return n + f(n - 1);

}

Calling statement: f(4)

Output: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void g(int n)

{

cout << n;

if (n > 0) g(n - 1);

}

Calling statement: g(4)

Output: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void h(int n)

{

if (n > 0) h(n - 1);

cout << n;

}

Calling statement: h(4)

Output: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void k(int n)

{

cout << n;

if (n > 0) k(n - 1);

cout << n;

}

Calling statement: k(4)

Output: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fibonacci:

int fib(int n)

{

if (n == 1 || n == 2) return 1;

else return fib(n - 1) + fib(n - 2);

}

Calling statement: fib(5)

Output: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Non-recursive of Fibonacci

int FIB(int n)

{

int fib1 = 1, fib2 = 1, fibn;

for (int i = 3; i <= n; i++)

{

fibn = fib1 + fib2;

fib1 = fib2;

fib2 = fibn;

}

return fibn;

}