University of Hertfordshire

7COM1025
Programming for Software Engineers
Lecture 21

Dr. Renato C. de Amorim

THE ONE HUNDRED UNDER FIFTY

RUN THE PROGRAM BELOW

```
#include <iostream>
using namespace std;
void stack_test(unsigned long s)
{
    long double x[s];
    cout<<"The size of a long double is: "<<sizeof(x[0])<< " bytes"<<endl;
    cout<<"The size of the whole array is "<<sizeof(x[0])*s/1024.0/1024.0<<" megabytes"<<endl;
    cout<<"Enter the number to initialise the array"<<endl;
    cin>>x[0];
    for (unsigned long i=1;i<s;i++)
        x[i] = x[0];
}
int main(){
    unsigned long x;
    cin>>x;
    stack_test(x);
return 0;
}
```

Did you run into potential problems?

DYNAMIC ALLOCATION

You can use dynamic allocation to store data as needed

It's allocated between your program (and its permanent storage area) and the stack (the heap).

Operators:

new and delete

var = new [data type];
delete var;
If the memory is full a bad_alloc exception will be thrown (defined in <new>)
Very old compilers may simply return a null-pointer.

DYNAMIC ALLOCATION

```
#include <iostream>
#include <new>
using namespace std;
int main(){
    int *p;
    try{
        p = new int;
    } catch (bad_alloc ex){
        cout<<"Allocation failure."<<endl;
        return 1;}
    *p=100;
    cout<<"At "<<p<<" value "<<*p<<endl;
    delete p;
    return 0;
}
```

```
#include <iostream>
#include <new>
using namespace std;
int main(){
    int *p;
    try{
        p = new int(30);
    } catch (bad_alloc ex){
        cout<<"Allocation failure."<<endl;
        return 1;}
cout<<"At "<<p<<" value "<<*p<<endl;
     delete p;
     return 0;
}
```

ALLOCATING ARRAYS

```
#include <iostream>
#include <new>
using namespace std;
int main(){
    int *p;
    try{
        p=new int[10];
    } catch (bad_alloc ex){
        cout<<"Allocation failure"<<endl;
        return 1;}
    for (int i=0; i<10;i++){
        p[i]=i;
        cout<<p[i]<<' '<<endl;
    }
    delete []p;
    return 0;
}
```

No array initialised by new can have an initializer.

PROBLEM 21.1

Fix the program in our first slide so that it can allocate a higher number of integers.

ALLOCATING OBJECTS

```
#include <iostream>
#include <new>
using namespace std;
class Rectangle{
int width;
int height;
public:
    Rectangle(int w, int h){
    width=w; height=h;
    cout<<"Constructing "<<w<<" by "<<h<<endl;}
    ~Rectangle(){
        cout<<"Destructing "<<width<<" by "<<height<<endl;}
    int area(){
    return width*height;}
};
```

```
int main()
{
    Rectangle *p;
    try{
        p = new Rectangle(10,8);
    }catch(bad_alloc ex){
        cout<<"Bad allocation."<<endl;
        return 1;}
    cout<<"Area: "<<p->area()<<endl;
    delete p;
    return 0;
}
```

ALLOCATING ARRAYS OF OBJECTS

```
#include <iostream>
#include <new>
using namespace std;
class Rectangle{
int width;
int height;
public:
    Rectangle(int w, int h){
    width=w; height=h;
    cout<<"Constructing "<<w<<" by "<<h<<endl;
    }
    Rectangle(){
    width=height=0;
    cout<<"Constructing "<<width<<" by "<<height<<endl;
    }
    void set(int w, int h){
        width=w; height=h;}
    ~Rectangle(){
        cout<<"Destruct. "<<width<<" by "<<height<<endl;
    }
    int area(){
    return width*height;}
};
```

```
int main()
{
    Rectangle *p;
    try{
        p = new Rectangle[3];
    }catch(bad_alloc ex){
        cout<<"Bad allocation."<<endl;
        return 1;}
    p[0].set(3,4);
    p[1].set(10,8);
    p[2].set(5,6);
    for (int i=0; i<3;i++)
        cout<<"Area: "<<p[i].area()<<endl;
    delete []p;
    return 0;
}
```

WHEN TO USE NEW

When you want an object to be in the memory until you use delete.

- Instead of being destroyed only at the end the block.
- A function may even return a pointer to an object created inside it (if it was created with new and it wasn’t deleted)

When the size of an object is only known at run-time

- Think of a large array whose size is determined by the user.

You must use delete in any case

- Even if there is an exception.
- Even if the code block is finished.

PROBLEM 21.2

Remember your sorting algorithm (bubble sort).
Re-write it using new and delete.

NAMESPACES

```
#include <iostream>
using namespace std;
namespace CounterNameSpace{
    int upperbound;
    int lowerbound;
    class Counter{
        int count;
    public:
        Counter (int n){
            n<=upperbound? count=n:count=upperbound;}
        void reset(int n){
            if (n<=upperbound) count=n;}
        int run(){
            if (count>lowerbound) return count--;
            else return lowerbound;
        }
    };
}
```

```
int main(){
int i;
CounterNameSpace::upperbound=100;
CounterNameSpace::lowerbound=0;
CounterNameSpace::Counter ob(10);
do{
    i=ob.run();
    cout<<i<<' ';
} while
(i>CounterNameSpace::lowerbound);
cout<<endl;
return 0;
}
```

NAMESPACES (USING)

```
int main(){
    using namespace CounterNameSpace;
    int i;
    upperbound=100;
    lowerbound=0;
    Counter ob(10);
    do{
        i=ob.run();
        cout<<i<<' ';
    } while (i>lowerbound);
    cout<<endl;
    return 0;
}
```

PROBLEM 21.3

You need to create a class to hold data about students. This data should be: Name, address, names of modules enrolled, at least one mark per module.

Note that:
There may be a large number of students. In extreme cases a student may have enrolled in many modules over the years. Also, write a main function that allows people to state how many students there are and input data for each of the students.