**Array** is pointer variable. However, pointers can move but not the array.

Int arr[5] is as int \* const arr;

Void changeArray(int arr[], int n);//P-by ref

Void checkArray(cons tint arr[],n); //P-by ref

**Array Example**

int var[MAX] = {10, 100, 200};

//MAX has to be an const int!

int \*ptr[MAX]; //array of pointer

for (int i=0; var[i]!=0; i++)

{ ptr[i] = &var[i]; //assign the address

Cout<< \*(ptr+i) << “ ”; }

Class student;

Student s[10]; //will call class Student **default** constructor for ten times. It won’t compile without a default constructor!

--------------------------------------------

Assert (**j[0] == \*(j + 0)**);

Assert (**(\*j)[0] == j[0][0] == \*(\*(j + 0) + 0)**)**;**

**Pointers** must be initialized before used

**Pointer Example**

int main(){

char msg[50] = "She'll be a massless princess.";

removeS(msg);

cout << msg; /\*prints: he'll be a male prince.\*/}

**void removeS(char\* str){**

**char\* k = str;**

**while (\*k != 0){**

**while (\*k =='s'||\*k=='S')**

**k++;** //move to next not-S

**\*str = \*k;**

**str++;**

**k++; }**

**\*str =0;}**

**Cstring**

**strcpy(s1, s2)**; //Copies string s2 into s1

**strcat(s1, s2);** //Concatenates string s2 onto the end of s1

**strlen(s1);** //Returns the length of string s1.

**strcmp(s1, s2);**

//Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2

**strchr(s1, ch);** //Returns a pointer to the first occurrence of character ch in string s1.

**strstr(s1, s2);**  //Returns a pointer to the first occurrence of string s2 in string s1.

**C++string**

// copy str1 into str3

**str3 = str1;**

cout << "str3 : " << str3 << endl;

// concatenates str1 and str2

**str3 = str1 + str2;**

cout << "str1 + str2 : " << str3 << endl;

// total lenghth of str3 after concatenation

len = **str3.size()**;

len2 = **str3.length();**

cout<<len<<” == ”<<len2<<endl;

Pointers can be initialized as nullptr;

You cannot have NULL **References.**

**Constant**

**const** type m; //do not allow to change m

**const** char \*p1; //do not allow to change \*p1

char **const** \* pContent == char const (\* pContent)

//the same, const can be before type or after

int **const** \* **const** p1,p2

// 1st const (\*p1)&p2; 2ed const p1

// const before \*, const the \*p1

// const after \*, const the p1 pointer

Void foo(**const** int& i ); //const reference

Int getFoo() **const**; //read only function

**Structure**

// passing value of a structure in three ways

#include <iostream>

#include <string>

using namespace std;

**struct** Student{

int num;

char name[20];

float score;}**;**

int main ()

{ void print1(**Student**);

void print2(**Student\***);

void print3(**Student&**);

Student stu= {12345,"Chou",98.5}; Student \* s = &stu;

print1(stu);

print2(s);

print3(stu);

return 0;}

void print1(Student stu){

cout << **stu.name** << <<endl;}

void print2(Student \*p){

cout << **p -> name** <<endl;

cout << (\*p).name<<endl;}

void print3(Student &stu){

cout << **stu.name** <<endl;} //same output

**Enumeration**

//A class can declare an enumeration in its public or private area.

Enum Color{ //look as if a struct

Red, //value is 0

Blue=3,

White}**;** //value is previous+1, so 4

std::ostream& operator<<(std::ostream& os, Color c)

{ switch(c)

{ case(Red): os<<”red”; break;

Case(Blue): os<<”blue”; break;

Default: os<<”white”; break;}

Return os;}

Int main(){

Color m\_carColor = Red;

If(m\_carColor == White)

double price=50.0/White; cout<<”color: ”<<m\_carColor<<endl;}

**Allocate? Heap or Stack?**

Student s(“Sam”,1044321); //on stack

Student \*ptr; //not allocated

Ptr = &s; //mot allocated

Ptr = new Student(“Sally”, 2034123);

//on heap

Student arrayS[5]; //on stack

**Class-constructor and destructor**

class Cat

{

public:

//constructors

Cat(){};

Cat(string name, int age):m\_age(age)

{m\_name=**new string(name)**;};

//copy constructor

Cat(const Cat& copyCat)

{m\_age = copyCat.getAge();

m\_name = **new string(copyCat.getName())**;};

//destructors

~Cat(){**delete m\_name**;};

//assginment operator

Cat& operator=(const Cat& copyCat)

{this->changeName(copyCat.getName());

this->changeAge(copyCat.getAge());

return \*this;};

//operator overload

friend std::ostream& operator<< (std::ostream& out,const Cat& c);

//accessor

int getAge() const{return m\_age;};

string getName() const{return \*m\_name;};

string getSound() const{return m\_sound;};

//mutator

void changeAge(const int n){m\_age=n;}

void changeName(const string& nickName)

{m\_name= new string(nickName);};

void changeSound(const string& sound)

{m\_sound= sound;};

private:

int m\_age;

string m\_sound;

**string \* m\_name**;

};

std::ostream& operator<<(std::ostream& out,const Cat& c)

{

out<<c.getSound();

return out;

};

int main()

{ Cat c;

Cat d("Debby",3);

c.changeName("Carry");

c.changeAge(2);

cout<<"I have two cats, "<<c.getName() <<" and " << d.getName()<<endl;

cout<<"They are "<<c.getAge()<<" and "<<d.getAge()<<" year old"<<endl;

Cat cc(c);

Cat dd = d;

cout<<"I love " << cc.getName()<<" and "<< dd.getName()<<endl;

c.changeSound("Meow!");

d.changeSound("Meow~Meow~");

cout<<c<<endl;

cout<<d<<endl;

}

**Class-parent,child,ancestor,descendant**

Parent: Base class

Child: Derived class

Ancestor: parent and their parent

Descendant: children and their children

**Class-protected/private**

Private: child cannot access, but friends can

Protected: public to Base class, friend class, their derived classes; private to others

Inheritance: “Is-A” relationship

Aggregation: “Has-A” relationship

Ie. A nerd student is a student; a student has several books.

Always invoke a base constructor!

Cat():Pet(“no name”,0){};

Pet\* ptr=new Cat();

//baseClass= derivedClass, not the other way

//virtual member functions decided at run time

**Regular Expression Examples**

[aeiou]: matches a single vowel in the text

[^aeiou]: matches a character Not in the text

[a-z]: use hyphen to match contiguous ranges

(n|s|e|w): matches one of the letters in the set

b.d: matches b3d,bed,bid,b{d…etc

+: one or more

\*: zero or more

{#}: exactly # matches

{#,}: at least # matches

{n,m}: at least n, no more m matches

Example:

[1-9][0-9]\* is an expression for a positive int

[a-zA-Z\_][a-zA-Z\_0-9]\* is the expression for a C++ variable name

[1-9][0-9]\*[.][0-9]\* is the expression for a non-zero float number

[+\-]\*[1-9][0-9]?[0-9]? (ne|nw|se|sw|n|s|e|w)

**Include Guard**

#ifndef SOMEFILE\_H

#define SOMEFILE\_H

//…

#endif //SOMEFILE\_H

#include <iostream> performs "textual substitution"

Using namespace std; //no need std::stirng