“data members” “fields” “instance variables” “attributes”

struct Employee

{

string name;

double salary;

int age;

}; //DON’T FORGET THE SEMICOLON!!!

int main()

{

Employee e1;

Employee e2;

e1.name = “Ray”;

e1.(name/salary/age)

e1.age = 35;

e1.age++; (it is a double!) – same types

cout << “Enter a name: “;

getline(cin, e2.name);

Employee company[];

company[1].name = “Ethel”;

for (int k = 0; k < company[1].name.size(); k++)

cout << company[1].name[k] << endl;

}

e1

* Name: “Ray”
* Salary: ?
* Age: ?

e2

* Name: “”
* Salary: ?
* Age: ?

an object of some struct type . the name of a member of that struct type

struct Employee

{

string name;

double salary;

int age;

};

void printPaycheck (const Employee& e) //pass by value (copy the object)

{

cout << “Pay to the order of “ << e.name << “ the same of $” << e.salary/12 << endl;

}

void celebrateBirthday(Employee\* e)

{

\*e.age++; //WRONG \*(e.age++)

Correct: (\*e).age++ -> use of hyphen greater than (->

New:

e -> age++; //more readable -> no overload

}

double doublePayroll (const Employee emp[], int n)

{

double total = 0;

for (int k = 0; k < n; k++)

total += emp[k].salary;

return total;

}

int main ()

{

Employee company [100];

int nEmployees = 0;

…

printPaycheck(company[1]);

celebrateBirthday(&company[1]);

for (int k = 0; k < nEmployees; ++k)

{

cout << company[k].name << endl;

}

for (Employee\* e = company; ep < company + nEmployees; ++e)

{

cout << e->name << endl;

}

cout << totalPayroll(company, nEmployees) << endl;

}

void f(blah b); //f will not change – use if it’s cheap to copy (for small data size)

void f(const blah& b); //f will not change – use for large data ~ do not create copies

To change: void f(blah& b)

“data member” “fields” “attributes” “instance variables”

“member functions” “operations” “methods”

struct target

{

int pos;

string history;

//history consists of only Rs and Ls

//pos = # of Rs - # of Ls -> **invariant**: always true for an object

void init();

void move(char dir);

void replayHistory();

void moveRight(t);

};

void target::init() //defining the function of a data structure

{

//auto passing of the pointer of the object

//this – name of the pointer of t

this->pos = 0;

this-> history = “”;

}

void target::move(char dir)

{

if(toupper(dir) == ‘R’)

{

this.pos++;

history += ‘R’;

}

else if if(toupper(dir) == ‘L’)

{

this.pos--;

history += ‘L’;

}

else

{

}

}

int main()

{

target t;

t.init();

t.move(‘R’);

t.move(‘r’);

for (int k = 0; k < 4; k++)

{

t.move(‘R’);

}

t.move(‘R’);

}

11/20/19

struct Target

{

}