CSE 165/ENGR 140 Intro to Object Orient Program

Lecture 5 – C in C++

Types: casting

- Types can be converted by C-like type-casts in parenthesis.

```
//: C03:FunctionCallCast.cpp
int main() {
  float a = float(200);
  // This is equivalent to:
  float b = (float)200;
}
```

- In C++ we should use:
  - **static_cast**: simple casts for type conversion etc
  - **const_cast**: to remove const
  - **reinterpret_cast**: to cast an object to something completely different (only justified in rare circumstances)
  - **dynamic_cast**: type-safe cast with **run-time checking**, can be used only with pointers and references to objects

Data types: creation of new types

- We can define our own data types based on existing data types
  - ***typedef*** *existing_type new_typename*;

```
typedef float coordinate;
typedef char * pointer_char;
typedef double dimension[3];
pointer_char p_char; // instead of char * p_char;
coordinate x;  // instead of float x;
coordinate y;  // instead of float y;
dimension a, b, c, d;  // instead of double a[3], b[3], …
```

Data types: structure

- Structure is a group of elements under one name
- Elements can be of different data type
- Data elements are called “members”
- Structure is defined for the rest of the program

```
struct structure_name
{
    member_type_1  member_name_1;
    member_type_2  member_name_2;
    …
    member_type_n  member_name_n;
} object_names;
```

Structure declaration

```
struct vehicle
{
    string make;
    string model;
    int year;
} car, truck, bike;
```

```
struct fruit
{
    double weight;
    float price;
    bool ripe;
};
fruit apple;
fruit banana, pear;
```

Member access in structure

- We can refer to
  - Whole object: *object_name*
  - Each member: *object_name.member_name*
- Examples
  - bike (vehicle)
  - car.model (string)
  - car.year (int)
  - apple.weight (double)
  - pear (fruit)
  - banana.ripe (bool)

```
struct vehicle
{
     string make;
     string model;
     int year;
} car, truck, bike;
```

```
struct fruit
{
     double weight;
     float price;
     bool ripe;
};
fruit apple;
fruit banana, pear;
```

Structure assignment

- To assign the whole object
  - fruit apple;
  - apple = { 0.22222, 1.75, false };
  - fruit peach = { 2./3., 2.50, true };
- To assign individual members
  - vehicle car;
  - car.make = “Acura”;
  - car.model = “NSX”;
  - car.year = 2017;

```
struct fruit
{
     double weight;
     float price;
     bool ripe;
};
```

```
struct vehicle
{
     string make;
     string model;
     int year;
};
```

Pointers to structures

- Accessing from pointers
  - variable.member
  - pointer->member
- Declaration
  - vehicle car;
  - vehicle * p_car;
  - p_car = &car;

Pointers to structures

- To access members using a pointer
  - (*pointer).member (*p_car).model = “NSX”;
  - pointer->member p_car->model = “NSX”;

```
p_car->make = “Fiat”;
(*p_car).model = “500”;
fruit * p_apple = &apple;
fruit * p_peach = &peach;
p_apple->weight = p_peach->weight;
(*p_apple).weight = p_peach->weight;
(*p_apple).weight = (*p_peach).weight;
```

Pointers to structures

- (*pointer).member ≠ *pointer.member

- *object.p_member = *(object.p_member)
  - The member of an object is a pointer
  - See operator precedence: http://www.cplusplus.com/doc/tutorial/operators/

Pointers to structures example

```
//: C03:SimpleStruct3.cpp
// Using pointers to structs
typedef struct Structure3 {  // skip typedef in C++
  char c;
  int i;
  float f;
  double d;
};

int main() {
  Structure3 s1;
  Structure3* sp = &s1;
  sp->c = 'a';
  sp->i = 1;
  sp->f = 3.14;
  sp->d = 0.00093;
}
```

Data types: enumerations

- Allows us to create new data types with predefined values.

```
enum Mood { HAPPY, SLEEPY, SAD, ANGRY }; //0,1,2,3
Mood myMood = SLEEPY;

enum Color { RED=1, GREEN=10, BLU=100 };
Color myColor;
myColor = BLU;
```

Data types: enumerations

```
//: C03:Enum.cpp
   // Keeping track of shapes

   enum ShapeType {
     circle,
     square,
     rectangle
   };  // Must end with a semicolon like a struct

   int main() {
     ShapeType shape = circle;
     // Activities here....
     // Now do something based on what the shape is:
     switch (shape) {
       case circle:  /* circle stuff */ break;
       case square:  /* square stuff */ break;
       case rectangle:  /* rectangle stuff */ break;
     }
   }
```

Data types: enumerations

- Enumerators offer type checking and allow for more intuitive types than ints

```
enum ShapeType { circle=10, square=20, rectangle=50 };

enum snap { crackle=25, pop=10 };

void draw ( ShapeType t )
{
    // call drawing commands according to type here
}

void main ()
{
    snap s = pop;
    draw ( s ); // will generate warning or error
}
```

Wake up

- https://youtu.be/BGLsUxe2pik

Data types: unions

- Unions allow one location of memory to be accessed as different data types
- The size of a union is the largest of its members
- Modification of one of the members will affect the value of all members
- It is not possible to store different values for individual members

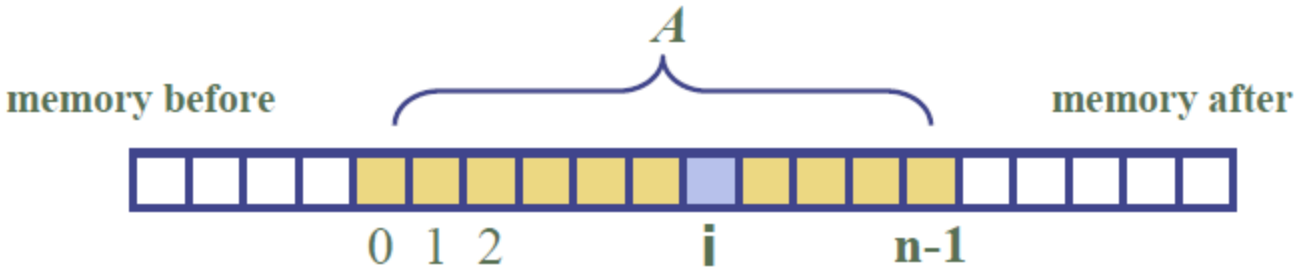
Data types: unions

```
//: C03:Union.cpp
#include <iostream>
using namespace std;

union Packed { // Declaration similar to a struct or class
  char i;      // The union will be the size of a
  short j;     // double, since that's the largest element
  int k;
  long l;
  float f;
  double d;
};

int main() {
  cout << "sizeof(Packed) = " << sizeof(Packed) << endl;
  Packed x;
  x.i = 'c';
  cout << x.i << endl;
  x.d = 3.14159;
  cout << x.d << endl;
}
```

Data types: array

- **Array**: collection of elements of the same type
- **Array size**: how many elements—integer number **n**

```
int a[10];          // a[0], ….. a[9]

int n=10;
int A[n];           // A[0], ….. A[n-1]
```



Array—Types and initialization

```
char c[4];

short s[5];

int y[2];

long l[7];

float f[10000];

double x[n];

bool b[2];
```

```
char c[] = { 'a', 'b', 'c', 'd' };

int a[5] = { 4, 3, 2, 1, 0 };

int n = 10;

double x[n];

int i = 3;

x[0] = 2.1;

x[i] = .1e5;

x[4] = i / 2;

x[a[i]] = a[i];

x[i] = i;

bool b[2] = {false, true};
```

Data types: arrays

```
//: C03:Arrays.cpp
#include <iostream>
using namespace std;

int main() {
  int a[10];
  for(int i = 0; i < 10; i++) {
    a[i] = i * 10;
    cout << "a[" << i << "] = " << a[i] << endl;
  }
}
```

Data types: arrays

```
#include <iostream>
using namespace std;

struct Point3D
{
  int x, y, z;
};

int main()
{
  Point3D p[10];
  for(int i = 0; i < 10; i++)
  {
    p[i].x = i + 1;
    p[i].y = i + 2;
    p[i].z = i + 3;
    cout << p[i].x << ", " << p[i].y << ", "<< p[i].z << endl;
  }
}
```

Pointers and arrays

- Identifier of an array equivalent to the address of array's first element
- The i-th element of ***a*** can be dereferenced either way:
  - a[i]
  - *(a+i)
- Example:
  - int a[20]; // a is a pointer to a[0]
  - a[5] = 0; // a [offset of 5] = 0
  - *(a+5) = 0; // pointed by (a+5) = 0

Pointer and arrays

- Examples:
  - int numbers[20];
  - int * p;
  - p = numbers;

  - double d[] = {0., 1., 2.} ;
  - double* p_var = d; //points to array d
  - double* p_array[3]; //array of 3 pointers
  - p_array[0] = &d[0]; //1st pointer is the address of d[0]

Pointer and arrays

```
//: C03:PointersAndBrackets.cpp (extended)
int main() {
  int a[10];

  int* ip = a; // get pointer to beginning of a
  for (int i = 0; i < 10; i++)
    ip[i] = i * 100; // access positions from pointer location

  ip = &a[0]; // another way of getting pointer to beginning of a
  for (int i = 0; i < 10; i++)
    *ip++ = i * 100; // here the pointer is incremented each time

  for (int i = 0; i < 10; i++)
    cout<<a[i]<<" ";
  cout<<endl;
}
```