정보통신프로그래밍

ICT Programming

Structure

임 민 중

동국대학교 정보통신공학과

Adding Two Distances

Structure is a user defined data type that allows to combine data items of different kinds. This program adds two distances in inch-feet system (1 foot = 12 inches).

```
#include <stdio.h>
struct Distance {
   int feet, inch;
}; // 8 bytes
typedef struct Distance Distance;
Distance Add(Distance d1, Distance d2);
void Print(char *s, Distance d);
```

feet
inch

```
typedef struct Distance {
   int feet, inch;
} Distance;
```

```
typedef struct {
   int feet, inch;
} Distance;
```

may be OK for some compilers

```
int main()
{
   // without typedef -> struct Distance d1, d2;
   Distance d1 = {5, 9}, d2;
   d2 = (Distance) {8, 8};
   // d2.feet = 8; d2.inch = 8;
   Print("Sum of distance = ", Add(d1, d2));
   return 0;
}
```

Sum of distance = 14' 5"

```
Distance Add(Distance d1, Distance d2)
{ // Add two distances
   Distance sum;
   sum.feet = d1.feet + d2.feet;
   sum.inch = d1.inch + d2.inch;
   if (sum.inch > 12) {
      sum.inch = sum.inch - 12;
      sum.feet++;
   }
   return sum;
}
```

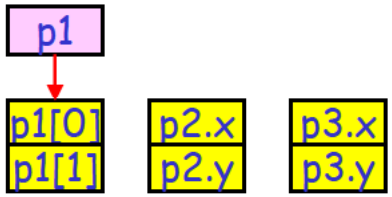
```
sum = (Distance) {d1.feet + d2.feet,
                  d1.inch + d2.inch};
sum.feet += sum.inch / 12;
sum.inch %= 12;
```

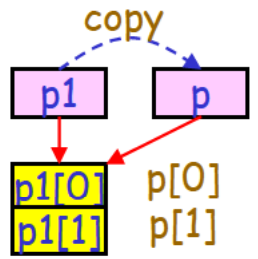
```
void Print(char *s, Distance d)
{ // Print distance
   printf("%s%d' %d\"\n", s, d.feet, d.inch);
}
```

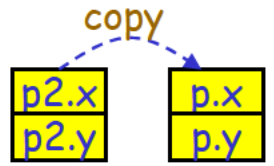
Array vs. Structure

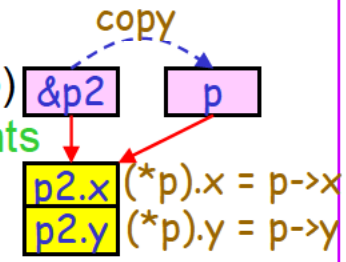
This program illustrates the difference between array and structure.

```
#include <stdio.h>
typedef struct Point {
   int x, y;
} Point; // 8 bytes
void IncrementArray(int p[2]);
void IncrementStruct1(Point p);
void IncrementStruct2(Point* p);

int main()
{
   int p1[2] = {0, 0}; // array
   Point p2 = {0, 0}, p3 = {10, 10}; // structure
   IncrementArray(p1);      // call by address
   printf("p1 = {%d, %d}; ", p1[0], p1[1]);
   IncrementStruct1(p2);   // call by value
   printf("p2 = {%d, %d}; ", p2.x, p2.y);
   IncrementStruct2(&p2); // call by address
   printf("p2 = {%d, %d}\n", p2.x, p2.y);
   p2 = p3;                    // not allowed for array
   printf("p2 = {%d, %d}; ", p2.x, p2.y);
   p2 = (Point) {20, 20}; // not allowed for array
   printf("p2 = {%d, %d}\n", p2.x, p2.y);
   return 0;
}

void IncrementArray(int p[2])
{ // Increment array elements
   p[0]++; p[1]++;
}

void IncrementStruct1(Point p)
{ // Increment copy of structure
   p.x++; p.y++;
} // p is incremented but p2 is not incremented

void IncrementStruct2(Point* p)
{ // Increment structure elements
   p->x++; p->y++;
} // (*p).x++; (*p).y++;
```





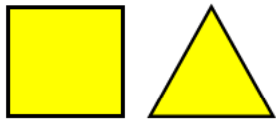




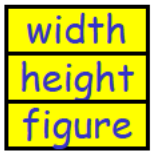

```
p1 = {1, 1}; p2 = {0, 0}; p2 = {1, 1}
p2 = {10, 10}; p2 = {20, 20}
```

Area of Rectangle and Triangle

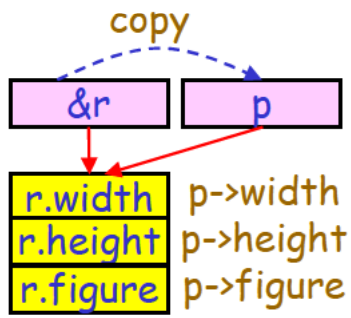
This program calculates the area of a polygon using 'enum'.



```
#include <stdio.h>
typedef enum Figure {
   Rectangle, Triangle
} Figure;
typedef struct Polygon {
   float width;          // 4 bytes
   float height;         // 4 bytes
   Figure figure;        // 4 bytes
} Polygon;               // 12 bytes
void Set(Polygon* p, float w, float h, Figure f);
float Area(Polygon p);

int main()
{
   Polygon r, t;
   Set(&r, 4, 5, Rectangle);
   Set(&t, 4, 5, Triangle);
   printf("Area of Rectangle is %.0f\n", Area(r));
   printf("Area of Triangle is %.0f\n", Area(t));
   return 0;
}
```




```
void Set(Polygon* p, float w, float h, Figure f)
{ // Set polygon
   *p = (Polygon) {w, h, f};
}
```




```
   p->width = w;
   p->height = h;
   p->figure = f;

float Area(Polygon p)
{ // Calculate area
   switch (p.figure) {
      case Rectangle: return p.width * p.height;
      case Triangle: return p.width * p.height / 2;
      default: printf("Illegal figure\n"); return 0;
   }
}
```

```
r = (Polygon) {4, 5, Rectangle};
t = (Polygon) {4, 5, Triangle};
```

```
Area of Rectangle is 20
Area of Triangle is 10
```

Sum, Mean, and Variance

This program calculates the sum, mean, and variance of elements in an array.

```
#include <stdio.h>
#define SIZE 6        // size of array
typedef struct Result {
   float sum, mean, variance;
} Result;
Result Statistics(int a[]);

int main()
{
   int array[SIZE] = {1, 2, 3, 4, 5, 6};
   Result result;
   result = Statistics(array);
   printf("Sum = %.2f\n", result.sum);
   printf("Mean = %.2f\n", result.mean);
   printf("Variance = %.2f\n", result.variance);
   return 0;
}
```

sum
mean
variance

void Statistics(Result* r, int a[]);

Statistics(&result, array);

```
Sum = 21.00
Mean = 3.50
Variance = 2.92
```

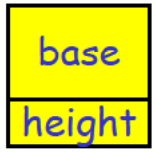
```
Result Statistics(int a[])
{ // Return structure
   float sum, mean, var, sum2, diff;
   int i;
   for (sum = 0, i = 0; i < SIZE; i++) {
      sum += (float) a[i];
   }
   mean = sum / SIZE;
   for (sum2 = 0, i = 0; i < SIZE; i++) {
      diff = (float) a[i] - mean;
      sum2 += diff * diff;
   }
   var = sum2 / SIZE;
   return (Result) {sum, mean, var};
}
```

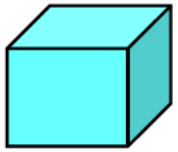
```
void Statistics(Result* r, int a[])
{ // Update using a pointer

   *r = (Result) {sum, mean, var};
}
```

Volume of Box

This program calculates volume of a box using structure of structure.

```
#include <stdio.h>
#include <stdlib.h>
#define EXIT { printf("Illegal value"); exit(-1); }
typedef struct Rectangle {
   int length, depth;
} Rectangle;
Rectangle SetRectangle(int l, int d);
typedef struct Box {
   Rectangle base;
   int height;
} Box;
Box* NewBox(int l, int d, int h);
int Volume(Box box);

int main()
{
   Box* box = NewBox(2, 4, 5);
   printf("Volume = %d\n", Volume(*box));
   return 0;
}
```





```
Volume = 40
```

```
Rectangle SetRectangle(int l, int d)
{ // Set rectangle
   if (l <= 0 || d <= 0) EXIT;
   return (Rectangle) {l, d};
}

Box* NewBox(int l, int d, int h)
{ // Construct a new box
   Box* box = (Box*) malloc(sizeof(Box));
   box->base = SetRectangle(l, d);
   if (h <= 0) EXIT;
   box->height = h;
   return box;
}

int Volume(Box box)
{ // Calculate volume of box
   return box.base.length * box.base.depth
          * box.height;
}
```

Student Information - 1

This program stores and prints student information.

```
#include <stdio.h>
#include <string.h>  // library for string
#define SIZE 32
typedef struct Student {
   char name[SIZE];              // 32 bytes
   int id;                       // 4 bytes
} Student;                       // 36 bytes
void Store(Student* s, char* name, int id);
void Print(Student s);

int main()
{
   Student s;
   Store(&s, "Kim", 1234);  // call by address
   Print(s);                // call by value
   return 0;
}
```

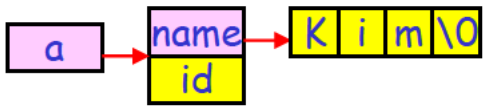
```
void Store(Student* s, char* name, int id)
{ // Store student information
   strcpy(s->name, name);  // string copy
   s->id = id;                    // (*s).id = id;
}

void Print(Student s)
{ // Print student information
   printf("Student: name = %s; id = %d\n",
          s.name, s.id);
}
```

```
Student: name = Kim; id = 1234
```

Student Information - 2

This program illustrates memory allocation for structure.

```
#include <stdio.h>   // C example
#include <stdlib.h>
#include <string.h>
typedef struct Student {
   char* name;  // needs memory allocation
   int id;
} Student; // 8 or 12 bytes
void Store(Student* s, const char* n, int i);
void Copy(Student* a, const Student b);
void Delete(Student* s);
void Print(Student s);

int main()
{
   Student* a = malloc(sizeof(Student));
   Student b, c;
   Store(a, "Kim", 123); Store(&b, "Lee", 234);
   Copy(&c, *a); Print(*a); Print(b); Print(c);
   Delete(a); free(a); Delete(&b); Delete(&c);
   return 0;
}
```



(Kim 123) (Lee 234) (Kim 123)

```
void Store(Student* s, const char* n, int i)
{ // Allocate memory and store student info
   s->name = (char*) malloc(strlen(n)+1);
   strcpy(s->name, n); s->id = i;
}

void Copy(Student* a, const Student b)
{ // Allocate memory and copy student info
   a->name = (char*) malloc(strlen(b.name)+1);
   strcpy(a->name, b.name); a->id = b.id;
}

void Delete(Student* s)
{ // Free memory
   if (s->name != NULL) free(s->name);
   s->name = NULL;
}

void Print(Student s) { // Print student info
   printf("(%s %d) ", s.name, s.id);
}
```

C Structure vs. C++ Class

The main purpose of C++ programming is to add object orientation to the C programming language. A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package.

```
typedef struct Student {
   char name[30];     // public
   int id;     // can be accessed from outside
} Student;
void Store(Student* s, char* n, int i);
void Print(Student s);
```

```
class Student {
   char name[30];     // data member
   int id;                    // default: private
public:     // can be accessed from outside
   void Store(char* n, int i); // member function
   void Print();
};
```

```
typedef struct Student {
   char* name;
   int id;
} Student;
void Store(Student* s, const char* n, int i);
void Copy(Student* a, const Student b);
void Delete(Student* s);
```

```
class Student {   public: accessible from anywhere
   char* name;    private: cannot be accessed from
   int id;                 outside the class (default)
public:          protected: accessed in derived classes
   Student(const char* n, int i); // constructor
   Student(const Student& b);
   ~Student();                    // destructor
};
```

Programs are divided into entities known as objects; Data structures are designed such that they characterize objects; Data is hidden and cannot be accessed by external functions
Encapsulation is capturing data and keeping it safely and securely from outside interfaces
Abstraction is the ability to represent data at a very conceptual level without any details

Object-Oriented Programming

- Object-Oriented Programming
  - Programming paradigm based on the concept of objects, which can contain data and code
  - Organizes software design around data, or objects, rather than functions and logic
    - Focuses on the objects that developers want to manipulate rather than the logics required to manipulate them
- Principles of Object-Oriented Programming
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

Programs are divided into entities known as objects; Data structures are designed such that they characterize objects; Data is hidden and cannot be accessed by external functions

Encapsulation is capturing data and keeping it safely and securely from outside interfaces
Abstraction is the ability to represent data at a very conceptual level without any details

Student Information - 1 (C++)

Classes are an expanded concept of data structures: classes can contain data members but also contain functions as members. An object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

```
#include <iostream>
#include <cstring>   // C library for string
#define SIZE 32
using namespace std;
class Student {
   char name[SIZE]; // default: private
   int id;          // information hiding
public:             // accessible
   void Store(char* name, int id);
   void Print();
};

int main()
{
   Student s;
   s.Store("Kim", 1234);
   s.Print();
   return 0;
}
```

```
void Student::Store(char* name, int id)
{ // Store student information
   strcpy(this->name, name);
   this->id = id;
}
// 'this pointer' is an implicit parameter
// to all member functions.

void Student::Print()
{ // Print student information
   cout << "Student name = " << name;
   cout << "; id = " << id << endl;
}
```

Abstraction is the ability to represent data at a very conceptual level without any details

Encapsulation is capturing data and keeping it safely and securely from outside interfaces

```
Student: name = Kim; id = 1234
```

Student Information - 2 (C++)

This program is an example of constructor, destructor, and copy constructor in C++.

```
#include <iostream>
#include <cstring>
using namespace std;
class Student {
   char* name;
   int id;
public:
   Student(const char* n, int i);// constructor
   Student(const Student&);// copy constructor
   ~Student();                    // destructor
   void Print();
};

                  (Kim 123) (Lee 234) (Kim 123)
int main()
{                              // a, b created
   Student* a = new Student("Kim", 123);
   Student b("Lee", 234), c = *a; // c copied
   a->Print(); b.Print(); c.Print();
   delete a;                   // a destructed
   return 0;                   // b, c destructed
}
```

```
Student::Student(const char* n, int i)
{ // Constructor (executed when created)
   name = new char[strlen(n) + 1];
   strcpy(name, n); id = i;
}

Student::Student(const Student& s)
{ // Copy constructor (executed when copied)
   name = new char[strlen(s.name) + 1];
   strcpy(name, s.name); id = s.id;
}

Student::~Student()
{ // Destructor (executed when deleted)
   if (name != NULL) delete[] name;
   name = NULL;
}

void Student::Print() { // Print student info
   cout << "(" << name << " " << id << ") ";
}
```

C++ Structure

In C++, a structure works the same as a class except for default accessibility.

```
#include <iostream> // class example
using namespace std;

class Count {
  int* num;             // private
public:
  Count(int n=0) { num = new int; *num = n; }
  ~Count() { delete num; }
  Count(const Count& c) {
    num = new int; *num = *(c.num); }
  void Increment() { (*num)++; }
  void Print() {
    cout << "num = " << *num << " "; }
}; // default is private

int main()
{
  Count n1(10), n2 = n1;
  n2.Increment(); n2.Print();
  return 0;
}
```

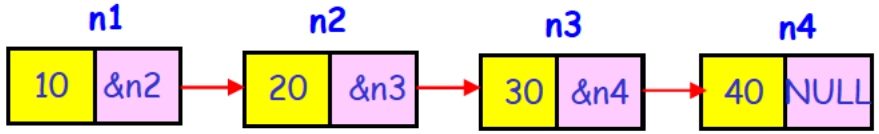
```
#include <iostream> // structure example
using namespace std;

struct Count {
  int* num;             // public
  Count(int n=0) { num = new int; *num = n; }
  ~Count() { delete num; }
  Count(const Count& c) {
    num = new int; *num = *(c.num); }
  void Increment() { (*num)++; }
  void Print() {
    cout << "num = " << *num << " "; }
}; // default is public


int main()
{
  Count n1(10), n2 = n1;
  n2.Increment(); n2.Print();
  return 0;
}
```

num = 11

Pointer to Structure (C++)

Structure is a user defined data type that allows to combine data items of different kinds.
'new' is used for memory allocation in C++.

```
#include <iostream>
using namespace std;
typedef int Data; // user-defined data type

struct Node { // in structure, default is 'public'
   Data data;
   Node* next; // next pointer
}; // In C++, 'struct' is the same as 'class'
except that default is 'public'

int main()
{
   Node n1, n2, n3, n4;
   n1.data = 10; n1.next = &n2;
   n2.data = 20; n2.next = &n3;
   n3.data = 30; n3.next = &n4;
   n4.data = 40; n4.next = NULL;
```




```
   for (Node* p = &n1; p != NULL; p = p->next){
      // p = &n1, &n2, &n3, &n4, NULL
      cout << p->data << " ";
   } // p->data = (*p).data

   // Node poineter and memory allocation
   Node* p1 = new Node();
   Node* p2 = new Node();
   Node* p3 = new Node();
   Node* p4 = new Node();
   p1->data = 10; p1->next = p2;
   p2->data = 20; p2->next = p3;
   p3->data = 30; p3->next = p4;
   p4->data = 40; p4->next = NULL;
   for (Node* p = p1; p != NULL; p = p->next) {
      // p = p1, p2, p3, p4, NULL
      cout << p->data << " ";
   }
   return 0;
}
```

```
10 20 30 40 10 20 30 40
```

Adding Two Complex Numbers

This program takes two complex numbers as structures and adds them.

```
#include <stdio.h>
typedef struct Complex {
   double real, imag;
} Complex;
void Assign(Complex* n, double f1, double f2);
Complex Add1(Complex n1, Complex n2);
void Add2(Complex* sum, Complex n1,
                        Complex n2);
void Print(char s[], Complex n);

int main()
{
   Complex n1 = {1.2, 2.3}, n2, n3, n4, sum;
   n2 = (Complex) {3.4, 4.5};
   sum = Add1(n1, n2);    // function output
   Print("n1 + n2 = ", sum);
   Assign(&n3, 2.3, 4.5); Assign(&n4, 3.4, 5.1);
   Add2(&sum, n3, n4);   // call by address
   Print("n3 + n4 = ", sum);
}
```

```
n1 + n2 = 4.6 + 6.8i; n1 + n2 = 5.7 + 9.6i;
```

```
void Assign(Complex* n, double f1, double f2)
{ // Assign values to complex number
   n->real = f1; n->imag = f2;
} // *n = (Complex) {f1, f2};

Complex Add1(Complex n1, Complex n2)
{ // Add two complex numbers (version 1)
   return (Complex) {n1.real + n2.real,
                     n1.imag + n2.imag};
}

void Add2(Complex* sum, Complex n1,
                        Complex n2)
{ // Add two complex numbers (version 2)
   sum->real = n1.real + n2.real;
   sum->imag = n1.imag + n2.imag;
}

void Print(char s[], Complex n) { // char* s
   printf("%s%.1f + %.1fi; ", s, n.real, n.imag);
}
```

Adding Two Complex Numbers (C++)

It is allowed to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

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

class Complex {
  double real, imag;  // real, imaginary numbers
public:
  // Constructor overloading
  Complex(double r, double i) { real = r; imag = i; }
  Complex(double r) { real = r; imag = 0.0; }
  Complex() { real = 0.0; imag = 0.0; }

  // Operator overloading
  Complex operator+ (const Complex& c) {
    return Complex(real + c.real, imag + c.imag); }

  void Print() {
    cout << "(" << real << ", " << imag << ") "; }
};
```

```
int main()
{
  Complex a(1.0, 2.0), b(3.0), c;
  c.Print();
  for (int i = 0; i < 3; i++) {
    c = c + a; // c = c.operator+(a);
    c.Print();
  }
  c = a + a + a + b;
  c.Print();
  c = b; // using default copy constructor
  c.Print();
}
```

```
// Default arguments
Complex(double r=0, double i=0)
  { real = r; imag = i; }
```

```
(0, 0) (1, 2) (2, 4) (3, 6) (6, 6) (3, 0)
```

Difference between Two Time Periods

This program calculates the difference between two time periods assuming that the time difference is less than 12 hours.

```
#include <stdio.h>
typedef struct Time {
   int hour;
   int minute;
} Time;
Time Subtract(Time t1, Time t2);
void Print(char s[], Time t);

int main()
{
   Time diff, start = {2, 45}, end = {5, 15};
   diff = Subtract(end, start);
   Print("Time difference: ", diff);
   return 0;
}
```

```
Time Subtract(Time t1, Time t2)
{ // Subtract two time values
   Time diff;
   diff.hour = t1.hour - t2.hour;
   diff.minute = t1.minute - t2.minute;
   if (diff.minute < 0) {
      diff.minute += 60;
      diff.hour--;
   }
   if (diff.hour < 0) {
      diff.hour += 12;
   }
   return diff;
}

void Print(char s[], Time t) // (char* s, Time t)
{ // Print time values
   printf("%s%d hours %d minutes\n",
            s, t.hour, t.minute);
}
```

```
Time difference: 2 hours 30 minutes
```

Difference between Two Time Periods (C++)

This program calculates the difference between two time periods.

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

class Time {
   int hour, minute;
public: // constructor and operator overloading
   Time(int h, int m) { hour = h; minute = m; };
   Time() { hour = 0; minute = 0; }
   Time operator- (const Time& t);
   void Print();
};

int main()
{
   Time diff, startTime(2, 45), endTime(5, 15);
   diff = endTime - startTime;
   diff.Print();
   return 0;
}
```

```
// Default arguments
Time (int h=0, int m=0) {
   hour = h; minute = h; }
```

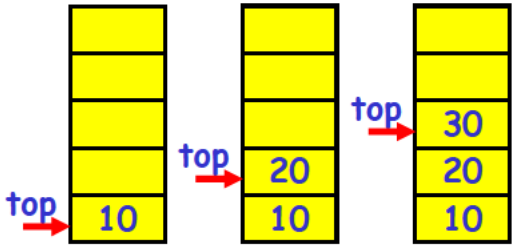
```
Time Time::operator- (const Time& t)
{ // Subtract two time values
   Time diff;
   diff.hour = hour - t.hour;
   diff.minute = minute - t.minute;
   if (diff.minute < 0) {
      diff.minute += 60;
      diff.hour--;
   }
   if (diff.hour < 0) {
      diff.hour += 12;
   }
   return diff;
}

void Time::Print()
{ // Print time
   cout << hour << " hours ";
   cout << minute << " minutes\n";
}
```

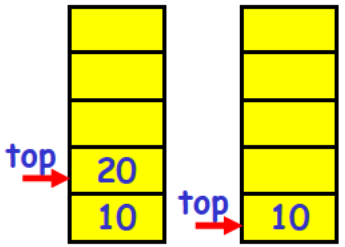
```
2 hours 30 minutes
```

Stack

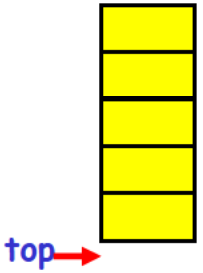
A stack is a data structure which is used to store data in a Last In First Out (LIFO) order.

```
#include <stdio.h>
#include <stdlib.h>
#define ERROR(s) { fprintf(stderr,s); exit(1); }
#define SIZE 5
typedef struct Stack {
   int data[SIZE], top;
} Stack;                    // 24 bytes
void Initialize(Stack* s);
void Push(Stack* s, int n);
int Pop(Stack* s);

int main()
{
   Stack s;
   Initialize(&s);
   Push(&s, 10); Push(&s, 20); Push(&s, 30);
   printf("%d ", Pop(&s));
   printf("%d ", Pop(&s));
   return 0;
}
```






```
void Initialize(Stack* s)
{ // Initialize stack
   s->top = -1; // (*s).top = -1;
}

void Push(Stack* s, int n)
{ // Insert a new node to stack
   if (s->top == SIZE-1) ERROR("Stack is full");
   s->data[++(s->top)] = n;
}

int Pop(Stack* s)
{ // Delete latest node from stack
   if (s->top == -1) ERROR("Stack is empty");
   return s->data[s->top--];
}
```




```
30 20
```

Stack (C++)

This program shows an example of class and default constructor in C++.

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define ERROR(s) { std::cerr << s; exit(-1); }
#define SIZE 5
class Stack {
   int data[SIZE], top; // private (not accessible)
public:
   Stack();              // default constructor
   void Push(int n);
   int Pop();
};

int main()
{
   Stack s;              // declare & initialize
   s.Push(10); s.Push(20); s.Push(30);
   cout << s.Pop() << " ";
   cout << s.Pop();
   return 0;
}
```

```
Stack::Stack()
{ // Default constructor
   top = -1;
}

void Stack::Push(int n)
{ // Insert a new node to stack
   if (top == SIZE-1) ERROR("Stack is full");
   data[++top] = n;
}

int Stack::Pop()
{ // Delete latest node from stack
   if (top == -1) ERROR("Stack is empty");
   return data[top--];
}
```

```
30 20
```

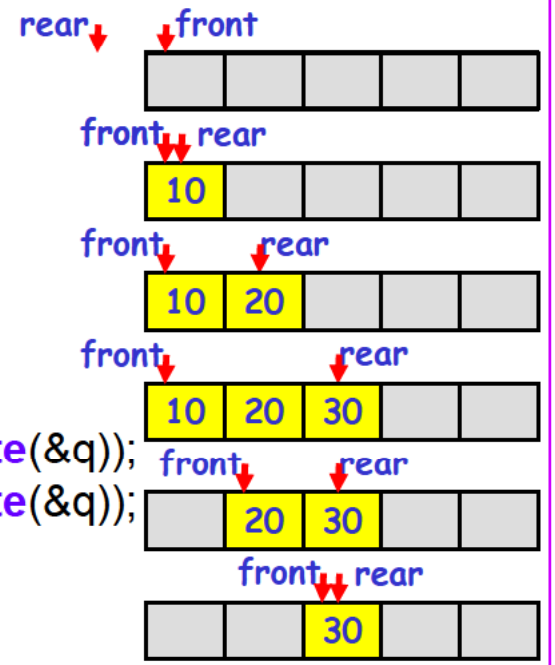
Queue

A queue is a data structure which is used to store data in a First In First Out (FIFO) order.

```
#include <stdio.h>
#include <stdlib.h>
#define EXIT(s) { fprintf(stderr,s); exit(-1); }
#define SIZE 5 // size of queue

typedef struct Queue {
   int data[SIZE], front, rear, length;
} Queue; // 8 * 4 bytes = 32 bytes

void Initialize(Queue* q)
{ // Initialize queue
   q->front = 0; q->rear = -1; q->length = 0;
}

void Insert(Queue* q, int data)
{ // Insert a new node
   if (q->length == SIZE) EXIT("Queue is full");
   q->rear = (q->rear + 1) % SIZE;
   q->data[q->rear] = data;
   q->length++;
}
```

```
int Delete(Queue* q)
{ // Delete oldest node
   int data = q->data[q->front];
   if (q->length == 0) EXIT("Queue is empty");
   q->front = (q->front + 1) % SIZE;
   q->length--;
   return data;
}

int main()
{
   Queue q;
   Initialize(&q);
   Insert(&q, 10);
   Insert(&q, 20);
   Insert(&q, 30);
   printf("%d ", Delete(&q));
   printf("%d ", Delete(&q));
   return 0;
}
```

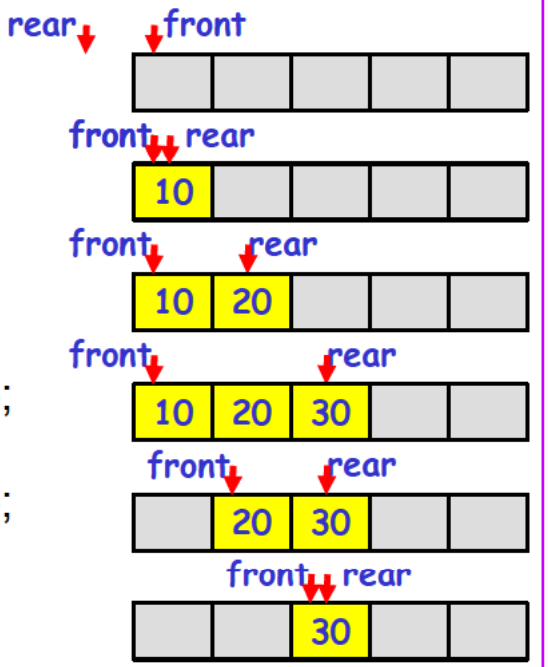
10 20



Queue (C++)

A queue is a data structure which is used to store data in a First In First Out (FIFO) order.

```
#include <iostream>
#include <cstdlib>
using namespace std;
const int QUEUE_SIZE = 5;
typedef int Data;
void Error(string s) { cerr << s; exit(-1); }

class Queue { // array representation of queue
  Data data[QUEUE_SIZE]; // queue array
  int front, rear, length; // first, last, # of data
public:
  Queue() { front = 0; rear = -1; length = 0; }
  void Insert(Data d) { // insert a new element
    if (IsFull()) Error("Queue is full");
    rear = (rear + 1) % QUEUE_SIZE;
    data[rear] = d; length++; }
  Data Delete() { // delete oldest element
    if (IsEmpty()) Error("Queue is empty");
    Data out = data[front];
    front = (front + 1) % QUEUE_SIZE;
    length--; return out; }
  Data Peek() { // get oldest element
    if (IsEmpty()) Error("Queue is empty");
    return data[front]; }
  bool IsEmpty() { return length == 0; }
  bool IsFull() { return length==QUEUE_SIZE; }
};

int main()
{
  Queue q;
  q.Insert(10);
  q.Insert(20);
  q.Insert(30);
  cout << q.Delete();
  cout << " ";
  cout << q.Delete();
  return 0;
}
```




```
10 20
```

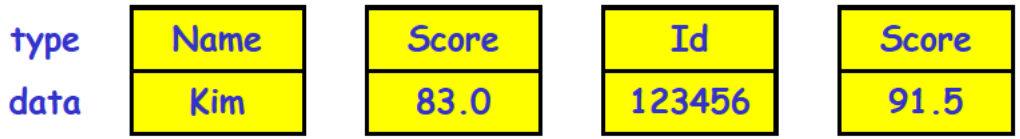
Student Information using Union

A union is a special data type that allows to store different data types in the same memory location. Only one member can contain a value at a given time.

```
#include <stdio.h>
#include <string.h>
typedef enum Type {
   Name, Id, Score
} Type; // sizeof(Type) = 4
typedef union Data {
   char name[12];
   int id;
   float score;
} Data; // sizeof(Data) = 12
typedef struct Student {
   Type type;
   Data data;
} Student;
void Print(Student s);
```

```
Name = Kim; Score = 83.0
Id = 123456; Score = 91.5
```

```
int main()
{
   Student s[4];
   s[1].type = Name; strcpy(s[1].data.name, "Kim"); Print(s[1]);
   s[2].type = Score; s[2].data.score = 83.0;       Print(s[2]);
   s[3].type = Id;    s[3].data.id = 123456;        Print(s[3]);
   s[4].type = Score; s[4].data.score = 91.5;       Print(s[4]);
   return 0;
}

void Print(Student s)
{ // Print student information
   switch (s.type) {
      case Name:    printf("Name = %s; ", s.data.name); break;
      case Id:      printf("Id = %d; ", s.data.id); break;
      case Score:   printf("Score = %.1f\n", s.data.score);
   }
}
```



Student Information using Pointer of Function

This program illustrates structure with a pointer of a function.

```
#include <stdio.h>
#include <string.h>
#define SIZE 100
typedef struct Student {
   char name[SIZE];
   void (*Print) (char* s);
} Student;
void PrintFreshman(char* name);
void PrintSophomore(char* name);

int main()
{
   Student s;
   strcpy(s.name, "Kim");
   s.Print = PrintFreshman;
   s.Print(s.name);
   s.Print = PrintSophomore;
   s.Print(s.name);
   return 0;
}
```

```
void PrintFreshman(char* name)
{ // Print for a freshman
   printf("%s is a freshman\n", name);
}

void PrintSophomore(char* name)
{ // Print for sophomore
   printf("%s is a sophomore\n", name);
}
```

```
Kim is a freshman
Kim is a sophomore
```