

```
#define PROCEDURE void
#define Main main()
#define BEGIN {
#define END }
#define VAR int
#define Integer
#define READLN scanf("%d",&
#define val Val)
#define WHILE while(
#define DO )
#define MOD %
#define DIV /
#define PRINTLN printf("%d\n",
#define digit Digit)
```

```
#include <stdio.h>
#include <stdlib.h>
#include "exam54.h"
```

```
PROCEDURE Main
BEGIN
```

```
    VAR
        Val, Digit  Integer;
```

```
    READLN val;
    WHILE Val > 0 DO
    BEGIN
```

```
        Digit = Val MOD 2;
        Val = Val DIV 2;
```

```
        PRINTLN digit;
```

```
    END
```

```
END
```

```
#define Sub
#define Main void main() {
#define Dim int
#define As ;
#define Integer
#define End }
#define VAL Val) {
#define Get scanf("%d",&
#define val Val);
#define While while(
#define Mod %
#define Put printf("%d\n",
#define digit Digit);
#define Two 2;
#define Wend }
```

```
#include <stdio.h>
#include <stdlib.h>
#include "exam55.h"

Sub Main
  Dim Val As Integer
  Dim Digit As Integer

  Get val
  While 0 < VAL
    Digit = Val Mod Two
    Val = Val / Two
    Put digit
  Wend
End Sub
```

```

#include <stdio.h>
#include <stdlib.h>

typedef struct person
{
    struct person *Next;
    char *Name;
    char *Address;
    int MaritalStatus;
}
PERSON;

/* Marital Status Codes */
#define UNKNOWN 0
#define SINGLE 1
#define MARRIED 2
#define DIVORCED 3
#define WIDOWED 4

void main()
{
    int i = MARRIED;

    switch (i)
    {
        case UNKNOWN:
        {
            printf("Status UNKNOWN\n");
        }
        break;
        case SINGLE:
        {
            printf("Status SINGLE\n");
        }
        break;
        case MARRIED:
        {
            printf("Status MARRIED\n");
        }
        break;
        case DIVORCED:
        {
            printf("Status DIVORCED\n");
        }
        break;
        case WIDOWED:
        {
            printf("Status WIDOWED\n");
        }
        break;
        default:
        {
            printf("Status ?\n");
        }
        break;
    }
}

```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i;

    for (i=0 ; i < argc ; i++)
    {
        fprintf(stdout, "%s\n", argv[i]);
    }
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i;
    FILE *Infile;
    char tc;

    if (argc <= 1)
    {
        tc = getc(stdin);
        while (!feof(stdin))
        {
            putchar(tc, stdout);
            tc = getc(stdin);
        }
        return 0;
    }
    for (i=1 ; i < argc ; i++)
    {
        Infile = fopen(argv[i], "r");
        if (Infile == NULL)
        {
            continue;
        }
        tc = getc(Infile);
        while (!feof(Infile))
        {
            putchar(tc, stdout);
            tc = getc(Infile);
        }
        fclose(Infile);
    }
    return 0;
}
```

```

#include <stdio.h>
#include <stdlib.h>

int Abs(int x);
int Sqrt(int x);

typedef struct point
{
    int x;
    int y;
}
POINT;

int main(int argc, char *argv[])
{
    POINT p1, p2;
    int dx, dy, sum, dist;

    dx = Abs(p1.x-p2.x);
    dy = Abs(p1.y-p2.y);
    sum = dx*dx + dy*dy;
    dist = Sqrt(sum);
    return 0;
}

int Abs(int x)
{
    if (x<0)
    {
        return -x;
    }
    return x;
}

int Sqrt(int x)
{
    int rv = 0;
    int sq = 0;

    for (;;)
    {
        sq = sq + 2*rv + 1;
        if (sq > x)
        {
            return rv;
        }
        else
        {
            rv++;
        }
    }
}

```

```

#include <stdio.h>
#include <stdlib.h>

#define Abs(x) (((x)>=0)?(x):(-(x)))
//int Abs(int x);
int Sqrt(int x);

typedef struct point
{
    int x;
    int y;
}
POINT;

int main(int argc, char *argv[])
{
    POINT p1,p2;
    int dx,dy,sum,dist;

    dx = Abs(p1.x-p2.x);
    dy = Abs(p1.y-p2.y);
    sum = dx*dx + dy*dy;
    dist = Sqrt(sum);
    return 0;
}

//int Abs(int x)
//{
//    if (x<0)
//    {
//        return -x;
//    }
//    return x;
//}

int Sqrt(int x)
{
    int rv = 0;
    int sq = 0;

    for (;;)
    {
        sq = sq + 2*rv + 1;
        if (sq > x)
        {
            return rv;
        }
        else
        {
            rv++;
        }
    }
}

```

```
#include <stdio.h>
#include <stdlib.h>

#define Ratio(x,y) x/y

int main(int argc, char *argv[])
{
    int x,y,r;

    if (argc < 3)
    {
        exit(100);
    }
    x = atoi(argv[1]);
    y = atoi(argv[2]);
    r = Ratio(x,y);
    printf("The ratio of %d and %d is %d\n",x,y,r);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

#define Ratio(x,y) ((x)/(y))
int Xint[3];

int main(int argc, char *argv[])
{
    int *x, *y, *r;

    x = Xint;
    y = Xint + 1;
    r = Xint + 2;
    if (argc < 3)
    {
        exit(100);
    }
    *x = atoi(argv[1]);
    *y = atoi(argv[2]);
    // *r = *x/*y;
    // *r = ((*x)/(*y));
    *r = Ratio(*x,*y);
    printf("The of %d and %d is %d\n", *x, *y, *r);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

int Abs(int x);
int Sqrt(int x);

typedef struct point
{
    int x;
    int y;
}
POINT;

typedef struct phold
{
    struct phold *Next;
    POINT pt;
}
PHOLD;

extern PHOLD *Head,*Tail;

int GetDist(POINT
```

```

#include "exam63.h"

int main(int argc, char *argv[])
{
    FILE *PointFile;
    int x,y,Sum;
    PHOLD *THold;

    PointFile = fopen("points.txt","r");
    if (PointFile == NULL)
    {
        fprintf(stderr,"Can't Open Point File\n");
        exit(100);
    }
    fscanf(PointFile,"%d %d",&x,&y);
    while (!feof(PointFile))
    {
        THold = (PHOLD *)malloc(sizeof(PHOLD));
        if (THold == NULL)
        {
            fprintf(stderr,"Out of Memory\n");
            exit(100);
        }
        THold->pt.x = x;
        THold->pt.y = y;
        if (Head == NULL)
        {
            Head = THold;
        }
        else
        {
            Tail->Next = THold;
        }
        THold->Next = NULL;
        Tail = THold;
        fscanf(PointFile,"%d %d",&x,&y);
    }
    fclose(PointFile);
    for (THold = Head,Sum=0 ; THold && THold->Next ; THold = THold->Next)
    {
        Sum += GetDist(THold->pt,THold->Next->pt);
    }
    printf("The Sum is: %d\n",Sum);
    return 0;
}

```

```
#include <iostream.h>

void main()
{
    int Value;

    cout << "Enter a Number: ";
    cin >> Value;
    cout << "The Square of " << Value << " is " << Value * Value << ".\n";
}
```

```
#include <iostream.h>

void main()
{
    char Name[20];

    cout << "Please enter your name: ";
    cin >> Name;
    cout << "Hello " << Name << "!\n";
}
```

```

#include <iostream.h>
#include <string.h>

void Convert(int n , char *OutStr , int base = 2);

void main()
{
    int Number,Base;
    char MyString[20];

    cout << "Enter a Number: ";
    cin >> Number;
    cout << "Enter a base: ";
    cin >> Base;
    Convert(Number,MyString,Base);
    cout << "Number is " << MyString;
}

void Convert(long n , char *OutStr , int base)
{
    int tn;
    char rv[20],*ts;

    rv[19] = '\0';
    for (ts = rv+18,tn=n ; tn > 0 ; ts--)
    {
        *ts = '0' + (tn % base);
        tn /= base;
    }
    strcpy(OutStr,ts+1);
}

```

```
#include <iostream.h>

void main()
{
    cout << "Enter a Number ";
    int Value;
    cin >> Value;
    for (int i = 1, Sum=0 ; i<=Value ; i++)
    {
        Sum += i;
    }
    cout << "The Sum Is: " << Sum;
}
```

```
#include "iostream.h"

void main()
{
    const int Two = 2;

    Two = 10; // This is illegal
}
```

```

#include <iostream.h>

inline double max(const double x,const double y);
int max(int x,int y);
double max(double x,int y);
double max(int x,double y);

void main()
{
    cout << "max of 1 and 2 is " << max(1,2) << "\n";
    cout << "max of 1.1 and 1.2 is " << max(1.2,1.1) << "\n";
    cout << "max of 1.1 and 1 is " << max(1,1.1) << "\n";
}

double max(int x,double y)
{
    if ((double)x > y)
    {
        return (double)x;
    }
    else
    {
        return y;
    }
}

double max(double x,int y)
{
    if (x > (double)y)
    {
        return x;
    }
    else
    {
        return (double)y;
    }
}

inline double max(const double x,const double y)
{
    if (x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}

int max(int x,int y)
{
    if (x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}

```

```

#include <iostream.h>

class Complex
{
public:
    Complex(int x,int y);
    Complex();
    ~Complex();
    int Real();
    int Imaginary();
    void Incr(Complex x);
    void Decr(Complex x);
    Complex operator+(Complex x);
private:
    int r,i;
};

Complex A,B(5,-31);
Complex *C;

Complex::Complex(int x,int y)
{
    r = x;
    i = y;
}

Complex::Complex()
{
    r = 0;
    i = 0;
}

Complex Complex::operator+(Complex x)
{
    return Complex(r+x.r,i+x.i);
}

void main()
{
    int x;
    x = A.Real();
    A.Incr(B);
    C = &B;
    C->Decr(B);
    A = A + B;
}

int Complex::Real()
{
    return r;
}

int Complex::Imaginary()
{
    return i;
}

void Complex::Incr(Complex x)
{
    r += x.r;
    i += x.i;
}

void Complex::Decr(Complex x)
{
    r -= x.r;
    i -= x.i;
}

```

```

#include <iostream.h>

class Complex
{
public:
    Complex();
    Complex(int x);
    Complex(int x, int y);
    Complex(const Complex &Other);
    int Real();
    int Imaginary();
    void Incr(Complex x);
    void Decr(Complex x);
private:
    int r,i;
};

int Complex::Real()
{
    return r;
}

int Complex::Imaginary()
{
    return i;
}

void Complex::Incr(Complex x)
{
    r += x.r;
    i += x.i;
}

void Complex::Decr(Complex x)
{
    r -= x.r;
    i -= x.i;
}

Complex::Complex()
{
    r = 0;
    i = 0;
}

Complex::Complex(int x)
{
    r = x;
    i = 0;
}

Complex::Complex(int x,int y)
{
    r = x;
    i = y;
}

Complex::Complex(const Complex &Other)
{
    r = Other.r;
    i = Other.i;
}

void main()
{
}

```

```

#include <iostream.h>

class Complex
{
public:
    Complex();
    Complex(int x);
    Complex(int x, int y);
    Complex(const Complex &Other);
    void operator=(Complex x);
    friend Complex operator+(Complex x,Complex y);
    friend Complex operator+(int x,Complex y);
    friend Complex operator+(Complex x,int y);
    friend Complex operator-(Complex x,Complex y);
    friend Complex operator*(Complex x,Complex y);
    friend Complex operator/(Complex x,Complex y);
    int Real();
    int Imaginary();
    void Incr(Complex x);
    void Decr(Complex x);
private:
    int r,i;
};

void Complex::operator=(Complex x)
{
    r = x.r;
    i = x.i;
}

Complex operator+(Complex x,Complex y)
{
    return Complex(x.r+y.r,x.i+y.i);
}

Complex operator-(Complex x,Complex y)
{
    return Complex(x.r-y.r,x.i-y.i);
}

Complex operator*(Complex x,Complex y)
{
    return Complex((x.r*y.r)-(x.i*y.i),(x.r*y.i)-(x.i*y.r));
}

Complex operator/(Complex x,Complex y)
{
    int d,n1,n2;

    d = (y.r*y.r) + (y.i*y.i);
    n1 = (x.r*y.i) + (x.i*y.i);
    n2 = (x.i*y.r) - (x.r*y.r);
    return Complex(n1/d,n2/d);
}

int Complex::Real()
{
    return r;
}

int Complex::Imaginary()
{
    return i;
}

void Complex::Incr(Complex x)
{
    r += x.r;
    i += x.i;
}

void Complex::Decr(Complex x)
{
    r -= x.r;
    i -= x.i;
}

Complex::Complex()
{
    r = 0;
}

```

```
        i = 0;
    }
Complex::Complex(int x)
{
    r = x;
    i = 0;
}
Complex::Complex(int x,int y)
{
    r = x;
    i = y;
}
Complex::Complex(const Complex &Other)
{
    r = Other.r;
    i = Other.i;
}
```

```
void main()
{
    Complex A,B,C;

    A = B + 2;
    A = 2 + B;
    A = B+C;
    A += B;
    A++;
}
```

```

#include <iostream.h>

class Complex
{
public:
    Complex();
    Complex(int x);
    Complex(int x, int y);
    Complex(const Complex &Other);
    void operator=(Complex x);
    friend Complex operator+(Complex x,Complex y);
    friend Complex operator+(int x,Complex y);
    friend Complex operator+(Complex x,int y);
    friend Complex operator-(Complex x,Complex y);
    friend Complex operator*(Complex x,Complex y);
    friend Complex operator/(Complex x,Complex y);
    int Real();
    int Imaginary();
    void Incr(Complex x);
    void Decr(Complex x);
private:
    int r,i;
};

void Complex::operator=(Complex x)
{
    r = x.r;
    i = x.i;
}

Complex operator+(Complex x,Complex y)
{
    return Complex(x.r+y.r,x.i+y.i);
}

Complex operator+(int x,Complex y)
{
    return Complex(y.r+x,y.i);
}

Complex operator+(Complex x,int y)
{
    return Complex(x.r+y,x.i);
}

Complex operator-(Complex x,Complex y)
{
    return Complex(x.r-y.r,x.i-y.i);
}

Complex operator*(Complex x,Complex y)
{
    return Complex((x.r*y.r)-(x.i*y.i),(x.r*y.i)-(x.i*y.r));
}

Complex operator/(Complex x,Complex y)
{
    int d,n1,n2;

    d = (y.r*y.r) + (y.i*y.i);
    n1 = (x.r*y.i) + (x.i*y.i);
    n2 = (x.i*y.r) - (x.r*y.r);
    return Complex(n1/d,n2/d);
}

int Complex::Real()
{
    return r;
}

int Complex::Imaginary()
{
    return i;
}

void Complex::Incr(Complex x)
{
    r += x.r;
    i += x.i;
}

```

```
void Complex::Decr(Complex x)
{
    r -= x.r;
    i -= x.i;
}
Complex::Complex()
{
    r = 0;
    i = 0;
}
Complex::Complex(int x)
{
    r = x;
    i = 0;
}
Complex::Complex(int x,int y)
{
    r = x;
    i = y;
}
Complex::Complex(const Complex &Other)
{
    r = Other.r;
    i = Other.i;
}

void main()
{
    Complex A,B,C;

    A = B+C;
}
```

```
#include <iostream.h>

class animal
{
public:
    int Color;
    int Weight;
};

class bird : public animal
{
public:
    int WingSpan;
};

void main()
{
}
```

```
#include <iostream.h>

class point
{
public:
    int x;
    int y;
};

class line
{
public:
    point Begin;
    point End;
    int Length();
};

class coloredLine : line
{
private:
    long Color;
}

int line::Length()
{
    h = Abs(Begin.x-End.x);
    v = Abs(Begin.y-End.y);
    return sqrt(h*h+v*v);
}
```

```

#include <iostream.h>
#include <string.h>

class QueueElement
{
friend class Queue;
protected:
    QueueElement *Next;
    int ElementType;
public:
    int GetType();
    QueueElement();
};

int QueueElement::GetType()
{
    return ElementType;
}

QueueElement::QueueElement()
{
    Next = NULL;
}

class Queue
{
protected:
    QueueElement *Head,*Tail,*Current;
public:
    Queue();
    ~Queue();
    void Add(QueueElement *NewItem);
    QueueElement *First();
    QueueElement *Next();
    int Size();
    void Remove(QueueElement *RemItem);
    void Erase();
};

Queue::~~Queue()
{
    Erase();
}

void Queue::Erase()
{
    QueueElement *Temp,*Temp2;

    for (Temp=Head ; Temp ; Temp=Temp2)
    {
        Temp2 = Temp->Next;
        delete Temp;
    }
    Head = NULL;
    Tail = NULL;
    Current = NULL;
}

Queue::Queue()
{
    Head = NULL;
    Tail = NULL;
    Current = NULL;
}

QueueElement *Queue::First()
{
    Current = Head;
    return Head;
}

QueueElement *Queue::Next()
{

```

```

    if (Current == NULL)
    {
        return NULL;
    }
    Current = Current->Next;
    return Current;
}

void Queue::Add(QueueElement *NewItem)
{
    NewItem->Next = NULL;
    if (Head == NULL)
    {
        Head = NewItem;
    }
    else
    {
        Tail->Next = NewItem;
    }
    Tail = NewItem;
}

int Queue::Size()
{
    int Count;
    QueueElement *Temp;

    for (Temp=Head,Count=0 ; Temp ; Temp=Temp->Next,Count++);
    return Count;
}

void Queue::Remove(QueueElement *RemItem)
{
    QueueElement *Prev,*Temp;

    for (Temp=Head,Prev=NULL ; Temp && Temp!=RemItem ; Prev=Temp,Temp=Temp->Next);
    if (Temp != NULL)
    {
        if (Prev == NULL)
        {
            Head = Temp->Next;
        }
        else
        {
            Prev->Next = Temp->Next;
        }
        if (Temp->Next == NULL)
        {
            Tail = Prev;
        }
        Current = NULL;
    }
}

class StringElement : public QueueElement
{
private:
    char *String;
public:
    StringElement();
    StringElement(char *InStr);
    ~StringElement();
    char *GetValue();
    void SetValue();
    void SetValue(char *InStr);
};

StringElement::~StringElement()
{
    if (String != NULL)
    {
        delete [] String;
    }
}

```

```

StringElement::StringElement(char *InStr):QueueElement()
{
    String = new char [strlen(InStr)+1];
    strcpy(String, InStr);
}

StringElement::StringElement():QueueElement()
{
    String = NULL;
}

char *StringElement::GetValue()
{
    return String;
}

void StringElement::SetValue()
{
    if (String != NULL)
    {
        delete [] String;
    }
    String = NULL;
}

void StringElement::SetValue(char *InStr)
{
    if (String != NULL)
    {
        delete [] String;
    }
    String = new char [strlen(InStr)+1];
    strcpy(String, InStr);
}

class IntegerElement : public QueueElement
{
public:
    int Value;
};

void main()
{
    char Buf[100];
    StringElement *Temp;
    IntegerElement *Temp2;
    Queue MyQueue;
    int Num;

    cout << "Enter a String and a Number: ";
    cin >> Buf;
    cin >> Num;
    while (!cin.eof())
    {
        Temp = new StringElement(Buf);
        MyQueue.Add(Temp);
        Temp2 = new IntegerElement;
        Temp2->Value = Num;
        MyQueue.Add(Temp2);
        cout << "Enter a String ...: ";
        cin >> Buf;
        cin >> Num;
    }
    for (Temp=(StringElement *)MyQueue.First() ; Temp ; Temp=(StringElement *)MyQueue.Next())
    {
        Temp2 = (IntegerElement *)MyQueue.Next();
        cout << Temp->GetValue() << " " << Temp2->Value << "\n";
    }
    MyQueue.Erase;
}

```