Bài tập trong các bài học

Ví dụ

```
char *pchar; short *pshort; long *plong;
pchar ++; pshort ++; plong ++;
```

Giả sử các địa chỉ ban đầu tương ứng của 3 con trỏ là 100, 200 và 300, kết quả ta có các giá trị 101, 202 và 304 tương ứng

Nếu viết tiếp

```
plong += 5; => plong = 324
pchar -=10; => pchar = 91
pshort +=5; => pshort = 212
```

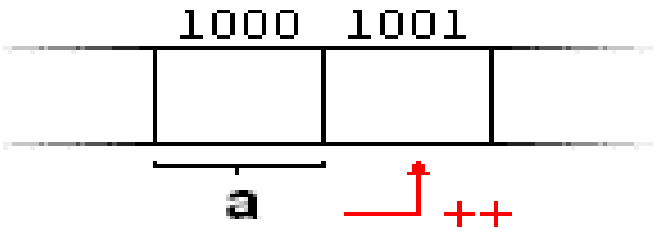
Ví dụ

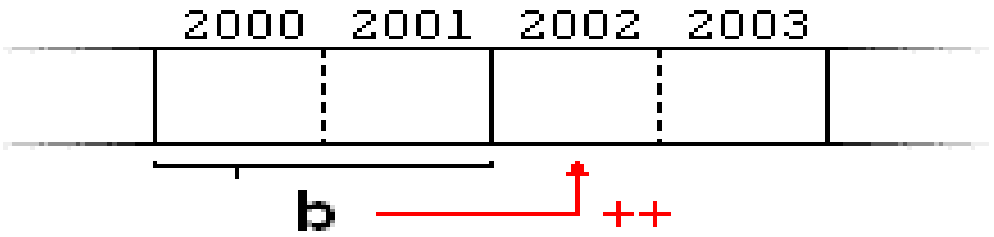
```
char *a;
short *b;
long *c;
```

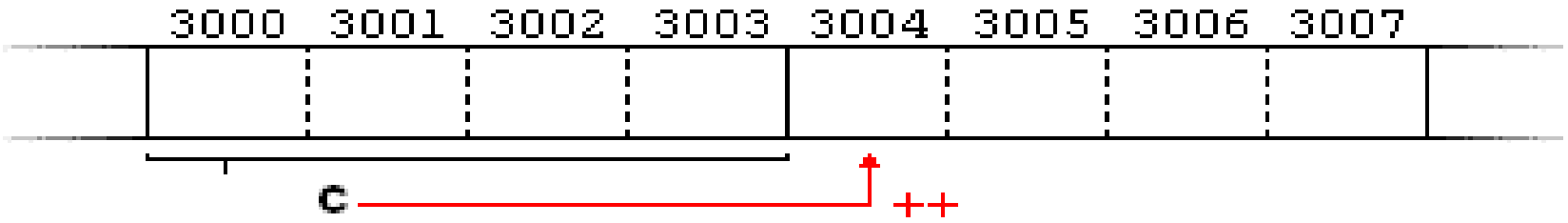
Các con trỏ a, b, c lần lượt trỏ tới ô nhớ **1000**, **2000** và **3000**.

Cộng các con trỏ với một số nguyên:

```
a = a + 1;//con trỏ a dời đi 1 byte
b = b + 1;//con trỏ b dời đi 2 byte
c = c + 1; //con trỏ c dời đi 4 byte
```







Chú ý

`++` và `--` có độ ưu tiên cao hơn `*` nên `*p++` tương đương với `*(p++)` tức là tăng địa chỉ mà nó trỏ tới chứ không phải tăng giá trị mà nó chứa.

`*p++ = *q++` sẽ tương đương với

```
*p = *q;
p=p+1;
q=q+1;
```

Ví dụ:

```
#include <iostream.h>
#include<conio.h>
void main ()
{
    int a = 20, b = 15, *pa, *pb, temp;
    pa = &a; // con trỏ pa chứa địa chỉ của a
    pb = &b; // con trỏ pb chứa địa chỉ của b
    temp = *pa;
    *pa = *pb;
    *pb = temp;
    cout << "a = " << a << endl;
    cout << “b = ” << b;
}
```

```
// kết quả xuất ra
màn hình
a = 15
b = 20
```

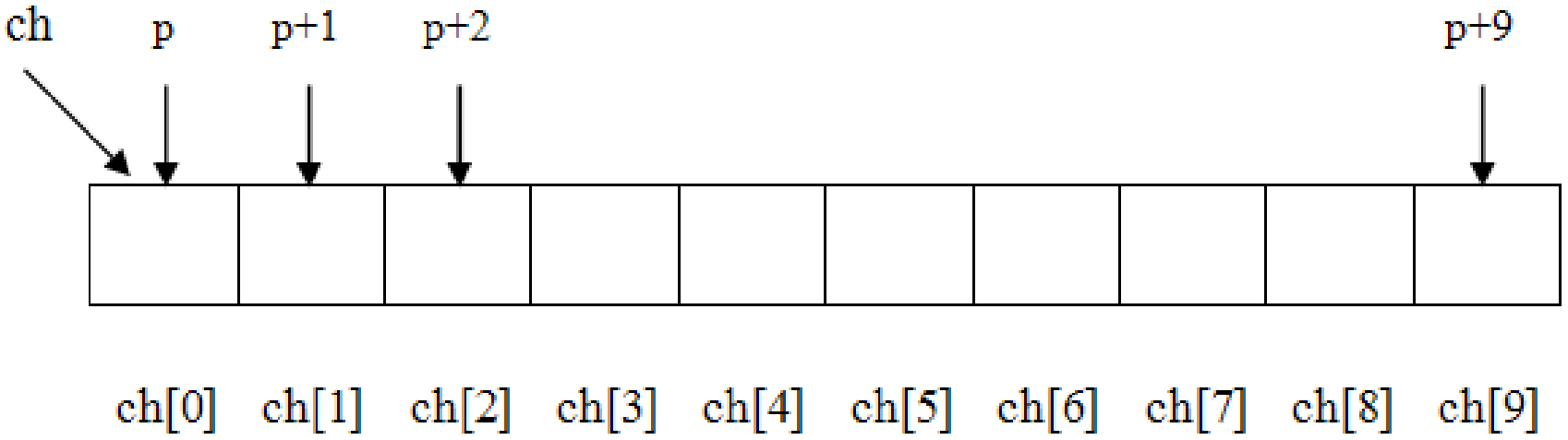
Con trỏ và mảng

- Truy cập các phần tử mảng bằng con trỏ

Kiểu mảng	Kiểu con trỏ
&<Tên mảng>[0]	<Tên con trỏ >
&<Tên mảng> [<Vị trí>]	<Tên con trỏ> + <Vị trí>
<Tên mảng>[<Vị trí>]	*(< Tên con trỏ > + <Vị trí>)

Ví dụ

```
char ch[10], *p;
p = ch;
```



- p được gán địa chỉ của phần tử đầu tiên của mảng ch.

```
p = ch;
```

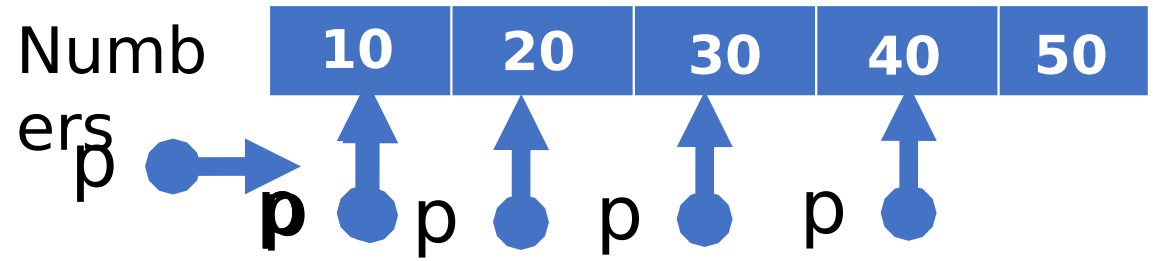
- Để tham chiếu phần tử thứ 3 trong mảng ch, ta dùng một trong 2 cách sau:
  - ch[2]
  - *(p+2)

Ví dụ

```
#include <iostream.h>
#include <conio.h>
void main ()
{
   int numbers[5], * p;
   p = numbers;  *p = 10;
   p++;  *p = 20;
   p = &numbers[2];  *p = 30;
   p = numbers + 3;  *p = 40;
   p = numbers;  *(p+4) = 50;
   for (int n=0; n<5; n++)
   cout << numbers[n] << ", ";
}
```

Ví dụ

```
int Numbers[5];
int *p;
p= Numbers;
*p = 10;
p++; *p = 20;
p = &numbers[2]; *p = 30;
p = numbers + 3; *p = 40;
p’= numbers; *(p+4) = 50;
```



Con trỏ và xâu

- Ta có `char tinhthanh[30] =“Da Lat”;`
- Tương đương :

```
char *tinhthanh;
tinhthanh=“Da lat”;
```

Hoặc: `char *tinhthanh =“Da lat”;`

- Ngoài ra các thao tác trên xâu cũng tương tự như trên mảng

```
*(tinhthanh+3) = “l”
```

- Chú ý : với xâu thường thì không thể gán trực tiếp như dòng thứ 3

Mảng các con trỏ

- Một ưu điểm khác của mảng trỏ là ta có thể hoán chuyển các đối tượng (mảng con, cấu trúc..) được trỏ bởi con trỏ này bằng cách hoán đổi các con trỏ
- Ưu điểm tiếp theo là việc truyền tham số trong hàm
- Ví dụ: Vào danh sách lớp theo họ và tên, sau đó sắp xếp để in ra theo thứ tự ABC.

```
#include <stdio.h>
#include <string.h>
#define MAXHS  50
#define MAXLEN 30
```

Mảng các con trỏ

```
int main () {
  int  i, j, count = 0;   char ds[MAXHS][MAXLEN];
  char *ptr[MAXHS],  *tmp;
  while ( count < MAXHS) {
      printf(" Vao hoc sinh thu : %d  ",count+1);
      gets(ds[count]);
      if (strlen(ds[count] == 0) break;
      ptr[count] = ds +count;
      ++count;
   }
   for ( i=0;i<count-1;i++)
      for ( j =i+1;j < count; j++)
         if (strcmp(ptr[i],ptr[j])>0) {
           tmp=ptr[i]; ptr[i] = ptr[j]; ptr[j] = tmp;
   }
   for (i=0;i<count; i++)
      printf("\n %d :  %s", i+1,ptr[i]);
}
```

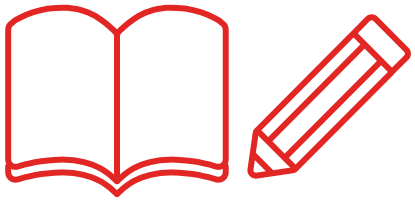
Con trỏ trỏ tới con trỏ

Ví dụ: in ra một ma trận vuông và cộng mỗi phần tử của ma trận với 10

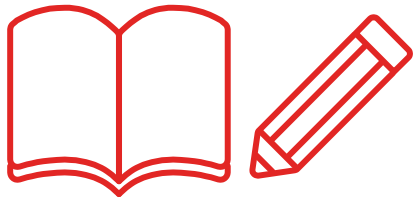
```
#include <stdio.h>
#define hang 3
#define cot 3
int main() {
  int mt[hang][cot] = {{7,8,9},
                       {10,13,15},
                       {2,7,8}};
  int i,j;
  for (i=o; i<hang; i++) {
     for (j=0; j<cot; j++)
       printf(" %d ", mt[i][j]);
     printf("\n");
   }
```

```
for (i=0; i<hang; i++) {
     for (j=0; j<cot; j++) {
       *(*(mt+i)+j)=*(*(mt+i)+j) +10;
       printf(" %d ", *(*(mt+i)+j);
     }
     printf("\n"); }
 }
```

Bài tập

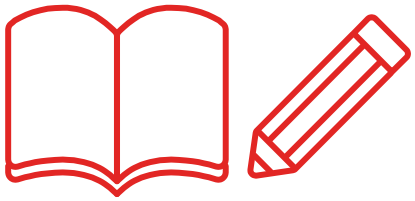


Viết chương trình tính tổng các phần tử mảng sử dụng con trỏ



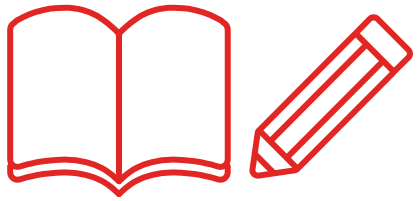
```
int main() {
 int numArray[6];
   int i, sum = 0;
   int *ptr = numArray;

   cout << "Nhap 6 phan tu: " << endl;

   for (i = 0; i < 6; i++)
      cin >> *(ptr+i);

   ptr = numArray;
   for (i = 0; i < 6; i++) {
      sum = sum + *ptr;
      ptr++;
   }
   cout << "Tong cac phan tu cua mang la: " << sum << endl;
       return(0);
}
```

Bài tập



Xây dựng một hàm tính độ dài của chuỗi sử dụng con trỏ.

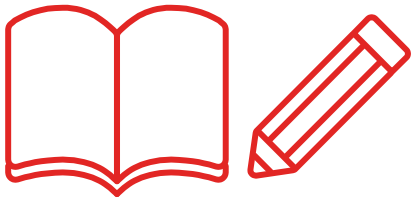
Bài tập



Xây dựng một hàm tính độ dài của chuỗi sử dụng con trỏ.

```
int string_ln(char*p) /* p=&str[0] */
{
    int count = 0;
    while (*p != '\0') {
        count++;
        p++;
    }
    return count;
}
```

Bài tập



Viết chương trình nhập vào n số nguyên, thực hiện (sử dụng con trỏ):

-Tính giá trị trung bình, giá trị min max của các phần tử trong mảng

-Sắp xếp các phần tử trong mảng theo thứ tự tăng dần

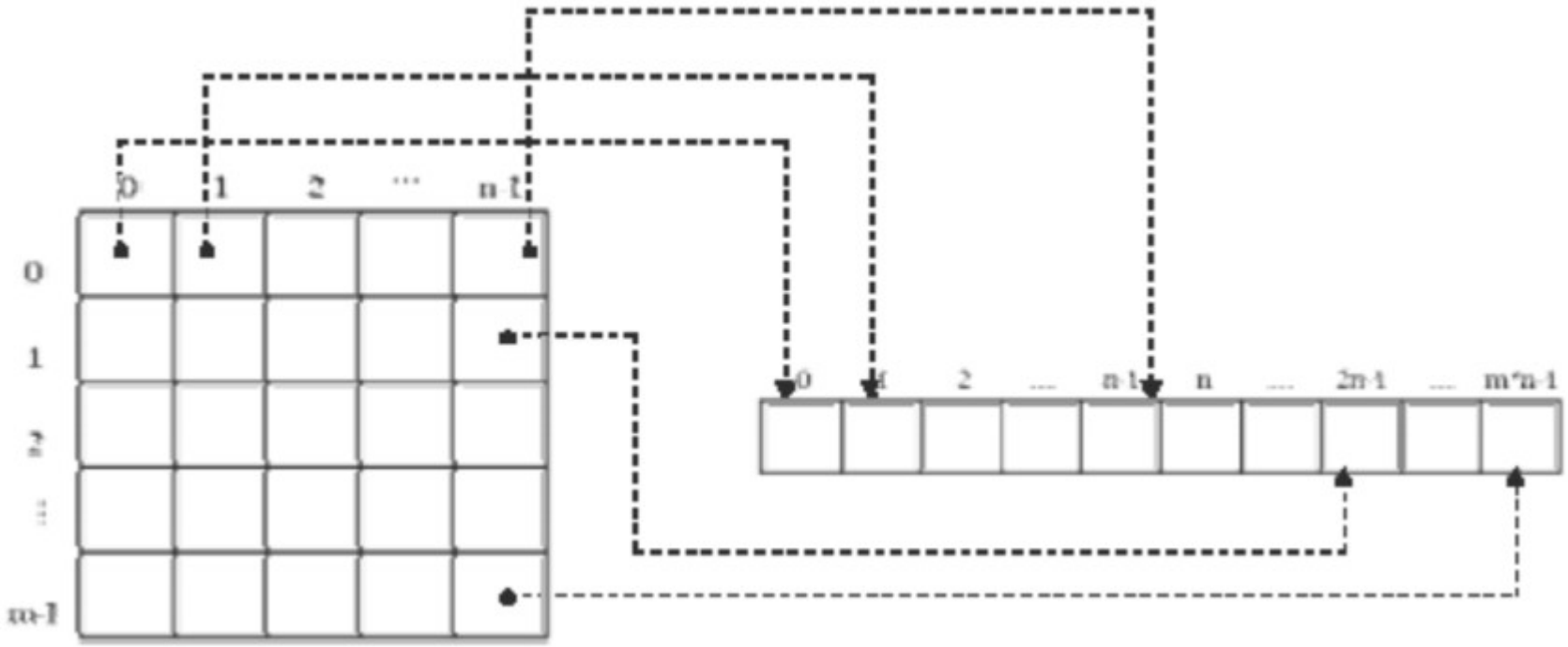
Ví dụ

```
#include <stdio.h>
int main() {
    int i,n; long total=100,x,*ds;
    printf(" Vao so ptu "); scanf(“%d”,&n);
    ds = new long [n];
    if (ds==NULL) exit(1);
    for (i=0;i<n;i++){
        printf("\n Vao so thu  %d : ", i+1 );
        scanf(“%d”,&ds[i] );
    }
    printf(“Danh sach cac so : \n”);
    for (i=0;i<n;i++)   printf(“%d”,ds[i]);
        delete []ds;
        return 0;
    }
```

Bộ nhớ động cho mảng 2 chiều

- Cách 1: Biểu diễn mảng 2 chiều thành mảng 1 chiều
- Gọi X là mảng hai chiều có kích thước m dòng và n cột. A là mảng một chiều tương ứng, khi đó

```
X[i][j] = A[i*n+j]
```



Bộ nhớ động cho mảng 2 chiều

- Cách 2: Dùng con trỏ của con trỏ
- Ví dụ: Với mảng số nguyên 2 chiều có kích thước là R * C ta khai báo như sau:

```
int **mt;
mt = new int *[R];
int *temp = new int[R*C];
for (i=0; i< R; ++i) {
   mt[i] = temp;
   temp += C;
}
```

- Để giải phóng:

```
delete [] mt[0];
delete [] mt;
```

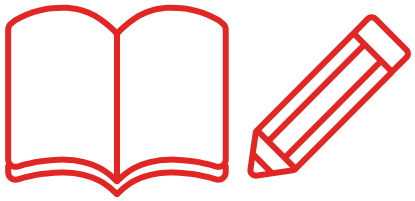
Bộ nhớ động cho mảng 2 chiều

- Ví dụ khác để cấp phát động cho mảng hai chiều chứa các số thực `float`

```
// Khởi tạo ma trận với
// R hàng và C cột
 float  ** M = new float *[R];
 for (i=0; i<R;i++)
    M[i] = new float[C];
// Dùng M[i][j] cho
// các phần tử của ma trận
```

```
// Giải phóng
 for(i=0; i<R;i++)
    // Giải phóng các hàng
    delete []M[i];
 delete []M;
```

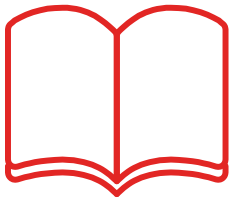
Bài tập



Viết chương trình cộng hai ma trận với dữ liệu mỗi ma trận được cấp phát bộ nhớ động theo hai cách:

1) Sử dụng con trỏ
2) Sử dụng con trỏ trỏ đến con trỏ

Cộng hai ma trận với mỗi ma trận được cấp phát động



```
#include <iostream.h>
#include <conio.h>
int main()
{
    int M,N;
    int *A = NULL,
        *B = NULL,
        *C = NULL;
    cout << "Nhap so dong cua
      ma tran:"; cin>>M;
    cout << "Nhap so cot cua
      ma tran:"; cin>>N;
```

```
    //Cấp phát vùng nhớ cho ma trận
A
    if(!AllocMatrix(&A,M,N))
    {
       cout << "Khong con du bo
nho! "
            << endl;
       return 1;
    }
    //Cấp phát vùng nhớ cho ma trận
B
    if(!AllocMatrix(&B,M,N))
    {
       cout << "Khong con du bo
nho! "
            << endl;
     FreeMatrix(A);
       return 1;
```

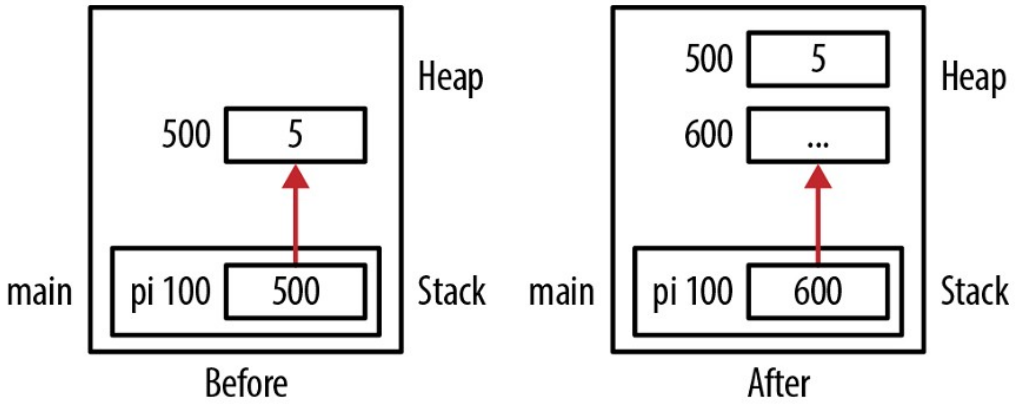
```
 //Cấp phát vùng nhớ
    //cho ma trận C
if (!
AllocMatrix(&C,M,N))
{
       cout << "Khong con
du
                  bo nho!"
             <<endl;
       //Giải phóng vùng
nhớ A
       FreeMatrix(A);
       //Giải phóng vùng
nhớ B
       FreeMatrix(B);
    return 1;
 }
cout << "Nhap ma tran
           thu 1"<< endl;
  InputMatrix(A,M,N,'A');
cout << "Nhap ma tran
           thu 2"<<endl;
InputMatrix(B,M,N,'B');
clrscr();
cout<<"Ma tran thu 1"<<endl;
 DisplayMatrix(A,M,N);
 cout<<"Ma tran thu
2"<<endl;
 DisplayMatrix(B,M,N);
 AddMatrix(A,B,C,M,N);
 cout <<"Tong hai ma tran "
          <<endl;
 DisplayMatrix(C,M,N);
 //Giải phóng vùng nhớ A
    FreeMatrix(A);
 //Giải phóng vùng nhớ B
    FreeMatrix(B);
    //Giải phóng vùng nhớ C
 FreeMatrix(C);
 return 0;
}
```

```
//Cộng hai ma trận
void AddMatrix(int *A,int *B,int*C,int M,int N)
{
   for(int I=0;I<M*N;++I)
   C[I] = A[I] + B[I];
}

   //Cấp phát vùng nhớ cho ma trận
int AllocMatrix(int **A,int M,int N)
{
   *A = new int [M*N];
   if (*A == NULL)
   return 0;
   return 1;
}

   //Giải phóng vùng nhớ
void FreeMatrix(int *A)
{
   if (A!=NULL)
   delete [] A;
}
```

```
//Nhập các giá trị của ma trận
void InputMatrix(int *A,int M,int N,char Symbol)
{
    for(int I=0;I<M;++I)
    for(int J=0;J<N;++J)
   {
       cout<<Symbol<<"["<<I<<"]["<<J<<"]=";
       cin>>A[I*N+J];
   }
}
//Hiển thị ma trận
void DisplayMatrix(int *A,int M,int N)
{
    for(int I=0;I<M;++I)
    {
          for(int J=0;J<N;++J)
          {
                        //canh le phai voi chieu dai 7 ky
tu
                   out.width(7);
              cout<<A[I*N+J];
          }
          cout<<endl;
    }
}
```

Mở rộng: các vấn đề với cấp phát bộ nhớ động

- Memory Leaks: Rò rỉ bộ nhớ xảy ra khi bộ nhớ được phân bổ không bao giờ được sử dụng lại nhưng không được giải phóng.

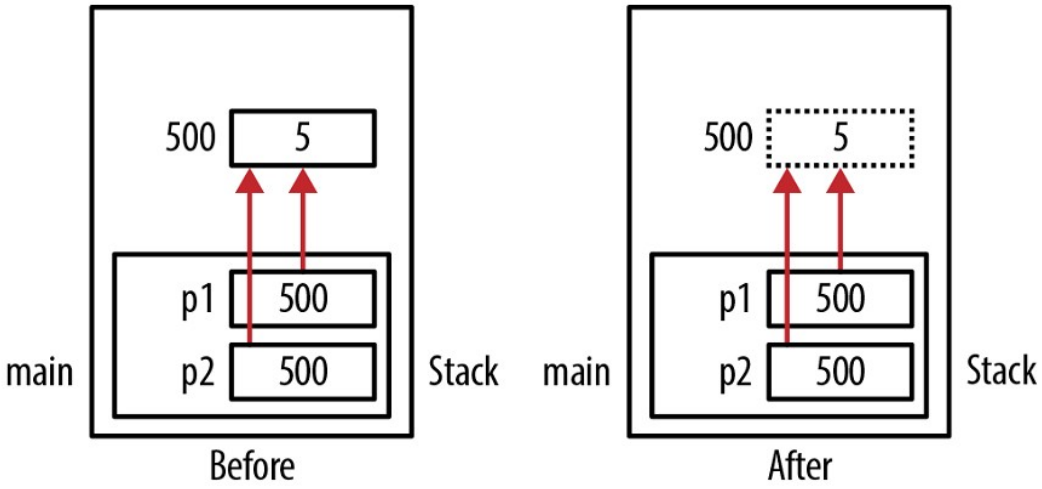



```
int *pi = (int*) malloc(sizeof(int));
*pi = 5;
...
pi = (int*) malloc(sizeof(int));
```

Mở rộng: các vấn đề với cấp phát bộ nhớ động

- Double Free: giải phóng một khối bộ nhớ hai lần

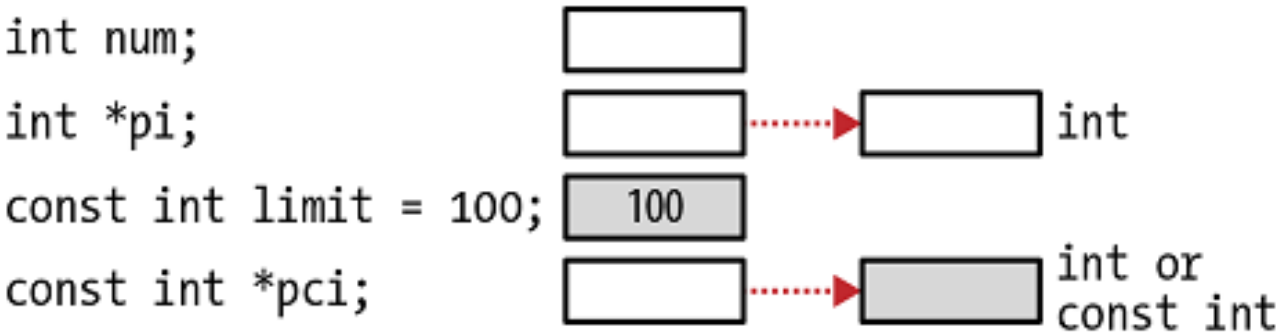
```
p1 = (int*) malloc(sizeof(int));
int *p2 = p1;
free(p1);
...
free(p2);
```



- Dangling Pointers: nếu một con trỏ vẫn tham chiếu bộ nhớ gốc sau khi nó được giải phóng, nó được gọi là con trỏ lơ lửng. Con trỏ không trỏ đến một đối tượng hợp lệ.

Mở rộng: Con trỏ đến hằng (Pointers to a constant)

- Một con trỏ có thể được xác định để trỏ đến một hằng số. Điều này có nghĩa là con trỏ không thể được sử dụng để sửa đổi giá trị mà nó đang tham chiếu.
  - `int num = 5;`
  - `const int limit = 500;`
  - `int *pi;  // Pointer to an integer`
  - `const int *pci; // Pointer to a constant integer`
  - `pi = &num;`
  - `pci = &limit;`



Mở rộng: Con trỏ hằng (Constant pointers)

- Con trỏ là hằng không thể thay đổi, giá trị mà nó trỏ đến có thể thay đổi
  - `int num;`
  - `int *const cpi = &num;`
  - Các lệnh sau là hợp lệ
    - `*cpi = limit;`
    - `*cpi = 25;`

Truyền tham chiếu

- Hàm nhận tham số là con trỏ

```
void Swap(int *X, int *Y) {
   int Temp = *X;
   *X = *Y;
   *Y = Temp;
}
```

- Để hoán đổi giá trị hai biến A và B

```
Swap(&A, &B);
```

Truyền tham chiếu

- Hàm nhận tham số là tham chiếu

```
void Swap(int &X, int &Y){
   int Temp = X;
   X = Y;
   Y = Temp;
}
```

- Để hoán đổi giá trị hai biến A và B

```
Swap(A, B);
```

Truyền tham chiếu

Khi một hàm trả về một tham chiếu, chúng ta có thể gọi hàm ở phía bên trái của một phép gán.

```
#include <iostream.h>
int X = 4;
int & MyFunc(){
   return X;
}
int main(){
   Cout << "X=" << X << endl;
   Cout << "X=" << MyFunc() << endl;
   MyFunc() = 20; // ~X=20
   Cout << "X=" << X << endl;
   return 0;
}
```

Đa năng hoá toán tử

- Định nghĩa lại chức năng của các toán tử đã có sẵn
  - Thể hiện các phép toán một cách tự nhiên hơn
- Ví dụ: thực hiện các phép cộng, trừ số phức
  - Trong C: Cần phải xây dựng các hàm `AddSP()`, `TruSP()`
  - Không thể hiện được phép cộng và trừ cho các biểu thức như: `a=b+c-d+e+f-h-k`

```
#include <stdio.h>
struct  SP {
   double real;
   double img;
};
SP SetSP(double real, double img);
SP AddSP(SP C1,SP C2);
SP SubSP(SP C1,SP C2);
void DisplaySP(SP C);
int main(void) {
   SP C1,C2,C3,C4;
   C1 = SetSP(1.0,2.0);
   C2 = SetSP(-3.0,4.0);
   cout << "\nSo phuc thu nhat:";   DisplaySP(C1);
   cout << "\nSo phuc thu hai:";    DisplaySP(C2);
   C3 = AddSP(C1,C2);
   C4 = SubSP(C1,C2);
   cout << "\nTong hai so phuc nay:"; DisplaySP(C3);
   cout << "\nHieu hai so phuc nay:"; DisplaySP(C4);
   return 0;
}
```

```
SP SetSP(double real,double img) {
   SP tmp;
   tmp.real = real; tmp.img = img;
   return tmp;
}
SP AddSP(SP C1,SP C2) {
   SP tmp;
   tmp.real = C1.real + C2.real;
   tmp.img = C1.img + C2.img;
   return tmp;
}
SP SubSP(SP C1,SP C2) {
   SP tmp;
   tmp.real = C1.real - C2.real;
   tmp.img = C1.img - C2.img;
return tmp;
}
void DisplaySP(SP C) {
   cout << C.real << " i "  << C.img;
}
```

C++

- C++ cho phép chúng ta có thể định nghĩa lại chức năng của các toán tử đã có sẵn một cách tiện lợi và tự nhiên. Điều này gọi là đa năng hóa toán tử.
- Một hàm định nghĩa một toán tử có cú pháp sau:

```
data_type operator operator_symbol ( parameters )
  { ..................................
}
```

Trong đó:

- `data_type`: *Kiểu trả về.*
- `operator_symbol`: *Ký hiệu của toán tử.*
- `parameters`: *Các tham số (nếu có).*

```
#include <iostream.h>
typedef struct SP { double real; double img;} SP;
SP SetSP(double real, double img);
void DisplaySP(SP C);
SP operator + (SP C1, SP C2);
SP operator - (SP C1, SP C2);
int main() {
   SP C1,C2,C3,C4;
   C1 = SetSP(1.1,2.0);
   C2 = SetSP(-3.0,4.0);
   cout << "\nSo phuc thu  nhat:"; DisplaySP(C1);
   cout << "\nSo  phuc thu hai:";  DisplaySP(C2);
   C3 = C1 + C2;
   C4 = C1 - C2;
   cout << "\nTong hai so phuc nay:"; DisplaySP(C3);
   cout << "\nHieu hai so phuc nay:"; DisplaySP(C4);
    return 0;
}
```

```
SetSP(double real,double img) {
   SP tmp;
   tmp.real = real; tmp.img = img;
   return tmp;
}
SP operator + (SP C1,SP C2) {
   SP tmp;
   tmp.real = C1.real + C2.real;
   tmp.img = C1.img + C2.img;
   return tmp;
}
SP operator - (SP C1,SP C2) {
   SP tmp;
   tmp.real = C1.real - C2.real;
   tmp.img = C1.img - C2.img;
   return tmp;
}
void DisplaySP(SP C) {
    cout << "(" << C.real << "," << C.img << ")";
}
```

Ví dụ sử dụng con trỏ hàm

```
void swapValue(int &value1, int &value2){
    int temp = value1;
    value1 = value2;
    value2 = temp;
}
int main(){
    void(*pSwap) (int &, int &) = swapValue;
    int a = 1, b = 5;
    cout << "Before: " << a << " " << b << endl;
    (*pSwap)(a, b);
    cout << "After:  " << a << " " << b << endl;
    return 0;
}
```

Sắp xếp dãy số

```
bool ascending(int left, int right){
    return left > right;
}
bool descending(int left, int right){
    return left < right;
}
void selectionSort(int *arr, int length, bool (*comparisonFunc)(int,
int)){
    for (int i_start = 0; i_start < length; i_start++) {
        int minIndex = i_start;
        for (int i_current = i_start + 1; i_current < length; i_current++){
     if (comparisonFunc(arr[minIndex], arr[i_current]))     {
         minIndex = i_current;
     }
        }
        swap(arr[i_start], arr[minIndex]); // std::swap
    }
}
```

Sắp xếp dãy số

```
int main() {
    int arr[] = { 1, 4, 2, 3, 6, 5, 8, 9, 7 };
    int length = sizeof(arr) / sizeof(int);
    cout << "Before sorted: ";
    printArray(arr, length);
    selectionSort(arr, length, descending);
    cout << "After sorted:  ";
    printArray(arr, length);
    return 0;
}
```

Sắp xếp dãy số

```
int main() {
    int arr[] = { 1, 4, 2, 3, 6, 5, 8, 9, 7 };
    int length = sizeof(arr) / sizeof(int);
    cout << "Before sorted: ";
    printArray(arr, length);
    selectionSort(arr, length, ascending);
    cout << "After sorted:  ";
    printArray(arr, length);
    return 0;
}
```

Khái quát hóa hàm (Function templates)

```
#include <iostream>
using namespace std;
template <typename T>
T maxval(T x, T y){
    return (x > y) ? x : y;
}
int main() {
    int i = maxval(3, 7); // returns 7
    cout << i << endl;
    double d = maxval(6.34, 18.523); // returns 18.523
    cout << d << endl;
    char ch = maxval('a', '6'); // returns 'a'
    cout << ch << endl;
    return 0;
}
```

Hàm nặc danh - cú pháp lambda

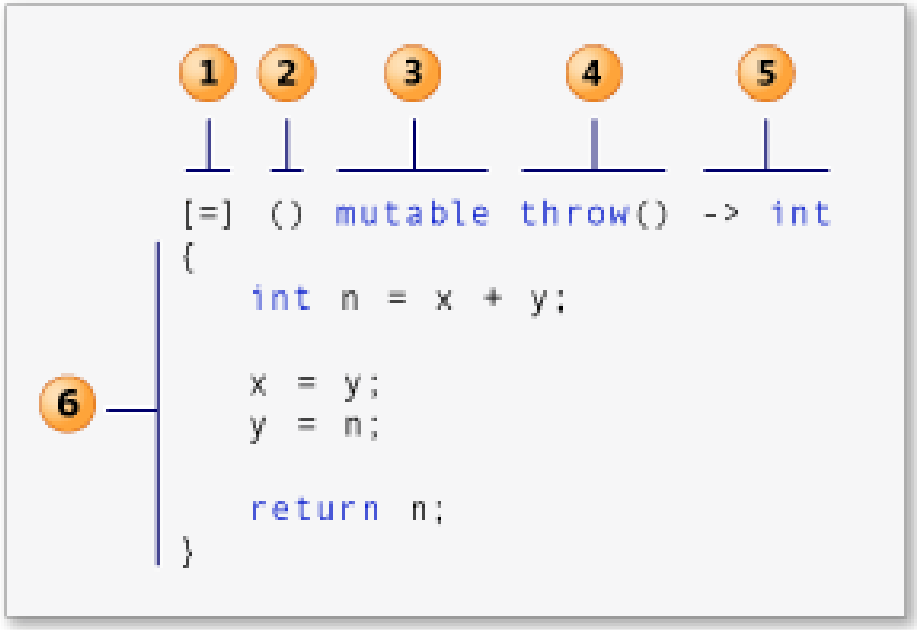
```
#include <iostream>
using namespace std;

void stdio_doing(int n) {
    n = n + 10;
    cout << n << " ";
}

void for_each (int *arr, int n, void (*func)(int a)){
    for (int i = 0; i < n; i++) {
        func(*(arr + i));
    }
}

int main(){
    int arr[] ={1, 2, 3, 4, 5} , n = 5;
    for_each(arr, n, stdio_doing);
    return 0;
}
```

Hàm nặc danh - cú pháp lambda

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

void for_each (int *arr, int n, void (*func)(int a)){
    for (int i = 0; i < n; i++)  {
        func(*(arr + i));
    }
}

int main(){
    int arr[] ={1, 2, 3, 4, 5} , n = 5;
    for_each(arr, n, [] (int a){
        a = a + 10;
        cout << a << " ";
    });
    return 0;
}
```

Lợi ích của lambda là không nhất thiết phải khai báo tên hàm ở nơi khác, mà có thể tạo ngay một hàm sử dụng tại chỗ (thường là các tác vụ nhỏ - dùng 1 lần hay chỉ có 1 chỗ gọi hàm đó).

Hàm nặc danh - cú pháp lambda



(1) Mệnh đề bắt giữ (capture clause)

(2) Danh sách tham số

(3) Tính bền vững của lambda

(4) Ngoại lệ có thể xảy ra trong lambda.

(5) Kiểu trả về của lambda

(6) Phần thân lambda

Hàm nặc danh - cú pháp lambda

Mệnh đề bắt giữ (capture clause)

[ ]	Không sử dụng bất kỳ biến bên ngoài thân hàm *e.g.* **[ ](int i){ return i+j; }** *//error vì j là biến bên ngoài và không được captured*
[&]	Được sử dụng các biến bên ngoài qua tham chiếu (**by reference**). *e.g.* **[&](int i){ j++; return i+j; }** *//changes made to j is reflected upon return*
[=]	Được sử dụng các biến bên ngoài, nhưng là dạng sao chép giá trị của biến đó (**by value**) *e.g.* **[=](int i){ return i+j; }** *//j không được thay đổi trong thân hàm*
[&,j]	Bất kỳ biến bên ngoài nào được captured qua tham chiếu, ngoại trừ biến j được captured qua giá trị. *e.g.* **[&,j](int i){ j++; k++; return i+j+k; }** *//j được tăng cục bộ. k tăng giá trị cho cả bên ngoài hàm*
[=,j]	Bất kỳ biến bên ngoài nào được captured qua giá trị, ngoại trừ biến j được captured qua tham chiếu. *e.g.* **[ ](int i){ j++; k++; return i+j+k; }**
[this]	Cho phép sử dụng this (OOP) như 1 bản sao
[&,&j]	Error. j đã được ghi lại theo tham chiếu theo mặc định
[=,this]	Error. khi sử dụng **=**, **this** được captured theo mặc được
[i,i]	Error. i bị lặp lại
[&this]	Error. không thể lấy địa chỉ của **this**
[42]	Error. Phải cung cấp **tên biến** hoặc **&** hoặc **=** hoặc ***this***

Ví dụ

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

int main(){
    int m = 0;
    int n = 0;
    auto func = [&, n] (int a) mutable {
        m = ++n + a;
    };
    func(4);
    cout << m << endl << n << endl;
}
```

Kết quả:

5

0

Bài tập

- Xây dựng cấu trúc thời gian Time chứa các thông tin giờ, phút và đa năng hóa các toán tử **+**, **-** cho cấu trúc thời gian này.

Bài tập

- Xây dựng hàm cấp phát động một ma trận có kích thước **n x m**.

Bài tập

- Xây dựng hàm cấp phát động một ma trận có kích thước **n x m**

Cách 1:

```
int** khoiTao1(int n, int m){
    int** a = new int *[n];
    for(int i = 0; i<n; i++){
        a[i] = new int[m];
    }
    return a;

}
```

```
int **test;
test = khoiTao1(n,m);
```

Bài tập

- Xây dựng hàm cấp phát động một ma trận có kích thước n x m

Cách 2:

```
void khoiTao2(int ***a, int n, int m){
    *a = new int *[n];
     for(int i = 0; i<n; i++){
         (*a)[i] = new int[m];
     }
}
```

```
int **test;
khoiTao2(&test,n,m);
```

Bài tập

- Xây dựng hàm cấp phát động một ma trận có kích thước n x m

Cách 3:

```
void khoiTao3(int **&a, int n, int m){
        a = new int *[n];
        for(int i = 0; i<n; i++){
            a[i] = new int[m];
        }
}
```

```
int **test;
khoiTao3(test,n,m);
```

Một vài ví dụ tối ưu mã C, C++

```
switch ( queue ) {
    case 0 : letter = 'W'; break;
    case 1 : letter = 'S'; break;
    case 2 : letter = 'U'; break;
}
Hoặc có thể là :
if ( queue == 0 )
    letter = 'W';
else if ( queue == 1 )
    letter = 'S';
```

```
static char *classes="WSU";
letter = classes[queue];
```

Một vài ví dụ tối ưu mã C, C++

`(x >= min && x < max)` có thể chuyển thành

`(unsigned)(x - min) < (max - min)`

Giải thích:

int: -2^{31} … $2^{31} - 1$

unsigned: 0 … $2^{32} - 1$

Nếu `x - min >= 0`: Hai biểu thức trên tương đương

Nếu `x - min <= 0`:

`(unsigned) (x - min) =` 2^{32} `+ x - min`

`>=` 2^{31} `> max - min`

Một vài ví dụ tối ưu mã C, C++

```
int fact1_func (int n) {
    int i, fact = 1;
    for (i = 1; i <= n; i++) fact *= i;
    return (fact);
}
int fact2_func(int n) {
    int i, fact = 1;
    for (i = n; i != 0; i--) fact *= i;
    return (fact);
}
```

`fact2_func` nhanh hơn, vì phép thử != đơn giản hơn <=

Số thực dấu phẩy động

- So sánh:

```
x = x / 3.0;
```

và

```
x = x * (1.0/3.0) ;
```

(biểu thức hằng được thực hiện ngay khi dịch)

- Hãy dùng `float` thay vì `double`
- Tránh dùng `sin, exp` và `log` (chậm gấp 10 lần *)
- Dùng `x * 0.5` thay vì `x / 2.0`
- `x+x+x` thay vì `x*3`
- Mảng 1 chiều nhanh hơn mảng nhiều chiều
- Tránh dùng đệ quy

Ví dụ

Tính giai thừa của n

- Định nghĩa không đệ quy n!

  n! = n * (n-1) * … * 1

- Định nghĩa đệ quy:

  n! = 1 nếu n=0

  n * (n-1)! nếu n>0

Mã C++

```
int factorial(int n) {
   if (n==0)  return 1;
   else
      return (n * factorial(n -
1));
}
```

Ví dụ

Tính

S(n) = 1/(1*2) + 1/(2*3) + … + 1/(n*(n+1))

S(n) = 1/2 khi n==1

= S(n-1)+1/(n*(n+1))

```
float  S(int n) {
  if ( n==1) return 0.5;
  else return S(n-1)+1.0/(n*(n+1));
}
```

Ví dụ

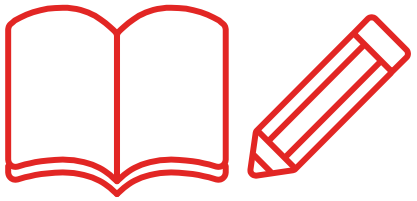
Tính tổng các giá trị của dãy số H(n), biết

H(n) = n khi n<3

= 2*H(n-1)*H(n-2) khi n>2

```
long  H(int n) {
   if (n<3) return n;
   else  return 2*H(n-1)*H(n-2);
}

long Tong(int n) {
   long tg=0;
   for( int i=1; i<=n;i++)
     tg+=H(i);
   return tg;
}
```

Bài tập



Tính X(n) với

- **$X_o = 1$**
- **$X_n = canba(n)*X_o + canba(n-1)*X_1 +...+ canba(1)*X_{n-1}$**

- canba(n): hàm tính căn bậc ba của n

Ví dụ

X(n) = 1,2,3,5,11,41……
Y(n) = 1,1,2,6,30,330 …..

```
void main() {
   int n;
   printf("\n Nhap n = ");
   scanf("%d",&n);
   printf( "\n  X = %d " ,X(n));
   printf( "\n Y = %d " , Y(n));
   getch();
}

long Y(int n); //prototype cua ham y
long X(int n) {
   if(n==0)
      return 1;
   else
      return X(n-1) + Y(n-1);
}

long Y(int n) {
   if(n==0)
      return 1;
   else
      return X(n-1)*Y(n-1);
}
```

Bài tập 1

- Xây dựng hàm đệ quy in đảo ngược số nguyên dương n

Bài tập 2

- Xây dựng hàm đệ kiểm tra xem một mảng các số nguyên có đối xứng hay không

Bài tập 2

- Phân tích
  - **Yêu cầu**: kiểm tra tính đối xứng của một mảng từ các phần tử từ hai đầu của mảng
  - **Kích thước đệ quy**: số phần tử của mảng
  - **Điểm dừng đệ quy**: phát hiện hai phần tử nằm tại các vị trí đối xứng không cùng giá trị hoặc kiểm tra hết các phần tử đều đối xứng
  - **Trường hợp tổng quát**: khi phần tử nằm tại các vị trí i và j đối xứng có cùng giá trị thì kiểm tra tính đối xứng của mảng con còn lại từ i+1 đến j-1

Bài tập 3

- Xây dựng hàm đệ quy kiểm tra một mảng có giá trị các phần tử theo thứ tự tăng dần (hoặc giảm dần) không?

Bài tập 4

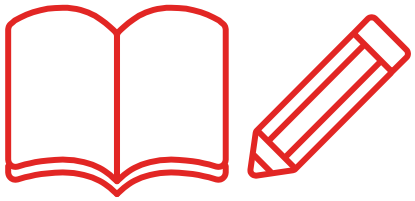
- Một palindrome là một từ hay một câu, đọc xuôi ngược đều giống như nhau. Viết chương trình, với giải thuật đệ quy, đọc một dòng từ bàn phím vào và báo cho biết đó có phải là palindrome không.

Bài tập 5

- Xây dựng hàm đệ quy đảo ngược một mảng các số nguyên?

- Xây dựng hàm đảo ngược một mảng các số nguyên không dùng đệ quy?

Bài tập



- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau

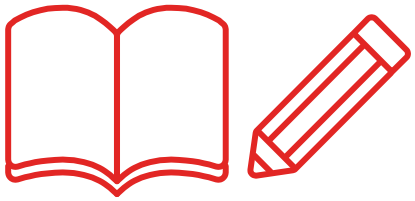
1,2,3,6,11,20,37,68,125...

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau :

1,2,3,7,13,23,43,79,145...

Biết rằng số phần tử của dãy số luôn >= 5

Bài tập



- Viết hàm đệ quy tính các phần tử của dãy số sau với n phần tử (n>4), rồi tính tổng các phần tử của dãy số

1,2,3,4,4,5,8,11,12,14,21,30,35,40,56,.....

1,2,3,4,6,9,14,21,32,48,73,110,167,252,...

- Khử đệ quy với hàm tính giai thừa

```
int FAC ( int n ) {
   int k = 0;
   int F = 1;
   while ( k < n ) F = ++k * F;
   return (F);
}
```

- Khử đệ quy với hàm tính S(n)

```
int S ( int n ) {
   int k = 1 , tg = 1 ;
   while ( k < n ) {
      k ++ ;
      if (k%2 == 1)  tg + = 2 * k -1;
      else tg -= 2 * k + 1 ;
       }
     return ( tg ) ;
}
```

Ví dụ
Tìm ước chung lớn nhất

Giải thuật đệ quy

```
int USCLN(int m, int
n) {
    if (n == 0)
return m;
    else USCLN(n, m %
n);
}
```

- X là(m , n)
- P(X) là USCLN(m ,n)
- B(X) là n == 0
- D(X) là lệnh return m
- A(X) là lệnh rỗng
- f(X) là f(m,n) = (n , m mod n)

Khử đệ quy

```
int USCLN(int m , int
 n ) {
  while(n != 0 ) {
     int sd = m % n ;
     m = n ;
      n = sd ;
  }
 return (m) ;
}
```

Ví dụ
Bài toán Tháp Hà Nội

Đệ quy

```
THN(n,X,Y,Z) ≡ if(n > 0)
{
    THN (n - 1, X, Z, Y);
    Move (X, Z );
    THN (n - 1, Y, X, Z);
}
```

Trong đó

- Biến X là bộ (n,X,Y,Z)
- C(X) là n<=0
- D(X), A(X) là rỗng
- B(X) = B(n,X,Y,Z) là move(X, Z)
- f(X) = f(n,X,Y,Z) = (n-1,X,Z,Y)
- g(X) = g(n,X,Y,Z) = (n-1,Y,X,Z)

Khử đệ quy

```
THN {
  Create_Stack (S) ;
  Push (S ,(n,X,Y,Z,1)) ;
  Repeat
    While (n > 0) do begin
      Push (S ,(n,X,Y,Z,2)) ;
      n = n - 1;
      Swap (Y,Z) ;
    end ;
    Pop (S,(n,X,Y,Z,k)) ;
    if ( k <> 1 ) then begin
      Move (X ,Z ) ;
      n = n - 1 ;
      Swap (X,Y) ;
    end ;
  until ( k == 1 ) ;
}
```

Ví dụ

Cho dãy số

1,2,3,7,14,27,55,110,219....

Viết hàm đệ quy tính số hạng thứ n của dãy số (n > 2 nhập từ bàn phím), rồi tính tổng các số hạng của dãy

Sau đó, khử đệ quy chương trình trên

Công thức tổng quát

S(n) = n, khi n <4

= S(n-1)+S(n-2)+ 2 * S(n-3), khi n>3

Ví dụ

Chương trình đệ quy

```
int S(int n) {
  if (n<4)
     return n;
  else
     return S(n-1)+S(n-2)+S(n-3)*2;
}

int main() {
  int n,i,Tong=0;
  printf("\n Vao n :");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
    Tong+=S(i);
  printf("\n Tong day so = %d",Tong);
}
```

Ví dụ

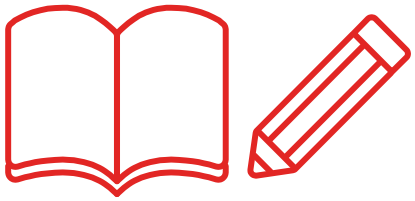

```
int main()
{
      int i,n,T=6; F1,F2,F3,F;
      printf("\n Vao n: ");
      scanf("%d",&n);
      if (n==1) T=1;
      else if (n==2) T=3;
      else if (n==3) T=6;
      else {
            F1=1; F2=2; F3=3;
            for(i=4;i<=n;i++)
            {
                  F=2*F1+F2+F3;
                  T+=F;
                  F1=F2; F2=F3; F3=F;
            }
      printf("\n Tong = %d ",T);
}
```

Ví dụ

Khử đệ quy (dùng mảng)

```
int main()
{
      int i,n,T=6, F[4]={1,2,3,7};
      printf("\n Vao n : ");
      scanf("%d",&n);
      if (n==0) T=1;
      else if (n==1) T=3;
      else if (n==2) T=6;
      else {
        for(i=3;i<=n;i++)
        {
          F[i%4] = F[(i-1)%4]+F[(i-
2)%4]
                   +2 * F[(i-3)%4];
           T+=F[i%4];
        }
      }
      printf("\n Tong = %d ",T);
      getch();
```

Bài tập



- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau, sau đó viết chương trình dưới dạng không đệ quy:

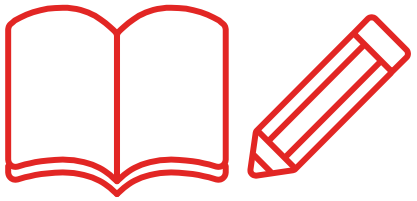
T=1+2+3+6+11+20+37+68+125+...

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau, sau đó viết chương trình dưới dạng không đệ quy:

T=1+2+3+7+13+23+43+79+145+...

Biết rằng số phần tử của dãy số luôn >= 5

Bài tập



- Viết hàm đệ quy tính các phần tử của dãy số sau với n phần tử (n>4), rồi tính tổng các phần tử của dãy số

1,2,3,4,4,5,8,11,12,14,21,30,35,40,56,.....

1,2,3,4,6,9,14,21,32,48,73,110,167,252,...

- Sau đó viết lại toàn bộ chương trình tính tổng dãy số trên mà không dùng đệ quy

Bài tập

- Xây dựng hàm đệ quy đảo ngược các phần tử trong một danh sách liên kết đơn?

- Xây dựng hàm đảo ngược các phần tử trong một danh sách liên kết đơn không dùng đệ quy?

Hướng dẫn

- Đệ quy?
  - Điểm neo / dừng đệ quy: NULL / chỉ có 1 nút
  - Thao tác đệ quy:
    - 1. Chia danh sách thành hai phần – node đầu tiên và phần còn lại của danh sách
    - 2. Thực hiện đảo ngược đệ quy cho phần còn lại của danh sách
    - 3. Kết nối phần còn lại của danh sách với node đầu tiên
    - 4. Thay đổi con trỏ head

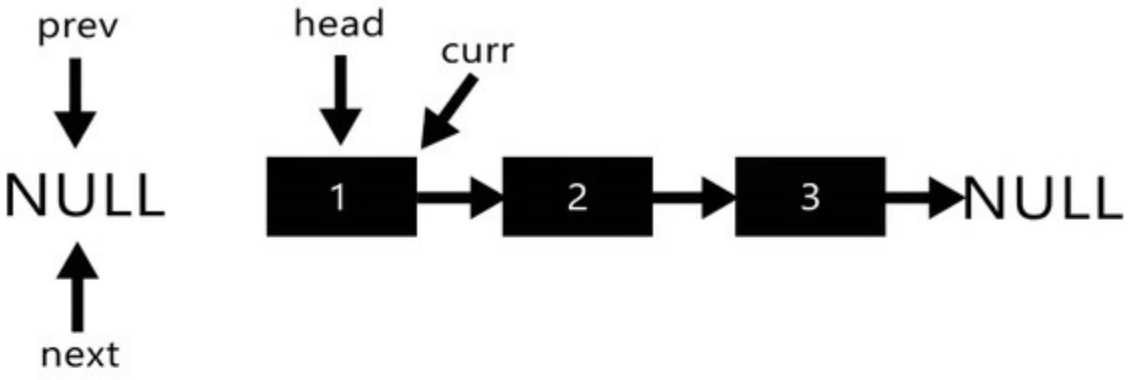
Hướng dẫn

Giải pháp sau đây có đúng không?

```
Node* reverse(Node* head) {
    if (head == NULL || head->next == NULL)
        return head;

    Node* rest = reverse(head->next);
    head->next->next = head;

    head->next = NULL;
    return rest;
}
```

Hướng dẫn

- Khử đệ quy?
  - dùng vòng lặp




```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

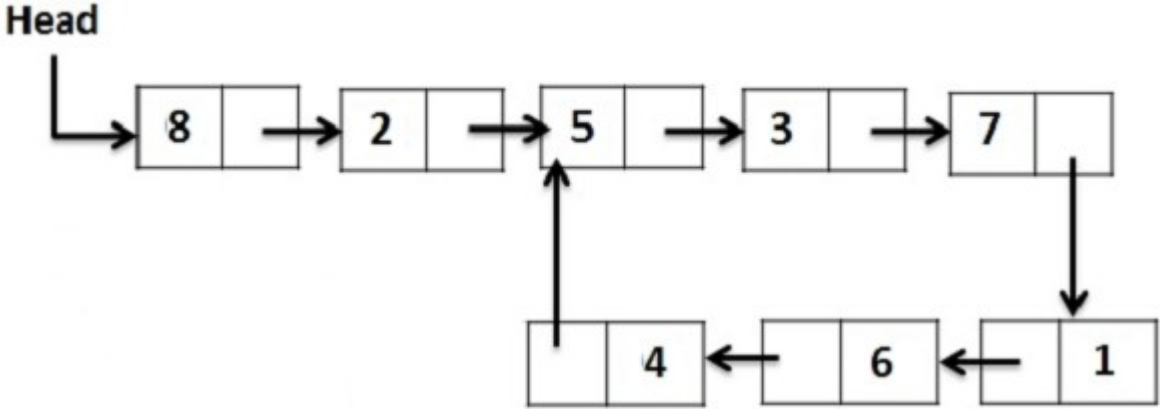
Hướng dẫn

Giải pháp sau đây có đúng không?

```
void reverse() {
    Node* current = head;
    Node *prev = NULL, *next = NULL;

    while (current != NULL) {
      next = current->next;

      current->next = prev;

      prev = current;
      current = next;
    }
    head = prev;
  }
```

Bài tập: Kiểm tra chu trình trong danh sách nối đơn

- Định nghĩa một danh sách liên kết đơn gồm các giá trị nguyên, hãy viết chương trình xác định danh sách liên kết có chứa chu trình hay không.



- Phân tích
  - Trong ví dụ này có tồn tại một chu trình:
    8 ☾ 2 ☾ 5 ☾ 3 ☾ 7 ☾ 1 ☾ 6 ☾ 4 ☾ 5 ☾ 3 ☾ 7 ☾ 1 ☾ 6 ☾ 4 ☾ 5 ☾ ...
  - Tồn tại chu trình giữa phần tử có giá trị 5 và 4

Bài tập: Kiểm tra chu trình trong danh sách nối đơn

- Thuật toán: Floyd's Cycle-Finding Algorithm
  - Đặt hai con trỏ: một con trỏ di chuyển chậm và một con trỏ di chuyển nhanh
    - Con trỏ chậm: khởi tạo trỏ vào phần tử đầu danh sách (Head), mỗi lần lặp sẽ di chuyển sang phần tử tiếp theo (1 bước)
    - Con trỏ nhanh: khởi tạo trỏ vào phần tử đầu danh sách (Head), mỗi lần lặp sẽ di chuyển sang phần tử kế tiếp của phần tử tiếp theo (2 bước)
  - Nếu sau một số lần lặp hữu hạn mà cả hai con trỏ cùng trỏ vào một nút ☾ danh sách tồn tại chu trình
  - Ngược lại nếu con trỏ nhanh nhận giá trị null ☾ danh sách không có chu trình

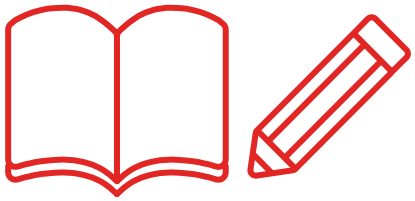
Bài tập: Kiểm tra chu trình trong danh sách nối đơn

```
int kiemtrachutrinh(Node *head){
    Node* slow = head, *fast = head;   //khởi tạo 2 con trỏ
    while(slow && fast && fast->next){
        slow = slow->next;      //di chuyển con trỏ chậm
        fast = fast->next->next;//di chuyển con trỏ nhanh
        if(fast==slow){     //nếu có chu trình
            return 1;
        }
    }
    return 0;           //nếu không có chu trình
}
```

Bài tập 2

- Xây dựng chương trình khai báo một danh sách liên kết đơn chứa các trị nguyên.
- Thực hiện: tách danh sách này thành các danh sách liên kết đơn con, mỗi danh sách liên kết đơn con chứa các phần tử là các giá trị nguyên tăng dần.
  - Ví dụ: danh sách đơn: 1 2 3 2 3 4 5 4 6 sẽ tác thành 3 danh sách con: “1 2 3”, “2 3 4 5” và “4 6”

Bài tập



Sử dụng **danh sách móc nối kép với nút đầu giả**, xây dựng bài toán quản lý điểm SV đơn giản, với các chức năng sau

1. Nhập dữ liệu vào danh sách
2. Hiển thị dữ liệu 1 lớp theo thứ tự tên
3. Tìm kiếm kết quả theo tên
4. Sắp xếp theo điểm trung bình

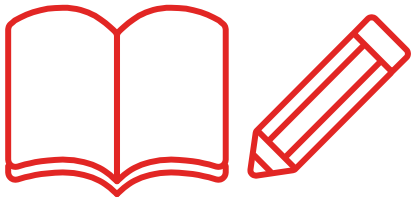
Với thông tin về mỗi sinh viên được định nghĩa trong cấu trúc sau

```
typedef struct {
    int masv;  // mã hiệu sv
    char malop[12];
    char ho[30];
    char ten[30];
    float diemk1;
    float diemk2;
} sinhvien
```

Bài tập 3

- Viết chương trình nhập vào một mảng các số nguyên, hãy xác định xem liệu có thể chia mảng này thành hai mảng con có tổng các phần tử của mỗi mảng là bằng nhau.
  - Ví dụ: mảng {1, 5, 11, 5} có thể chia được (hai mảng con {1, 5, 5} và {11})
  - Ví dụ: mảng {1, 5, 3} không chia được.

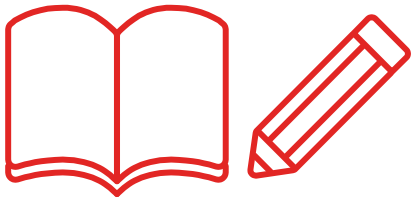
Bài tập



- Sửa chương trình trên để tính toán kết quả của 1 biểu thức hậu tố với các toán hạng tổng quát (có thể là số thực, có thể âm…)
- Xây dựng chương trình chuyển đổi 1 biểu thức từ trung tố sang hậu tố, biểu thức trung tố là 1 xâu ký tự với các toán hạng tổng quát và các phép toán cùng độ ưu tiên như sau :

() > ^ > * = % = / > + = –

Bài tập 4



- Viết một chương trình:
  - *Nhập vào từ bàn phím một số nguyên dương có N chữ số.*
  - *Nhập vào một giá trị nguyên dương M.*
  - *Hãy thực hiện xoá đi M chữ số trong số N để thu được số còn lại sau khi xoá là lớn nhất có thể, xây dựng thuật toán sử dụng Stack.*

  - *Ví dụ: số 2019*
    - Xoá 1 chữ số: 219
    - Xoá 2 chữ số: 29
    - Xoá 3 chữ số: 9

Xoá để được số lớn nhất

- Input:
  - Số nguyên dương có N chữ số
  - M: số chữ số cần xoá (0<= M < N)
- Output:
  - Số lớn nhất sau khi xoá M chữ số
- Phân tích
  - Chuyển đổi số thành xâu ký tự để thuận tiện xử lý
  - Nhận thấy khi xóa đi mà số vẫn lớn nhất thì mỗi lần xóa một chữ phải tạo ra số lớn nhất
  - Các số sau khi xoá vẫn theo thứ tự ban đầu nên để thu được số lớn nhất có thể ☾ bắt đầu xoá từ phía bên trái

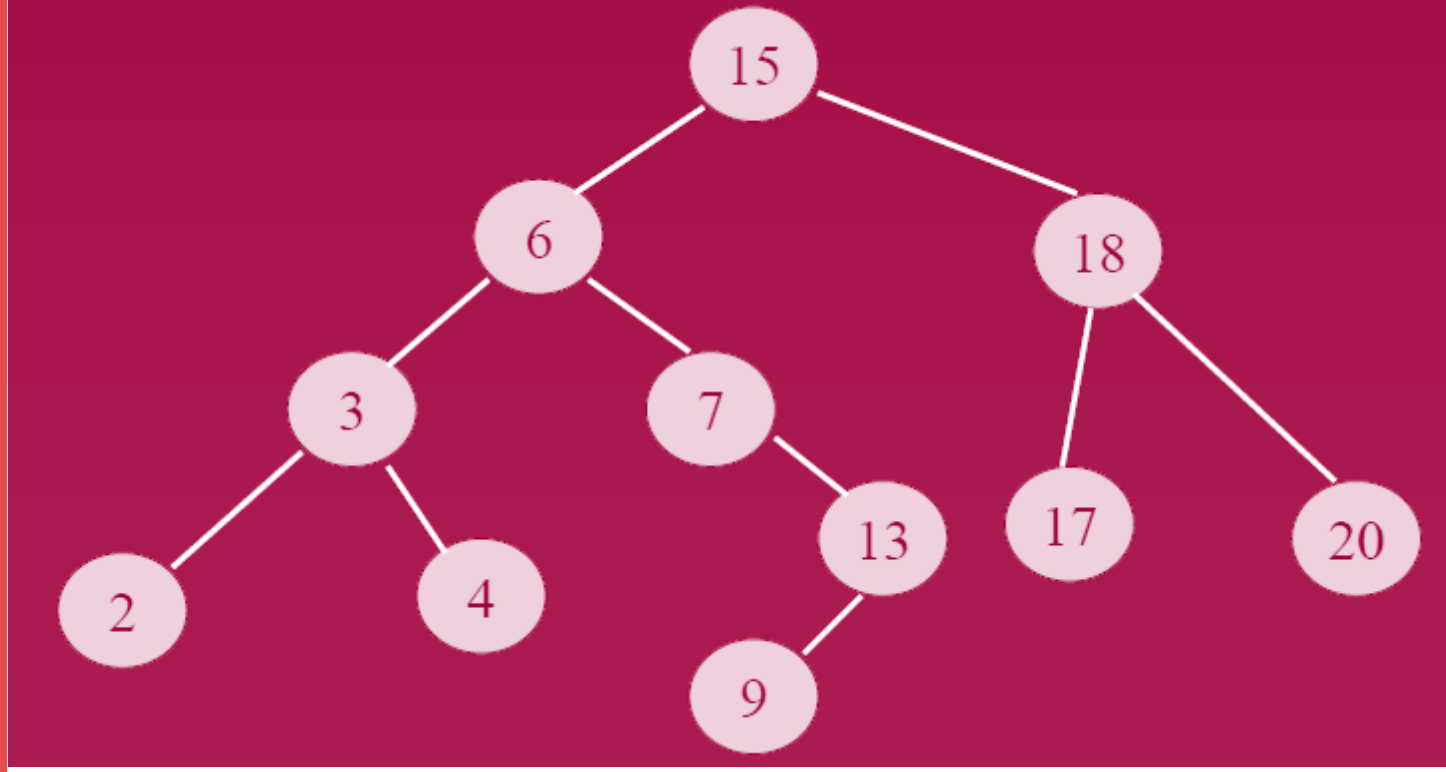
Xoá để được số lớn nhất

- Phân tích: sử dụng ngăn xếp
  - Dãy đó luôn là dãy lớn nhất có thể tạo được khi xóa
  - Stack sẽ chứa các chữ số được chọn
  - Ví dụ: số 3973811 gồm N=7 chữ số, cần xoá M=3 chữ số
    - Duyệt lần lượt các chữ số từ trái sang phải, stack ban đầu rỗng
    - Đọc chữ số đầu tiên: 3 và M=3 ☾ push vào stack
      - Stack: 3 (cần xoá M=3)
    - Đọc chữ số tiếp theo: 9 và M=3 ☾ so sánh với chữ số trong đỉnh stack: 9 > 3 ☾ pop 3 ra khỏi stack và push 9 vào thay thế, 3 là chữ số sẽ bị xoá
      - Stack: 9 (cần xoá M=2)
    - Đọc chữ số tiếp theo: 7 và M=2 ☾ so sánh với chữ số trong đỉnh stack: 7 < 9 ☾ push 7 vào trong stack
      - Stack: 9 7 (cần xoá M=2)

Xoá để được số lớn nhất

- Phân tích: sử dụng ngăn xếp
  - Ví dụ: số 3973811 gồm N=7 chữ số, cần xoá M=3 chữ số
    - Đọc chữ số tiếp theo: 3 và M=2 ☾ so sánh với chữ số trong đỉnh stack: 3 < 7 ☾ push 3 vào trong stack
      - Stack: 9 7 3 (cần xoá M=2)
    - Đọc chữ số tiếp theo: 8 và M=2 ☾ so sánh với chữ số trong đỉnh stack: 8 > 3 ☾ pop 3 ra khỏi stack (M=1), 8 > 7 ☾ pop 7 ra khỏi stack (M=0) và push 8 vào thay thế, 3 và 7 là chữ số sẽ bị xoá
      - Stack: 9 8 (cần xoá M=0)
    - Đọc chữ số tiếp theo: 1 và M=0 ☾ push 1 vào stack
      - Stack: 9 8 1 (cần xoá M=0)
    - Đọc chữ số tiếp theo: 1 và M=0 ☾ push 1 vào stack
      - Stack: 9 8 1 1 (cần xoá M=0)
    - Kết thúc ☾ số thu được 9811

Ví dụ



- Thứ tự trước: **15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20**
- Thứ tự giữa: **2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20**
- Thứ tự sau: **2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15**