C & C++ Programming

অটোমেটিক স্ক্রলের মাধ্যমে ই-বুক পড়া / রিড়ের জন্যঃ

আপনার ই-বুক বা pdf রিডারের Menu Bar এর View অপশনটি তে ক্লিক করে Auto /Automatically Scroll অপশনটি সিলেক্ট করুন (অথবা সরাসরি যেতে ⇨ Ctrl + Shift + H)। এবার ↑ up Arrow বা ↓ down Arrow তে ক্লিক করে আপনার পড়ার সুবিধা অনুসারে স্ক্রল স্পীড ঠিক করে নিন।

এই কোর্স আউটলাইন কয়েকটি ভাগে ভাগ করা হয়েছে।

1. C fundamental, Variable, Operator and Expressions.
2. Data input and output
3. Control Statement and array concept
4. Array and Function
5. Structure
6. File

C languale শিখার জন্য practice ই সবচেয়ে বড় কথা।

C fundamental

বেসিক কনসেপ্ট : ইতোমদ্ধে পূর্ববর্তী টিউন থেকে আপনারা কিছু শিখেছেন।তবুও আবার একটু ঝালিয়ে নিন।

Variable: Variable মানে চলক।আপনি যখন কোন প্রোগ্রাম করবেন, তখন বিভিন্ন ধরনের মান এর প্রয়োজন পড়বে যা variable এ জমা থাকবে। যেমন: a=5;b=3;

এখানে a,b দুইটি variable যার মান যথাক্রমে ৫ ও ৩।

Variable লিখার নিয়ম।

১-> A-Z, a-z, 0-9 এবং '_' যেকোন character এর মাধ্যমে variable declare করা যায়।

২-> তবে প্রথম character হিসেবে number ব্যবহার করা যায় না।

Valid variable names: _asdf , name , value , i , j , etc (all combination of valid character set)

Invalid variable names: 3name (প্রথম character এ Number ব্যবহার করা হয়েছে), @data (@ is not a valid character)

কোন keyword কে variable এর নাম হিসেবে লিখা যায় না। যেমনঃ case, break, int, float, double। এম্ন অল্প কিছু keyword রয়েছে।আস্তে আস্তে তা জেনে যাবেন।

Data type: Data type নিয়েও পূর্বে আলোচনা হয়েছে। চাইলে দেখে নিতে পারেন।

কমন ডাটাটাইপঃ

int -> পূর্ণসংখ্যা(integer) জমা রাখার জন্য।

float -> ভগ্নাংশ(floating) রাখার জন্য।

char -> character জমা রাখে।

ASCII Character set: এটিও আপনারা পূর্বে দেখেছেন। আমাদের কীবোর্ড এর সকল কী এর corresponding কিছু value রয়েছে। যেমন A এর ASCII value 65. অর্থাৎ আমরা যখন্ কোন
variable এ character 'A' টা জমা রাখি তখন ঐ variable এ ৬৫ জমা থাকে।

Statement: আমরা যখন কোন প্রোগ্রাম লিখব তখন বিভিন্ন ধরনের কাজ করব।যেমন এক variable এর সাথে আরেক variable যোগ করা এবং তা অন্য কোন variable এ জমা রাখা।

A=3; // A variable e 3 রাখা.

c=A+3; // c variable e A এর সাথে 3 যোগ করে c variable এ রাখা।এতে A এর value পরিবর্তিত হয় না।

উল্লেখ্য C language এ সকল statement এর পর ; দেয়া লাগে। //(double forwardslash) এর পর যা লিখা হয় তা কমেন্ট হিসেবে গন্য হয়।

Control Statement আমরা পূর্ববর্তী টিউনে relational operator ও logical operator এর কথা বলেছিলাম। এর কিছু বেসিক এখন বলা প্রয়োজন। এগুলো জানা থাকলেই control statement এর কাজ খুব সহজ হয়ে যাবে। যেকোন logical expression True হলে 1 return করে।false হলে 0 return করে। ধরুন দুইটি ভেরিএবল a , b যার value হচ্ছে ৫ ও ৭। কিছু logical expression ও তার interpretation

Expression	True/False	Value
a>b	T	1
a>=b	T	1
a<b	F	0
a<=b	F	0
a==5	T	1
a!=5	F	0

Logical operator: And(&&) ,Or(||) বোঝার জন্য আমাদের নিচের টেবিলটা বোঝা লাগবে।

	Expression A	Expression B	AND output
AND(A&&B)	True	True	True
	True	False	False
	False	True	False
	False	False	False

And operator এর ক্ষেত্রে দুইটি expression True হলেই শুধু expression টা true হয়।

	Expression A	Expression B	OR output
OR(A||B)	True	True	True
	True	False	True
	False	True	True
	False	False	False

OR operator এর ক্ষেত্রে দুইটি expression False হলেই শুধু expression টা False হয়। Expression এর ক্ষেত্রে আরও একটি কথা বলে নেয়া দরকার।একটু আগে আমরা logical expression দেখেছি। এ হয়তোবা শুধু variable ও হতে পারে। সেক্ষেত্রে ভেরিএবল এর ভ্যালুর উপর নির্ভর করে expression true না false। variable এর শুধুমাত্র শুণ্য(0) মানের জন্য expression false ।অন্য যেকোন ভ্যালুর জন্যই expression true। Suppose A=12;

Expression	True/False	Value
A	True	1
A && 5	True	1
A && A<10	False	0
A || A<10	True	1

এতটুকু বুঝতে পারলে control statement একদম সহজ্ হয়ে যাবে। control statement এর প্রধান ফিচারগুলো।

7. Conditional operator
8. If-else
9. Looping à for loop, while loop, do-while loop
10. Switch

Conditional Operator: General form of conditional operator expression1 ? expression2 : expression3 এর মূল বিষয় হচ্ছে expression1 True হলে expression2 execute হবে, false হলে expression3 execute হবে। Example:

```
#include<stdio.h>
int main()
{
    int i=56,j;

    j= i>50?10:15; //i>50 is true,so j's value
                   //will be 10
    printf("j=%d\n");

    return 0;
}
```

.OUTPUT: **j=10**

.

if-else: if-else হল programming এর মূল স্তম্ভ। এটি তার নাম হতেই বোঝা যায়।

General form:

if(expression) statement;

else statement;

if এর statement true হলে প্রথম statement execute করবে, false হলে else এর statement execute হবে।

last example by if-else:

Output: **j=10**

if তার নিচের একটি statement কভার করে। যদি এক এর অধিক statement হয়, সেক্ষেত্রে compound statement এর মত {} ব্র্যাকেট এর মধ্যে লিখা লাগবে।

**

simple statement:

i=5;

j=var;

a=a+b; etc

compound statement:

{

i=5;

j=var;

c=a-20;

}

Example of if-else using compound statement:

int i=20;

if(i>50)

{

printf("i=%d\n", i);

i=i*2;

}

else

{

printf("i=%d\n", i); i=i*3;

}

printf("i=%d\n);

Output: i=20 i=60

else ছাড়াও শুধুমাত্র if ব্যবহার করা যায়।

if(expression) statement;

else if:Multiple if condition:

general form:

if(exp) statement;

else if(exp) statement;

else if(exp) statement;

else statement;

উদাহরনটা দেখুন।

. **Nested if-else:** একটি if বা else এর statement এর মধ্যে আবার if-else লিখাই Nested if-else.

General Form:

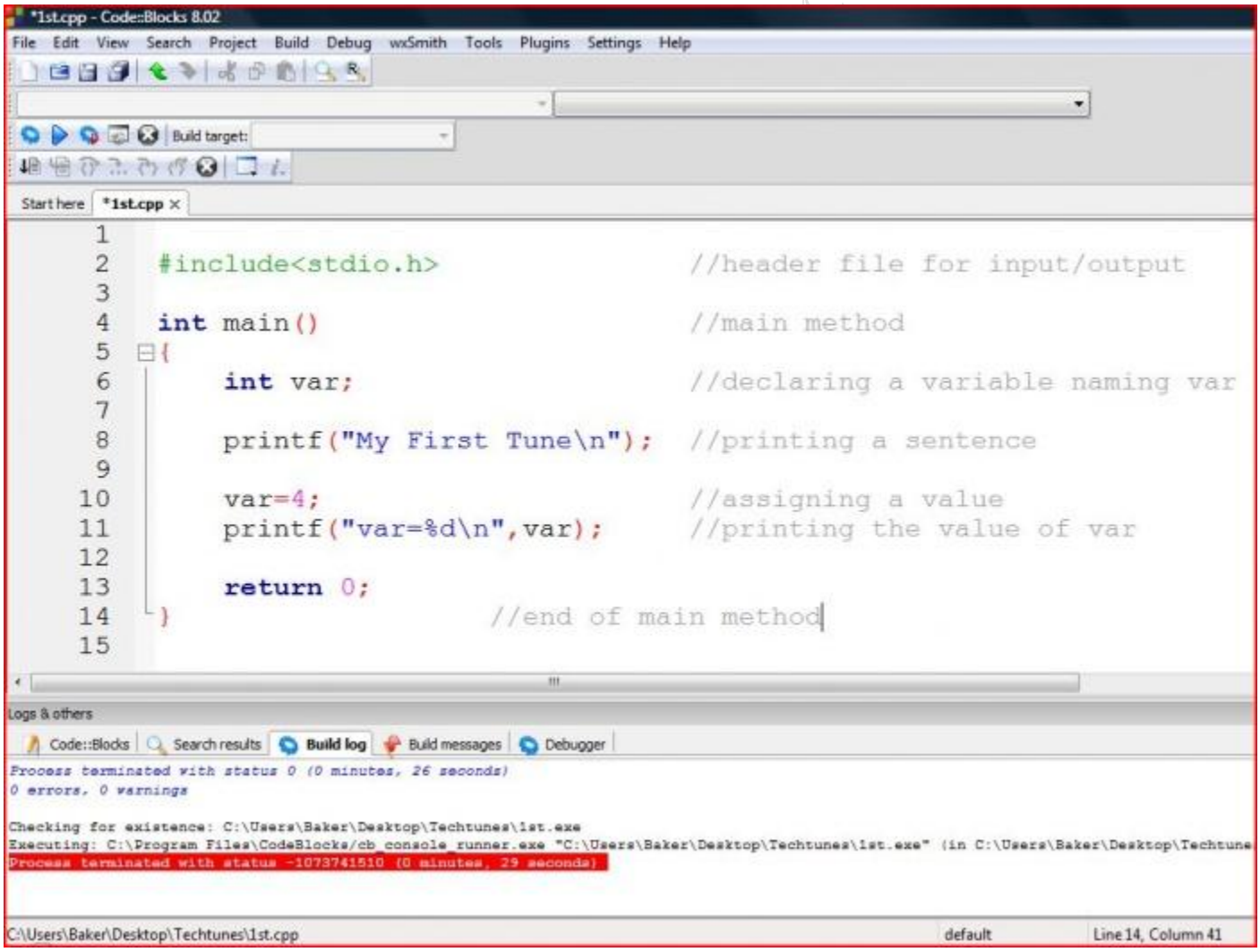
```
if(exp) {
if(exp) statement;
else statement;
}
else {
if(exp) statement;
else statement;
}
```

উদাহরনটি দেখুন।

Tasks:

1. একটি পূর্ণসংখ্যা ইনপুট নিয়ে তা জোড় না বিজোড় প্রিন্ট করে দেখান।

তিনটি সংখ্যা ইনপুট নিয়ে সবচেয়ে বড় সংখ্যাটি প্রিন্ট করুন।

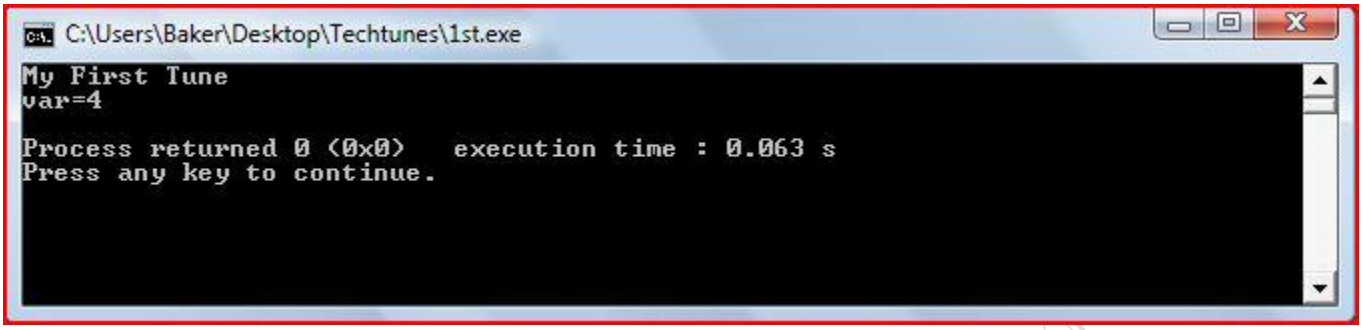


এখানে প্রত্যেকটি statement এর পর কমেন্ট করা আছে। এখানে ব্যবহৃত int main() এর int এর কাজ কি এবং return 0 statement এর কাজ function আলোচনা করার সময় বুঝতে পারবেন।

প্রোগ্রামটি compile করুন। আমি CodeBlocks IDE ব্যবহার করি। compile এর জন্য ctrl+f9 চাপতে হয়। প্রোগ্রাম এ কোন ভুল থাকলে তা নিচে দেখাবে।

রান করতে শুধু f9 চাপুন।

Output:



.আপনারা ইতোমধ্যেই %d এর কাজ বুঝতে পারছেন। এটি variable ,var এর মান দিয়ে পরিবর্তিত হয়েছে।

এমন আরও অনেক specifier আছে। যেমনঃ

%c-> shows a single character.

%f-> shows float value upto six decimal digit

%s-> shows a string.

String সম্পর্কে ধারনা পাবেন array নিয়ে আলোচনা করার সময়।

এবার ঝটপট করে যা শিখেছেন তা দিয়ে নিচের প্রোগ্রামগুলো করে ফেলুন।

2. আউটপুট স্ক্রীন এ আপনার নাম প্রিন্ট করুন।
3. একটি variable এ 5 and আরেকটি variable এ 10 রেখে যোগ, বিয়োগ, গুন, ভাগ করুন এবং প্রতিবার ফলাফল আউটপুট স্ক্রীন এ প্রিন্ট করুন।

Operator: +,-,*,/,% এগুলো সাধারন কিছু mathematical operator. C language এও এদের ব্যবহার পূর্বের মত।যেমন *,/ এর কাজ +,- এর আগে হবে। এখানেও ব্র্যাকেট এর ব্যবহার রয়েছে। তবে expression এ {},[] ব্যবহার হয় না। শুধু () ব্যবহৃত হয়। কিছু arithmetic expression

A+B+C

(A+B)*C

(A+B)*(C+D)

(A-(B*(C+D))) ...etc.

% à modulus operator.

C=A%B; এই expression এ A কে B দ্বারা ভাগ করলে যে ভাগশেষ থাকবে তা C তে জমা রাখা।

C=570%10;

C=A%5;

Modulus Operator এর একটি নিয়ম হলো operator এর আগে বা পড়ে যে variable বা ভ ব্যবহৃত হয় তা অবশ্যই integer type (int) হতে হবে।

C=A%5.1; an error

float A=5.1;

int C;

C=A%2; an error কারন A variable টা float type.

Assignment Operator: Assignment operator হচ্ছে '=' সাইন।

general form of assignment operator:

identifier (variable) = expression

Variable কে identifier ও বলা হয়।

A=33;

A=B;

A=33*B-C;

এই সবগুলো statement এ কোন একটি value, A variable এ জমা হচ্ছে। value নির্ভর করে expression এর value'র উপর।

Type Casting: বিভিন্ন গানিতিক কাজ করার পূর্বে আমাদের type casting সম্পর্কে জানা দরকার। ধরুন একটি integer type (int) variable এর মধ্যে একটি float type variable এর value assign করতে চান।

তখন int variable এর মধ্যে float variable এর মানের শুধুমাত্র integer part টা জমা হয়।

Consider following statements in a program.

int a;

float b=5.3;

a =b; //এই statement এর পর a এর মান হবে 5.

b=a; // এই statement এর পর a এর মান হবে 5.00.

এগুলো হলো auto type casting. তবে কখনো কখনো manually typecast করা লাগতে পারে।

int a,c,d;

float b=5.3;

c=503;

//a=c%b; এই লাইন টা ভুল কারন b float type variable.

//to do that task

d=(int) b; //manual type casting.

a=c%d;

এখানে d variable এ মান রাখার সময় manual type casting করা হয়েছে। এই কাজটি আরও অনেকভাবে করা যায়।যেমনঃ

int a,c=505;

float b=5.3;

a=c%(int)b; //doing the same thing.

Relational and Logical operator: mathematical operator ছাড়াও C language এ relational and logical operator রয়েছে। যা আমাদের control statement ও আরও বিভিন্ন কাজে পরবর্তীতে লাগবে।

Relational operator	Meaning
<	less than
>	greater than

>=	greater or equal to
<=	less or equal to
==	equal to
!=	Not equal to

Logical operator	Meaning
&&	And
||	Or

কোনকিছু তুলনা করার জন্য এই operator গুলো ব্যবহৃত হয়। ব্যবহার আপনারা control statement এ পাবেন।

Unary Operator: mathematical operator গুলোকে binary operator ও বলা হয়।Unary operator গুলো একটিমাত্র variable এর উপ্র কাজ করে। এগুলো হল ++, – - .

Variable এর আগে বা পরে ব্যবহার করা হয়।

int varr=5;

varr++; অথবা ++varr; // এই statement এর meaning: varr=varr+1; varr এর মান এক বাড়ানো।

varr- -; অথবা - -varr; // এই statement এর meaning: varr=varr-1; varr এর মান এক কমানো।

Conditional Operator: এটাও আমরা control statement এর সময় দেখব।

Data Input And Output

Input নেয়ার কিছু ফাংশনঃ scanf(),gets(),getchar(),getch();

scanf(): এই ফাংশনটা দিয়ে int, float, double, single character, string input হিসেবে নেয়া যায়।

example টা দেখুন।

```c
#include<stdio.h>
int main()
{
    int i;
    float f;
    char ch;                                //declaration of variables

    printf("input a number:");
    scanf("%d",&i);                         //input of an integer,%d is specifier
    printf("your entered number is %d\n\n",i);

    printf("input a floating number:");
    scanf("%f",&f);                         //input of a float
    printf("your entered floating number is %f\n\n",f);

    getchar();                              //explanation importent***

    printf("input a character:");
    scanf("%c",&ch);                        //input a character
    printf("your entered character is %c\n\n",ch);

    return 0;
}
```

.

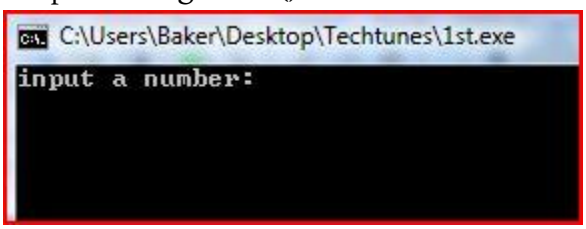
এখানে উল্লেখযোগ্য ব্যাপার হচ্ছে scanf()function এর মধ্যে & সাইন ব্যবহার। যখন কোন variable declare করা হয় তখন র‍্যাম এ ঐ variable এর জন্য জায়গা তৈরি হয়। input নেয়ার সময় C তে ঐ memory location উল্লেখ করা লাগে। তাই &সাইন দেয়া হয়েছে। যদি & সাইন ব্যবহার না করেন তবে প্রোগ্রাম compile করলেও কোন ভুল ধরবে না। কিন্তু প্রোগ্রাম রান

করে ইনপুট দেয়ার সময় প্রোগ্রাম্ ক্রাশ করবে। একে runtime error ও বলা হয়। আউটপুট এ শুধুমাত্র ভেরিএবল এর নাম দিলেই হয়।সেখানে & সাইন ব্যবহার করা লাগে না।

scanf() function এর আলাদাকারী(separator) কী হচ্ছে Enter ও space। scanf() এ int,float,double ইনপুট দেয়ার সময় নাম্বার ইনপুট না দেয়া পর্যন্ত সে wait করে। কিন্তু charater(char)type data ইনপুট নেয়ার সময় space বা enter এর ASCII value নিয়ে নেয়।

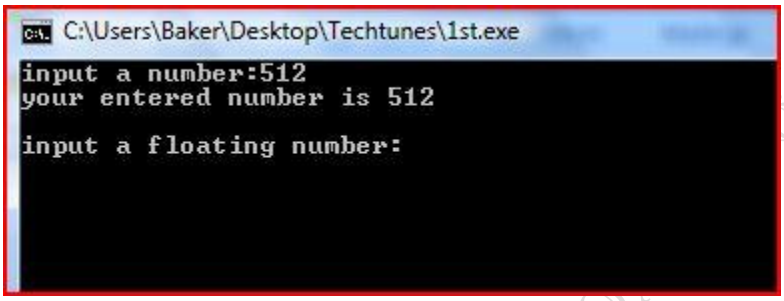
getchar() function কোন একটি character (single) ইনপুট নেয়ার জন্য ব্যবহার হয়। তাই তৃতীয় ব্লক এ single character ইনপুট নেয়ার আগে getchar() ব্যবহার করা হয়েছে float এবং character এর মধ্যবর্তী separator টা নেয়ার জন্য। getchar() টা ব্যবহার না করলে আউটপুট এর character এ কিছু দেখাবে না।

Output with getchar() function:



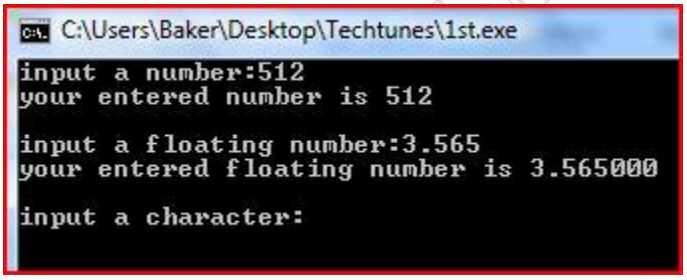
.

. Typing 512 then enter



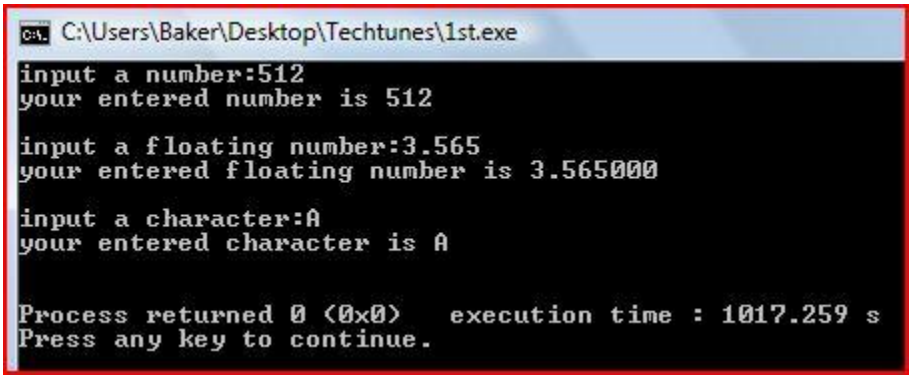
.

. Type 3.565 then enter



.

.3.565 টাইপ করার পরবর্তী এন্টার টা getchar() function এ ঢুকেছে। তারপর একটি character type করে এন্টার দিন।



. প্রোগ্রাম এর শেষ লাইন return 0; execute করে প্রোগ্রাম শেষ হল।

printf() ফাংশন এ একাধিক ভেরিএবল এর আউটপুট একসাথে দেখার জন্য প্রতিটি specifier এর জন্য ভেরিএবল এর নাম ক্রমানূসারে লিখলেই হয়।

printf("var1=%d, var2=%f ,var3=%c\n" , var1,var2,var3);

char type variable এর specifier হল %c. যদি %c এর জায়গায় %d ব্যবহার করা হয় তবে ঐ variable এর ASCII value দেখাবে।

char ch='A';

printf("ch=%d",ch);

ইনপুট নেয়ার আরো ফাংশন পরবর্তীতে দেখব।

এখন পর্যন্ত যা দেখলেন তা দিয়ে নিচের প্রোগ্রামগুলো করে ফেলুন।

1. দুইটি পূর্ণসংখ্যা ইনপুট নিয়ে আউটপুট এ দেখান।
2. একটি float type সংখ্যা ইনপুট নিন।তারপর তা একটি integer type variable এ রাখুন এবং দুইটির মান আউটপুট এ দেখুন।
3. যেকোন character ইনপুট দিয়ে সেই character ও তার ASCII value আউটপুট স্ক্রীন এ দেখান।
4. বিভিন্ন operator ব্যবহার করে প্রোগ্রাম লিখুন।

আগামী দিন control statement নিয়ে আলোচনা করব। প্রোগ্রামীং এর মূল মজাই পাবেন

Control Statement: Looping

আগের টিউনগুলোতে আমরা বিভিন্ন logical expression ও if-else এর ব্যবহার দেখেছিলাম।আজ আমরা দেখব Looping। তার আগে আপনার কাছে looping এর প্রয়োজনীয়তা তুলে ধরছি।

ধরুন আপনার দশটা সংখ্যা ইনপুট নিয়ে যোগফল দেখা দরকার। আপনি কি দশবার scanf() function দিয়ে ডাটা ইনপুট নিবেন। কখনোই না। কারন সংখ্যাটা দশ না হয়ে আরও বড় হতে পারে। এ ধরনের সমস্যার সমাধানই হচ্ছে লুপিং।

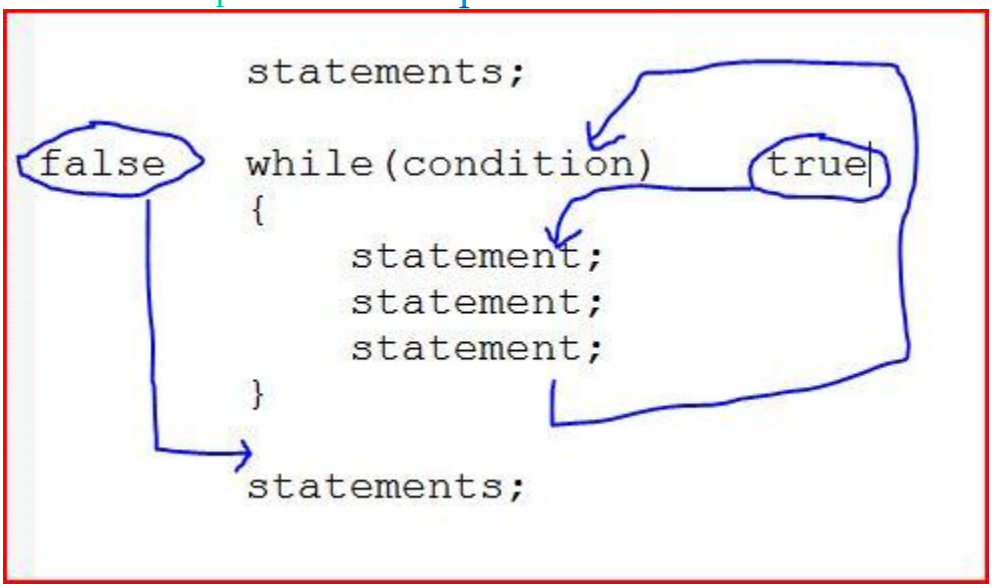
সি তে তিন ধরনের লুপিং রয়েছে।

11. while Loop
12. do-while Loop
13. for loop

while loop: General Form:

```
while(condition)    //condition means Logical Expression.
{
statement;
statement;
....
}
```

এখানে condition এর ভ্যালু যতক্ষন True ততক্ষন ও তার নিচের সবগুলো statement execute করে আবার while loop এর condition statement এ চলে যাবে। যখন condition এর ভ্যালু false হবে তখন তা while loop এর সম্পূর্ণ অংশ বাদ দিয়ে নিচে চলে যাবে। নিচের ছবিটা দেখুন।



.

এবার প্রথমে উল্লেখ করা প্রবলেমটা দেখি।

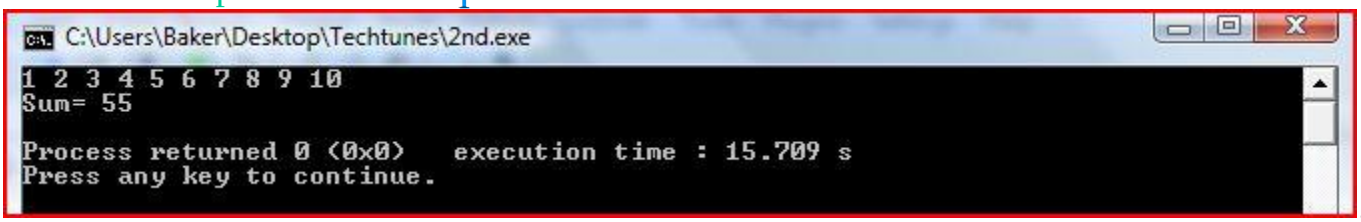
আমার দশবার loop চালানো প্রয়োজন। এবার নিচের উদাহরনটা দেখুন।

```
#include<stdio.h>

int main()
{
    int sum,i,val;          //sum এ আমাদের দশটি ইনপুট এর যোগফল থাকবে।
                            // i হল counter যা দিয়ে দশবার লুপটি control হবে।
                            // val এ আমরা ইনপুট নিব।

    i=0;
    sum=0;                  // যোগফল তো প্রথম এ শুন্যই থাকবে।

    while(i<10)
    {
        scanf("%d",&val);
        sum = sum+val;      // পূর্ববর্তী sum এর সাথে val যোগ করে যোগফল sum এ রাখছে।
        i=i+1;              // counter এর মান এক বাড়ানো হল।
    }

    printf("Sum= %d\n",sum);      // loop টি দশবার ঘুরে এই লাইন এ আসবে।

    return 0;
}
```

.

এখানে দেখুন, condition এর স্থানে i<10 রয়েছে। i এর ভ্যালু প্রতিবার loop ঘুরার পর এক করে বাড়ছে। অর্থাৎ, 0,1,2,3,4,5,6,7,8,9 এর জন্য মোট দশবার লুপ ঘুরবে। আর প্রতিবারই একটি ভ্যালু ইনপুট নিয়ে তা স এর সাথে যোগ করবে। লুপ শেষে সেই যোগফল আউটপুট এ দেখাবে।

OUTPUT:



.

উল্লেখ্য, প্রতিটি নাম্বার এর আলাদাকারী হিসেবে স্পেস কাজ করেছে। শেষে 10 লিখার পর এন্টার চাপুন আর আউটপুট দেখুন। লুপিং এর ক্ষেত্রে condition টা অতি গুরুত্তপূর্ণ। কেননা condition যদি সঠিকভাবে কাজ না করে, মানে condition যদি কখনো false না হয় তবে তা infinity loop এ পড়বে।

যেমনঃ

```
int i=1;
while(i==1)
{
printf("a");
}
printf("program ended successfully\n");
```

উপরের উদাহরনটি করে দেখুন। এই প্রোগ্রামটি কখনই শেষ হবে না। যা প্রোগ্রামটিকে ক্রাশ করাবে।

While loop নিয়ে আরও গভীরে যাওয়ার আগে আসুন array সম্পর্কে কিছুটা ধারনা নিয়ে নেই।

Array: আমাদের আজকের প্রথম প্রবলেমটা দেখুন। সেখানে আমরা দশবার ইনপুট নিয়ে তা sum এর সাথে যোগ করেছি। কিন্তু সেখানে যে নাম্বারগুলো ইনপুট নিয়েছিলাম তা যদি পরবর্তীতে প্রয়োজন হয় তা কোথাও জমা নেই। আর যদি জমা করতেও চাই তা দশটি আলাদা variable declare করে জমা রাখাও সঠিক সমাধান নয়।

একই data type এর অনেকগুলো value একই নাম দিয়ে জমা করার জন্যই array. আমরা যখন কোন variable declare করি তখন মেমরীতে ঐ variable এর জন্য জায়গা তৈরী হয়। আর array তে আমি যে সাইজ দিয়ে declare করব ঠিক ততটি জায়গা তৈরী হবে।

Array declare করার নিয়মঃ

Data-Type array-name[size]

ex: int arr[5];

নিচের ছবিটা দেখুন।

int a;

Memory:

a	2 byte

int arr[5];

Memory:

arr	2 byte	2 byte	2 byte	2 byte	2 byte
index	0	1	2	3	4

.
.
.
.
.
.
.
.

যখন একটি ভেরিএবল declare করা হয় তখন মেমরীতে একটি জায়গা তৈরী হয়েছে। আর যখন ৫ সাইজ এর array declare করা হয়েছে তখন ৫টা একই ধরনের জায়গা তৈরী হলো। Array এর ক্ষেত্রে index টা অতি গুরুত্তপূর্ণ। index নির্দেশ করে আপনি array এর কোন element টা access করবেন।

সি ল্যাঙ্গুয়েজ এর ক্ষেত্রে index শুরু হয় শুন্য(zero) থেকে। তাই index number শুন্য থেকে size-1 পর্যন্ত। size-1 এর বড় বা zero এর ছোট কোন index ব্যবহার করা যাবে না। করা হলে তাকে invalid indexing বলা হয় যা একটি runtime error তৈরী করবে।

Accessing array element: কোন element access করার জন্য লিখতে হয়,

Array_name[index_number]

Example:

```
int nums[10];
nums[0]=5;  // 0th ইনডেক্স এ ৫ রাখা।
nums[9]=25;  // 9th এ ২৫ রাখা।
num[10]=125;  //error-> invalid indexing
```

এবার প্রথম্ প্রবলেমটা আবার করি array ব্যবহার করে।

উদাহরনটা দেখুন।

```c
#include<stdio.h>

int main()
{
    int sum,i,arr[10],ind;        //sum এ আমাদের দশটি ইনপুট এর যোগফল থাকবে।
    // i হল counter যা দিয়ে দশবার লুপটি control হবে। arr তে নাম্বারগুলো আমরা ইনপুট নিব যা জমা থাকবে।
    i=0;
    sum=0;                  // যোগফল তো প্রথম এ শুন্যই থাকবে।

    while(i<10)
    {
        scanf("%d",&arr[i]);
        sum = sum+arr[i];   // পূর্ববর্তী sum এর সাথে val যোগ করে যোগফল sum এ রাখছে।
        i=i+1;              // counter এর মান এক বাড়ানো হল।
    }

    printf("Sum= %d\n",sum);     // loop টি দশবার ঘুরে এই লাইন এ আসবে।
    ind=5;
    printf("arr[%d]=%d\n",ind,arr[ind]);//5th element print করা।

    return 0;
}
```

.

সর্বশেষ printf() এ array এর একটি element access করেছি। ind তে অন্য ভ্যালু দিয়ে অন্যান্য element ও access করতে পারি।

Character array: character array এর আরেক নাম string।string এর মধ্যে একাধিক character একসাথে থাকে। “hello world”, ”what is your name?” এ ধরনের sentence রাখতে string type ব্যবহৃত হয়। আর int,float etc. type array এর সাথে তেমন পার্থক্য নেই। character array এর প্রত্যেক element ও সাধারন array’র মত access করা যায়। সাধারন array তে একটি করে element ইনপুট দেয়া লাগে। কিন্তু charater array তে gets() or scanf() function ব্যবহার করে একসাথে ইনপুট নেয়া যায়।

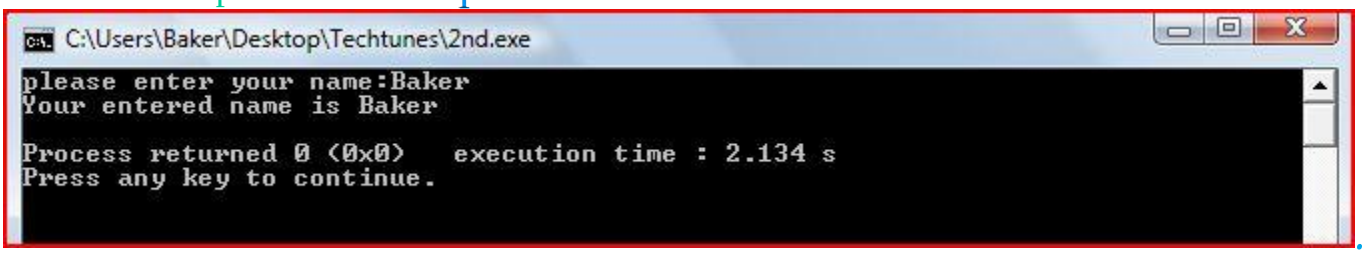
নিচের উদাহরনগুলো দেখুন।

```c
#include<stdio.h>

int main()
{
    char name[100];

    printf("please enter your name:");
    scanf("%s",name);          //%s specifier হলো string input নেয়ার জন্য।
    printf("Your entered name is %s\n",name);

    return 0;
}
```

.

.

OUTPUT:



.

.

ইনপুটটি array তে কিভাবে জমা হচ্ছে তা নিচের টেবিলে দেখুন।

Name	'B'	'a'	'k'	'e'	'r'	''			
Index	0	1	2	3	4	5	6	7	8

Character array এর বিশেষ উল্লেখযোগ্য ব্যাপার হল সবগুলো ক্যারেক্টার এর শেষে সে '' বা Null automatically বসিয়ে নেয়।int বা float type array তে যতটি element, Size ঠিক তত হলেই হয়।কিন্তু character array তে সাইজ element এর চেয়ে এক বেশী হওয়া লাগে।

Character array এর ক্ষেত্রে ইনপুট নেয়া নিয়ে কিছু কথাঃ

ধরুন আগের example টায় ইনপুট এ baker টাইপ না করে টাইপ করলেন baker md. anas. Output হিসেবে কিন্তু নাম এর অংশে baker ই দেখাবে। কারন, scanf() function er separator হিসেবে স্পেস ব্যবহৃত হয়।তাই নাম এর প্রথম অংশটি শুধু name array তে উঠবে। এন্টার কী এর আগ পর্যন্ত যদি ইনপুট হিসেবে array তে নিতে চান তবে scanf() function টা এভাবে লিখুন।

scanf("%[^\n]",name);

তৃতীয় ব্র্যাকেট এর শুরুর পর '^'(shift+6) sign এর পর যেসকল character উল্লেখ করবেন তাই separator হিসেবে কাজ করবে।ex:

scanf(" %[^\n-A]",name); // এখানে sepeartor হল enter, A, -.

Character array ইনপুট নেয়ার জন্য আরেকটি function রয়েছে। তা হল gets.

General form: gets(char_array_name);

gets() function এর separator হল এন্টার।

replace the code: scanf("%[^\n]",name); by gets(name);

তারপর রান করে আউটপুট দেখুন।

array initialization:

উপরে আমি array তে ইনপুট নিয়েছি।array element access ও করেছি। আমরা চাইলে array declare করার সময়ও মান দিয়ে দিতে পারি।

```
int num[5]={1,2,3,4,5};
float f_num[100]={5.2,3.5};
char name[]="Hello world";
```

.

.

.

.

প্রথম লাইন এর ক্ষেত্রে সাইজ ৫ এবং আমরা ৫টি element দিয়েছি।ডাটার ইনডেক্স ক্রমানুসারে থাকবে।

num[0]=1,num[1]=2,etc.

Float type এর declaration এর সময় সাইজ দিয়েছি ১০০, কিন্তু initialize করেছি মাত্র ২টা।এক্ষেত্রে প্রথম ২টা element ছাড়া বাকীগুলো undefined থাকবে।

f_num[0]=5.2, f_num[1]=3.5.

Char array তে আমরা সাইজ দেই নাই। কোন array declaration এ সাইজ declare না করলে তার সাইজ initialization এর জন্য যতটুকু দরকার ঠিক তত হয়। আর array declaration এর সময় initialize করলে অবশ্যই সাইজ দিতে হবে।

এতক্ষন যা শিখলেন তা দিয়ে নিচের প্রোগ্রামগুলো করে ফেলুন।যদি কোন সমস্যা হয় তবে জানান।

4. প্রোগ্রাম এ ইউজার থেকে যেকোন ধনাত্বক পূর্নসংখ্যা নিয়ে ১ থেকে ঐ নাম্বার পর্যন্ত যোগফল আউটপুট এ দেখান।
5. প্রোগ্রাম এ ইউজার থেকে ১০ টি নাম্বার ইনপুট নিয়ে প্রতিটি নাম্বার ১০০০ থেকে বিয়োগ করে বিয়োগফল প্রতিবার আউটপুট দেখান।
6. ইনপুট এ আপনার নাম নিন। তারপর আপনার নাম এর প্রতিটি ক্যারেক্টার এর ASCII ভ্যালু প্রিন্ট করুন।

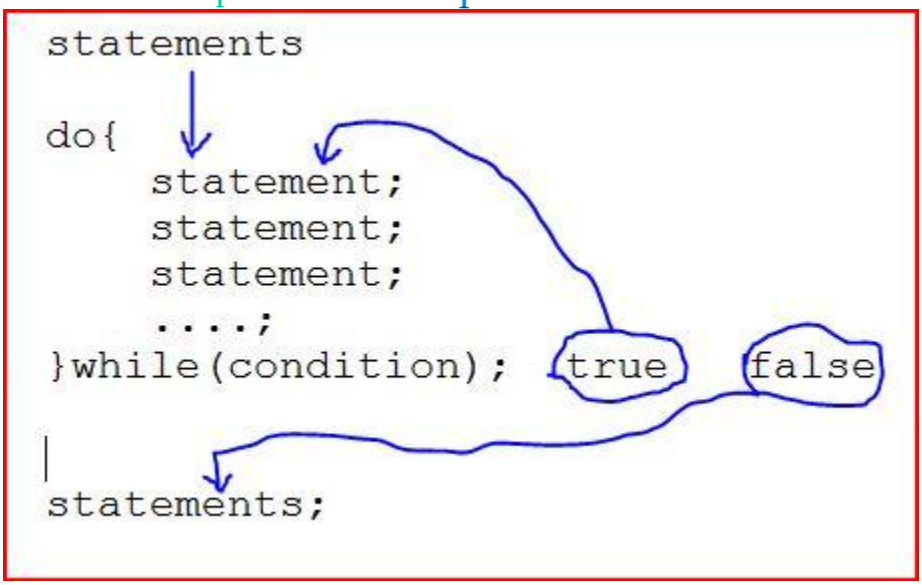
Control Statement: Looping (continued):

আগের দিন আমরা while loop ও তার ব্যবহার, সাথে array এর ধারনা পেয়েছিলাম।আজ শুরুতে do-while loop, for loop এবং লুপ এর মধ্যবর্তী পার্থক্য আলোচনা করব।

do-while loop: General form

```
do{
statement;
statement;
statement;
.....;
}while(condition);
```

চিত্রটি দেখুন।



নিচের উদাহরনটি দেখুন।

গত টিউন এর প্রবলেমটি আজ আবার করলাম do-while ব্যবহার করে।আউটপুট আগের প্রোগ্রাম এর মত হবে।

প্রোগ্রাম এ দেখুন্ i=sum=0;

একাধিক variable এ একই মান জমা করার জন্য এভাবে লিখা হয়।যেমনঃ i=j=k=sum=0; এই লাইন এ সবগুলো variable এর মান শুন্য দিয়ে replace হবে।

sum += arr[i]; এর মানে হল sum = sum + arr[i];

এ ধরনের আরও কিছু short form রয়েছে।

A += B; –> A = A + B;

A -= B; –> A = A – B;

A *= B-C; –> A = A * (B-C);

etc;

আর i++ হল i=i+1; ++ হল increment operator যা আমরা unary operator বলার সময় পড়েছিলাম।

While এবং do-while এর একটি গুরুত্বপূর্ন পার্থক্য হল, যেখানে while একবার ও execute করে না, সেখানেও do-while একবার execute করে।

লিখে দেখুনঃ

```
while(0)  //zero
{
printf("This line will not printed\n");
}
```

এবং

```
do{
printf("this line will printed\n");
}while(0);
```

কারন হল do-while এ condition check হয় লুপ এর শেষে।

for loop: General form:

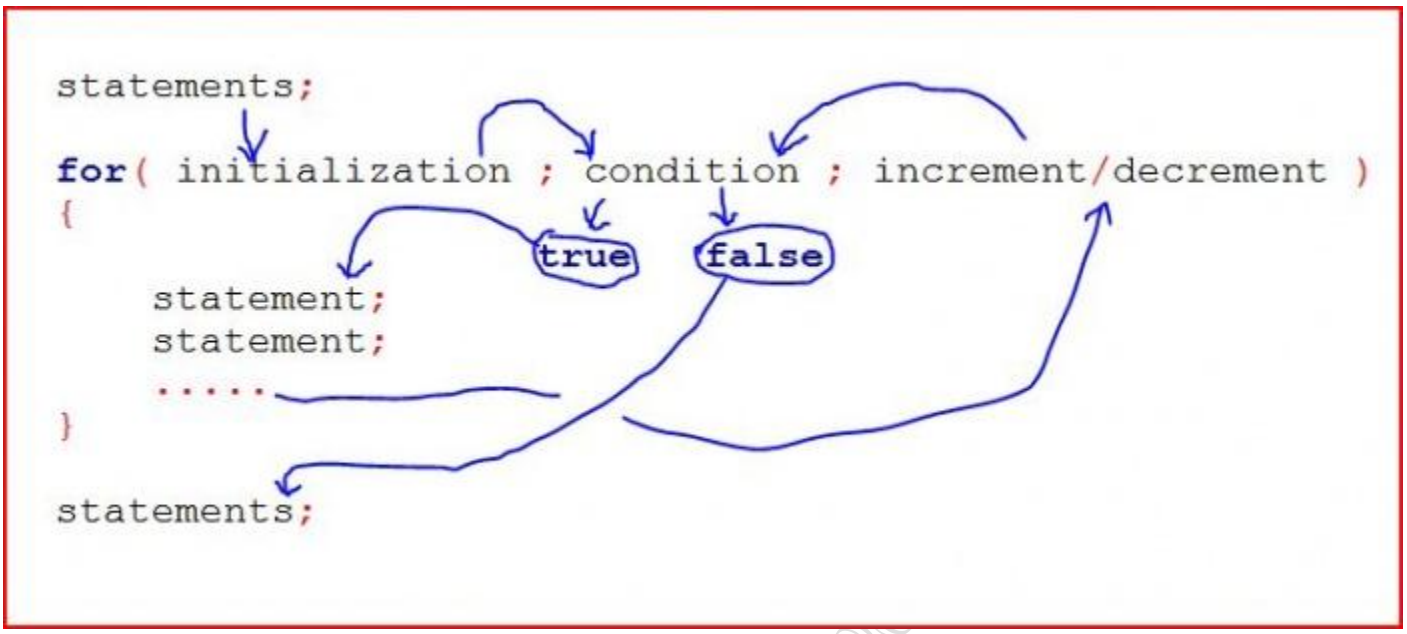
for(initialization ; condition ; increment/decrement)

{

Statements;

}

নিচের ছবিটা দেখুন।



আগের প্রোগ্রামটি আবার for loop use করে করলাম।

```c
#include<stdio.h>

int main()
{
    int i,sum,arr[10];

    for(i=sum=0; i<10 ;i++)
    {
        scanf("%d",&arr[i]);
        sum+=arr[i];
    }

    printf("Sum=%d\n",sum);

    return 0;
}
```

Nested loop: nested if-else এর মতই একটি loop এর ভেতরে আরেকটি loop ই হচ্ছে nested loop.

k=0;

for(i=0;i<10;i++)

for(j=0;j<10;j++)

{

k++;

}

printf("k=%d\n",k);

Output আসবে k=100.

Library Function: Library function হল সি ল্যাঙ্গুয়েজ এ দেয়া default function's. Library function । সব্ল ফাংশন্ এর corresponding header file আছে। যেমন আমরা যখন printf(), scanf() functionব্যবহার করেছি, তখন আমাদের "stdio.h" header file declare করতে হয়েছে। যেকোন default function প্রোগ্রাম এ ব্যবহার করা হলে তার header file অবশ্যই প্রথমে declare করা লাগবে।

কিছু গুরুত্বপূর্ন লাইব্রেরী ফাংশন।

Function name	Header file	Work
scanf(...)	stdio.h	Input নেয়া
printf(...)	stdio.h	Output দেয়া
abs(i)	stdlib.h	Return the absolute value of i
acos(d)	math.h	Return $\cos^{-1}(d)$ (মান radian এ return করে, not degree)
asin(d)	math.h	Return $\sin^{-1}(d)$
atan(d)	math.h	Return $\tan^{-1}(d)$
Sin(i)	math.h	Return sin(d) (d এর মান radian এ, degree এর জন্য function টা use করলে degree কে radian এ convert করতে হবে)
cos(i)	math.h	Return cos(d)
tan(i)	math.h	Return tan(d)
floor(d)	math.h	Return only the integer part of a fractional number.
ceil(d)	math.h	Return the integer_part+1 of a fractional number
getchar()	stdio.h	Take a single character input
isalnum(c)	ctype.h	Return true if character c is either A-Z,a-z,0-9
isalpha(c)	ctype.h	Return true if character c is either A-Z,a-z
isdigit(c)	ctype.h	Return true if character c is either 0-9
isupper(c)	ctype.h	Return true if character c is either A-Z
islower(c)	ctype.h	Return true if character c is either a-z
log(c)	math.h	Return ln(c)
log10(c)	math.h	Return log(c)
pow(base,p)	math.h	Return $base^p$
sqrt(d)	math.h	Return d (square root of d)
toupper(c)	ctype.h or	Return capital letter of a letter.

	stdlib.h	
tolower(c)	ctype.h or stdlib.h	Return small letter of a letter.
strlen(name)	string.h	Return the length of a character array
strcpy(name1,name2)	string.h	Copy name2 to name1
strcmp(name1,name2)	string.h	Return true if name1 and name2 are exactly same
gets(name)	stdio.h	Take charater array or string as input

এবার কিছু প্রবলেম করার চেষ্টা করুন। এগুলো আমি কাল example এ করে দেখাব।

1. একটি integer ইনপুট নিয়ে তা মৌলিক সংখ্যা(prime) কিনা পরিক্ষা করুন।
2. একটি character array বা string ইনপুট নিয়ে তা reverse order(last character থেকে first character) এ প্রিন্ট করুন।
3. দুইটি string ইনপুট নিয়ে তা সমান হলে equal আর না হলে not equal প্রিন্ট করুন।
4. একটি string ইনপুট নিয়ে তার সকল character, capital letter এ প্রিন্ট করুন।
5. একটি integer ইনপুট নিয়ে তা বর্গসংখ্যা কিনা পরিক্ষা করুন।

আগামীকাল লুপিং এর কিছু special কীওয়ার্ড নিয়ে কথা বলব, বিভিন্ন function ব্যবহার করব, আর সাথে থাকবে অনেক অনেক example যা আপনার কাছে লুপিং কে করে তুলবে পানির মত পরিষ্কার। সকলকে ধন্যবাদ

Control Statement:(Looping continued)

কেমন আছেন সবাই। আশা করি সি এর প্র্যাকটিস ভালভাবে করে যাচ্ছেন। আজ আমরা দেখব switch statement, break statement, continue statement;

Switch statement: switch statement অনেকটা multiple if-else এর মত যা আমরা পূর্বে দেখেছিলাম। তবুও আরেকবার multiple if-else এর general form টা দেখুন।

```
if(exp1) statement;
else if(exp2) statement;
else if(exp3) statement;
else statement;
```

এবার switch statement এর general form টা দেখুন।

```
switch(expression)
{
case expression1: statement1;
statement2;
.....
case expression2: statement3;
statement4;
.....
case expression3: statement5;
```

statement6;

.....

default: statement7;

statement8;

.....

}

Switch এর মধ্যবর্তী expression এর যে ভ্যালু হবে, ঐ ভ্যালুটা case এর যে ভ্যালু এর সাথে মিলে যাবে, তখন ঐ case এর statement হতে execute করা শুরু হবে।

নিচের উদাহরন দুইটা দেখুন।

```c
#include<stdio.h>

int main()
{
    int val;

    printf("Enter a positive number between(0-2):");
    scanf("%d",&val);

    if(val==0)  printf("you entered zero\n");
    else if(val==1) printf("you entered one\n");
    else if(val==2) printf("you entered two\n");
    else printf("your entered value not in 0-2\n");

    return 0;
}
```

উপরের প্রোগ্রাম এর switch Version:

```c
#include<stdio.h>

int main()
{
    int val;

    printf("Enter a positive number between(0-2):");
    scanf("%d",&val);

    switch(val)
    {
        case 0:
            printf("you entered zero\n");
            break;
        case 1:
            printf("you entered one\n");
            break;
        case 2:
            printf("you entered two\n");
            break;
        default: printf("your entered value not in 0-2\n");

    }
    return 0;
}
```

উপরে উল্লেখিত দুইটি প্রোগ্রাম এরই আউটপুট একই হবে। আপনারা প্রোগ্রামটি লিখে রান করে আউটপুট দেখবেন। switch এর ভ্যালু case এর যে ভ্যালুর সাথে মিলে যায়, ঐ case ব্লক হতে statement execution শুরু হয়।

Switch এর প্রোগ্রামটিতে উল্লেখ্য বিষয় হল, break statement এর ব্যবহার। switch statement এ প্রতিটি case এর ব্লক এ সর্বশেষ statement টা হল break। break statement যদি প্রতিটি ব্লক এ ব্যবহৃত না হত তবে কোন একটি case এর statement শুরু করে break statement না পাওয়া পর্যন্ত বা switch statement শেষ না হওয়া পর্যন্ত সকল statement execute করবে।

ধরুন, শুধুমাত্র “case 0:” এর ব্লক এর break statement টা নেই। সেক্ষেত্রে 0 ইনপুট দিলে আউটপুট হবে

Output:

you entered zero

you entered one

Break Statement: break statement ব্যবহৃত হয় প্রধানত লুপ terminate করা বা switch থেকে বের হয়ে যাওয়ার জন্য। যেমন পূর্ববর্তী উদাহরনটিতে আমরা break statement ব্যবহার করেছি।

For loop, while loop, do-while loop এর ক্ষেত্রেও break statement ব্যবহৃত হয়। নিচে for loop এর মধ্যে break statement ব্যবহার করা হল।

```c
#include<stdio.h>

int main()
{
    int i;


    for(i=0;i<10;i++)
    {
        if(i==5)
        {
            break;
        }
        printf("i=%d\n",i);
    }
    printf("outside for loop: i= %d\n",i);

    return 0;
}
```

লুপিং এর ক্ষেত্রে যখন break statement execute হয়, তখন যে loop এর ভিতরে break statement টি লিখা হয়েছে, সেই লুপ এর মধ্যে আর কোন কাজ না করে সরাসরি বাইরে চলে আসে।

Continue statement: continue statement ব্যবহৃত হয় বিভিন্ন লুপিং প্রোগ্রাম এর ক্ষেত্রে। continue statement যদি কোন while or do-while loop এর ক্ষেত্রে execute হয় তবে ঐ continue statement এর পর সরাসরি লুপিং এর condition এর অংশে চলে যায়।আর যদি for loop হয় সেক্ষেত্রে for loop এর increment/decrement অংশে চলে যায়। নিচের উদাহরনগুলো ব্যাপারটাকে পরিস্কার করে তুলবে।

```c
i=0;
while(i<10)
{
    i++;
    if(i==5||i==6)
        continue;
    printf("%d ",i);
}
printf("\noutside loop\n");

output: 1 2 3 4 7 8 9 10
        outside loop
```

```
i=0;
do
{
    i++;
    if(i==5||i==6)
        continue;
    printf("%d ",i);
}while(i<10);

printf("\noutside loop\n");

output: 1 2 3 4 7 8 9 10
        outside loop
```

```
for(i=1;i<=10;i++)
{
    if(i==5||i==6)
        continue;

    printf("%d ",i);
}
printf("\noutside loop\n");

output: 1 2 3 4 7 8 9 10
        outside loop
```

Comma Operator: comma operator সাধারনত একের অধিক expression একইসাথে উল্লেখ করার জন্য ব্যবহৃত হয়। যেমনঃ for loop এর initialization বা increment অংশে একটি expression লিখার সুযোগ রয়েছে। কিন্তু আমরা ‘,’ operator ব্যবহার করে একাধিক expression একইসাথে লিখতে পারি।

যেমনঃ

```
for(i=0,j=10,k=0; i<j ; i++,j--)
{
    k++;
}

printf("total time looped: %d\n",k);

output:
total time looped: 5
```

অনেকসময় এভাবেও ব্যবহার করা যেতে পারেঃ

if(i==3)

j=j+2,k++;

কিংবা

i=1 , j=0; //একই statement এ দুইটি ভ্যারিএবল initialize করা।

আজ আমরা সি এর একটি অতি গুরুত্বপূর্ন বিষয় নিয়ে আলোচনা করব। আজকে আমরা ফাংশন এর কিছু বেসিক ধারনা, প্রয়োজনীয়তা ও প্রোগ্রামের মধ্যে এর ব্যবহার দেখব।

Function (ফাংশন):

ফাংশন নিয়ে আলোচনা করার পূর্বে আমাদেরকে এর প্রয়োজনীয়তা জানা দরকার। আমার গত টিউনে আমি প্রবলেম এর সমাধান হিসেবে একটি প্রাইম নাম্বার চেক এর কোড করে দেখিয়েছিলাম।গতপর্বের সমাধান পেতে এখানে ক্লিক করুন। ঐ প্রোগ্রাম এ আমরা শুধুমাত্র একটি নাম্বার প্রাইম কি প্রাইম না তা চেক করেছিলাম। এখন ধরুন আমাদের প্রোগ্রাম এর বিভিন্ন জায়গায় প্রাইম টেস্ট করা দরকার। তখন আপনি কি করবেন? প্রতিবার ঐ একই কোড লিখবেন। এ ধরনের সমস্যার সমাধানই হচ্ছে ফাংশন। ফাংশন লিখার ফলে একই কোড বারবার লিখতে হয় না।

ফাংশন কিঃ

ফাংশন হচ্ছে একটি প্রোগ্রাম সেগমেন্ট যা কিছু সুনির্দিষ্ট কাজ করে থাকে। সকল সি প্রোগ্রাম এক বা একাধিক ফাংশন এর সমন্বয়ে গঠিত। আমরা যে main() declare করি ও তার মধ্যে বিভিন্ন কাজ করি সেই main() ও একটি ফাংশন। ফাংশন লিখার কিছু নিয়ম রয়েছে। এগুলো নিচে উল্লেখ করছি।

ফাংশন লিখা ও Call করার নিয়মঃ

ফাংশন এর general form:

return_type function_name(type1 par1,type2 par2,.....,typeN parN)

{

contents of function;

return ret_value;

}

type1 parameter1,type2 parameter2,.... ,type_n parameter n হল parameter_list.

এখানে return_type হচ্ছে ফাংশনটি যে ভ্যালু return করবে তার ডাটাটাইপ। এখানে উল্লেখ্য, return করা বলতে প্রোগ্রাম এর যে জায়গা হতে function call করা হয়েছে সে জায়গায় ভ্যালু ফেরত পাঠানো। function_name হচ্ছে ঐ ফাংশন এর নাম। ফাংশন এর নামকরনের ক্ষেত্রে identifier declare করার নিয়ম মানতে হয়। identifier নিয়ে আমরা প্রথম্ টিউনেই আলোচনা করেছিলাম।আর parameter_list হচ্ছে ঐ প্রোগ্রামে যে সকল ভ্যালু পাঠাতে হয় তার list. আমরা সি তে যেসকল ডিফল্ট ফাংশন দেখেছিলাম তেমন একটি ফাংশনের স্ট্রাকচার দেখুন।

```
char toupper(char ch)
{
    if(ch>='a' && ch<='z')
        ch=ch-'a'+'A';        //code for
                              //converting from upper case
                              //to  lower case

    return ch;
}
```

toupper() ফাংশন এর কাজ ছিল একটি ক্যারেক্টার নিয়ে তার uppercase ফেরত পাঠানো। উপরের ফাংশনটিতে function_name এর পরে ব্র্যাকেটে char ch হল parameter list. parameter যদি একাধিক হত তবে parameter গুলো কমা (",") দিয়ে লিখা হত। যেমনঃ pow(int base,int power) . পাওয়ার ফাংশন এর দুইটি প্যারামিটার। একটি হল বেস ,আরেকটি পাওয়ার।

প্রথমে উল্লেখ করা char হল return_type. toupper ফাংশনটি একটি character নিয়ে তার আপারকেস character return করে।

ফাংশন এর সর্বশেষ লাইনটি হচ্ছে return statement. ঐ স্টেটমেন্ট এর মাধ্যমে ফাংশনটি যে জায়গা হতে call করা হয়েছিল সে জায়গায় return এর ভ্যালু নিয়ে ফিরে আসে।উপরের ফাংশনটিতে যখন return ch; স্টেটমেন্টটি execute হবে তখন ch এর ভ্যালু যেখান থেকে ফাংশনটি call করা হয়েছে,সেখানে পাঠাবে এবং code এর execution ও সেখানে ফেরত আসবে। return_type এর মধ্যে int,float,double,char, [unsigned/signed specifier] সহ datatype উল্লেখ করা যায়। কিছু কিছু ফাংশন রয়েছে যারা শুধুমাত্র parameter নেয়ার পর ঐ ফাংশন এর সুনির্দিষ্ট কাজ করে কিন্তু কোন ভ্যালু return করে না। সে ধরনের ফাংশন এর return_type হিসেবে declare করা হয় void । যদি কোন ফাংশন এর পূর্বে কোন ডাটাটাইপ উল্লেখ করা না হয় তবে সেই ফাংশন এর ডাটাটাইপ void ধরা হয়। return_type হিসেবে কোন array সি তে ডিক্লেয়ার করা যায় না। main function হতে কোন ফাংশন কল করার জন্য ঐ ফাংশন এর নাম ও তার প্যারামিটার এর ভ্যালু উল্লেখ করে দিতে হয়।

```c
#include<stdio.h>

char _toupper(char ch)
{
        if(ch>='a' && ch<='z')  //code for
            ch=ch-'a'+'A';      //converting from upper case
                                //to  lower case
        return ch;
}

void main()
{
    char c;

    printf("enter a lowercase character:");
    scanf("%c",&c);

    printf("equivalant uppercase:%c\n",_toupper(c));

}
```

toupper() একটি ডিফল্ট ফাংশন। আর _toupper হচ্ছে আমাদের define করা ফাংশন।দুটো ফাংশন এর নাম আলাদা রাখা হয়েছে।

যদি কোন ডিফল্ট ফাংশন এর header file declare করা না থাকে তবে ঐ নামে অন্য কোন ফাংশন declare করা যায়। কিন্তু যদি header file declare করা থাকে তবে ঐ নামে অন্য কোন ফাংশন declare করা যায় না। আমাদের উপরের প্রোগ্রামটিতে ফাংশন এর নাম চাইলে _toupper কেটে toupper লিখলে error message দেখাবে না। কিন্তু যদি #include<ctype.h> দেয়া হয় সেক্ষেত্রে আর ফাংশন এর নাম toupper দেয়া যাবে না।

ফাংশনকে main function হতে, কিংবা অন্য ফাংশন হতে কিংবা যে ফাংশন এ থাকবে সেটা থেকেও কল করা যায়। কোন ফাংশন হতে ঐ ফাংশনকে কল করাকে recurrsion বলা হয়। পরবর্তী টিউনে recurrsion নিয়ে আলোচনা করব।

ফাংশন দুইভাবে প্রোগ্রাম এ লিখা যায়। একটি হল main() function এর পূর্বে, আরেকটি হল main() function এর নিচে। main function এর উপরে লিখার ক্ষেত্রে সাধারন নিয়মে একটি ফাংশন ডিক্লেয়ার করার পর আরেকটি ফাংশন, এভাবে সবগুলো ফাংশন ডিক্লেয়ার করা হয়।আর main() function এর নিচে কোন একটি, দুটি বা সবগুলো ফাংশন ডিক্লেয়ার করা যায়। সেক্ষেত্রে যে ফাংশনটি main() function এর পর ডিক্লেয়ার করা হবে তার প্রথম লাইন বা declaration টা main() function এর উপর লিখে সেমিকোলন(;) দিতে হয়। আর main() function এর নিচে ফাংশনটি সম্পূর্ণ করা হয়। নিচের উদাহরনটি দেখুন।

```
#include<stdio.h>

int maximum(int x,int y)
{
    return x>y?x:y;
}
int minimum(int x,int y);
char _toupper(char ch);
int main()
{
    char c;
    int a,b;
    printf("enter a character:");
    scanf("%c",&c);
    printf("enter two integer:");
    scanf("%d %d",&a,&b);
    printf("uppercase character:%c\n",_toupper(c));
    printf("max=%d min=%d\n",maximum(a,b),minimum(a,b));
}

int minimum(int x,int y)
{
    return x<y?x:y;
}
char _toupper(char ch)
{
    //previus example code here
}
```

উপরের উদাহরনটিতে একটি ফাংশন main() function এর পূর্বে, আর দুইটি ফাংশন main() function এর নিচে ডিক্লেয়ার করা হয়েছে।

এবার একটি ফাংশন লিখছি যার কাজ হচ্ছে তিনটি সংখ্যা ইনপুট নিয়ে তার মধ্য থেকে সবচেয়ে বড় সংখ্যাটি return করবে। return type যেহেতু নাম্বার তাই ডাটাটাইপ ডিক্লেয়ার করছি int.

নিচের উদাহরনটি দেখুনঃ

```c
#include<stdio.h>

int max_of_3_num(int x,int y,int z)
{
    int t;

    t=x>y?x:y;        //first finding the largest of x,y and
                      //putting into t;

    t=t>z?t:z;        //then finding the largest between t and z
                      // and putting it into t;

    return t;         //now t have the largest value,returning it.
}

void main()
{
    int a,b,c,largest;

    printf("enter three integer:");
    scanf("%d %d %d",&a,&b,&c);

    largest=max_of_3_num(a,b,c);
    printf("maximum value=%d\n",largest);

}
```

উপরে উল্লেখিত প্রোগ্রামটিতে আমরা তিনটি integer value এর মধ্যে সবচেয়ে বড়টি বের করার একটি ফাংশন লিখেছি। এখন আমার যেখানে প্রয়োজন সেখানে শুধুমাত্র ফাংশনটি কল করে কাজটি করতে পারি।

আজ এ পর্যন্তই। কাল আমরা ফাংশন এর pass by value, pass by reference, recursion নিয়ে আলোচনা করব।

নিজে নিজে করিঃ

এবার ঝটপট নিচের প্রোগ্রামগুলো করে ফেলুন। কেননা প্র্যাকটিস না করে সি পড়ে তেমন লাভ নেই।

১। এমন একটি প্রোগ্রাম লিখুন যে প্রোগ্রামটি শুন্য(0) ইনপুট দেয়ার আগে পর্যন্ত একটি করে নিবে এবং তা প্রাইম integer input হলে"%d is prime" আর না হলে "%d is not prime show" করবে।

২। এমন একটি প্রোগ্রাম লিখুন যা q ইনপুট দেয়ার আগে পর্যন্ত একটি ক্যারেক্টার ইনপুট নিবে এবং তার lowercase আউটপুট এ দেখাবে।

৩। এমন একটি প্রোগ্রাম লিখুন যে প্রোগ্রামটি শুন্য(0) ইনপুট দেয়ার আগে পর্যন্ত একটি করে নিবে এবং ঐ নাম্বার integer input এর সবগুলো ডিজিট এর যোগফল দেখাবে।

৪। এমন একটি প্রোগ্রাম লিখুন যে প্রোগ্রামটি শুন্য(0) ইনপুট দেয়ার আগে পর্যন্ত একটি করে নিবে এবং ঐ integer input ,০২১দিলে ১২০করবে।যেমনঃ show এ reverse order নাম্বারটিকে১২৫৪ দিলে ৪৫২১ ইত্যাদি।

আজকে আমরা দেখব ফাংশনের pass by value, pass by reference, recurrsion.

Pass by value:

Pass by value হচ্ছে কোন একটি ফাংশন এর মধ্যে যে সকল ভ্যালু প্যারামিটার হিসেবে পাস করা হয়েছে তার ভ্যালু যদি ঐ ফাংশন এর মধ্যে চেঞ্জ করা হয় তবে ঐ চেঞ্জ শুধুমাত্র ঐ ফাংশন এর মধ্যেই চেঞ্জ হবে। ফাংশন যখন আবার যেখান থেকে কল করা হয়েছিল সেখানে ফেরত আসবে তখন ঐ প্যারামিটার এর ভ্যালু আগের মতই থাকবে।

নিচের উদাহরনটি দেখুন।

```
#include<stdio.h>

void double_the_value(int val)
{
    printf("inside the function,value of a:%d\n",val);
    val=val*2;
    printf("a has been doubled. a=%d\n",val);
}

int main()
{
    int a;

    printf("enter an integer:");
    scanf("%d",&a);

    double_the_value(a);
    printf("returned from function and value of a:%d\n",a);

    return 0;
}
```

এখানে কিছু কথা বলে নেয়া ভাল, কোন ফাংশন এক্সেস করার সময় প্যারামিটার এর ভ্যারিএবল ও ফাংশন এর ভ্যারিএবল সংখ্যা অবশ্যই সমান হতে হবে। যেমন ধরুন কোন ফাংশন এর definition টা অনেকটা এরকম

```
int findmax(int a,int b,int c)
{
//internal code of function
}
```

এই ফাংশন এক্সেস করার জন্য কোড লিখা হলঃ

```
value= findmax(x,y,z);
```

তখন ঐ ফাংশন এর মধ্যে

–>a এর ভ্যালু হবে x এর ভ্যালু

–>b এর ভ্যালু হবে y এর ভ্যালু

–>c এর ভ্যালু হবে z এর ভ্যালু

যে সিরিয়ালে প্যারামিটার উল্লেখ করা হবে ভ্যালুও সেই সিরিয়ালে assigned হবে।

এবার আমরা উদাহরনটি নিয়ে আলোচনা করি। উদাহরনটিতে ফাংশন এর প্যারামিটার ভ্যালু হিসেবে শুধুমাত্র a এর ভ্যালু পাঠানো হয়েছে। আর ফাংশন এর মধ্যে ঐ ভ্যালুটি ডাবল হলেও সেই ডাবল ভ্যালুটি শুধুমাত্র ঐ ফাংশন এর মধ্যেই কার্যকর। তাই যখন main() function এ ফেরত এসেছে তখন a এর ভ্যালু হিসেবে ফাংশন এ যাওয়ার সময় যে ভ্যালু ছিল তাই থাকবে।

লক্ষ্য করুন, আমরা যখন ভ্যারিএবল বা অ্যারে নিয়ে আলোচনা করেছিলাম তখন আমরা বলেছিলাম যে, যদি কোন ভ্যারিএবল ডিক্লেয়ার করা হয় সেক্ষেত্রে মেমরীতে ঐ ভ্যারিএবল এর জন্য জায়গা তৈরী হয়। ফাংশন কল এর সময় আমরা যখন প্যারামিটার লিস্ট এ কোন ভ্যারিএবল এর নাম উল্লেখ করছি তখন ঐ ভ্যারিএবল এর ভ্যালুটা argument হিসেবে পাস হয়। আর ফাংশন এর ডেফিনেশন এর মধ্যে যে ভ্যারিএবল এর নাম উল্লেখ করা হয়, ঐ ভ্যারিএবল এর মেমরী লোকেশন এ যে argument receive হয় সেই ভ্যালুটাই assign হয়। তাই যখন ফাংশন এ কোন ভ্যারিএবল এর মান ডাবল করা হচ্ছে, তখনঐ ফাংশনের ভ্যারিএবল এর ভ্যালু ডাবল হচ্ছে, কিন্তু main() function এর মধ্যে উল্লেখিত ভ্যারিএবল এর মানের কোন পরিবর্তন হয় না। সেকারনে যখন main() function এ ফেরত আসে, তখন main() function এর ভ্যারিএবল এর ভ্যালুর কোন পরিবর্তন দেখা যায় না।

Pass by reference:

Pass by reference বোঝার জন্য আমাদের Pointer সম্পর্কে জানা থাকা প্রয়োজন। Pointer নিয়ে আমরা বিস্তারিত পরে দেখব। এখন শুধুমাত্র অল্প ধারনা নিন। তা না হলে pass by reference এর উদাহরনটি ঠিকভাবে বুঝতে পারবেন না।

Pointer variable হলো সে সকল ভ্যারিএবল যারা কোন ভ্যালুর পরিবর্তে "মেমরী লোকেশন" জমা রাখে।

Pointer variable declare করার নিয়মঃ

datatype *variable_name;

datatype হল pointer variable টি কোন integer এর মেমরী লোকেশন রাখবে, না double এর মেমরী লোকেশন রাখবে তা।

pointer variable নামের পূর্বে (*) asterix চিহ্ন দেয়া লাগে।

এতো গেল নাম ডিক্লেয়ার করা। প্রোগ্রাম এ pointer variable ব্যবহার সাধারন ভ্যারিএবল হতে কিছুটা আলাদা।

যেমনঃ ডিক্লেয়ার করার পর অন্য কোথাও যদি শুধুমাত্র ভ্যারিএবল এর নাম লিখা হয় তবে তা memory adderss নির্দেশ করে। আর যদি * দিয়ে ভ্যারিএবল এর নাম লিখা হয় তবে তা ঐ মেমরী লোকেশন এর ভ্যালু নির্দেশ করে।

নিচের প্রোগ্রামটি ও তার কমেন্ট এবং রান করে আউটপুট দেখলে pointer সম্পর্কে প্রাথমিক ধারনা পেয়ে যাবেন।

```
#include<stdio.h>

void main()
{
    int u=3,v;          //integer value

    int *pu;            //pointer to an integer.
    int *pv;            //pointer to an integer.

    pu=&u;              //u variable এর address, pu তে assign করা হল।
    v=*pu;              //v variable এর মধ্যে pu address এর ভ্যালু বা u এর ভ্যালু assign করা হচ্ছে।
    pv=&v;

    printf("u=%d &u=%X pu=%X *pu=%d\n",u,&u,pu,*pu); //u মানে ঐ ভ্যারিএবল এর ভ্যালু, &u মানে ঐ ভ্যারিএবল এর
                                                     //address, pu মানে address নির্দেশ করছে।
                                                     //*pu দিয়ে pu address এ যে ভ্যালু রয়েছে তা নির্দেশ করছে।
    printf("v=%d &v=%X pv=%X *pv=%d\n",v,&v,pv,*pv);

}
```

উল্লেখ্য, %X specifier হল হেক্সাডেসিমেল এ কোন ভ্যারিএবল এর মান দেখায়। মেমরীর কোন এড্রেস সাধারনত হেক্সাডেসিমেল এ দেখানো হয়। উপরের প্রোগ্রামটির আউটপুট এর ক্ষেত্রে u,*pu,v,*pv এর মান সমান (3) । &u,pu এর মান সমান,আবার v,pv&এর মান সমান,কিন্তু তা আলাদা আলাদা কম্পিউটারে আলাদা আলাদা হতে পারে।মেমরী লোকেশন কি হবে তা সিস্টেম) । (এর উপর নির্ভর করে

Pass by reference এর মানে হচ্ছে প্যারামিটার হিসেবে কোন ভ্যারিএবল এর ভ্যালু পাস না করে ঐ ভ্যারিএবল এর মেমরী লোকেশন পাঠানো হয়। যার ফলে ফাংশন এর মধ্যে ঐ ভ্যারিএবল এর মান পরিবর্তন করা হলে ঐ পরিবর্তনটা main() function এ ফিরে আসার পরও বজায় থাকে।

নিচের উদাহরনটি দেখুন।

```
#include<stdio.h>

void double_the_value(int *val)
{
    printf("inside the function,value of a:%d\n",*val);
    *val=(*val)*2;
    printf("a has been doubled. a=%d\n",*val);
}

int main()
{
    int a;

    printf("enter an integer:");
    scanf("%d",&a);

    double_the_value(&a);
    printf("returned from function and value of a:%d\n",a);

    return 0;
}
```

উপরের উদাহরনটিতে ভ্যারিএবল এর লোকেশন পাঠানো হয়েছে। ফলে ফাংশন এর মধ্যে ভ্যালুর পরিবর্তন হলে তা main() function এও বজায় থাকে। ফাংশন কল করার সময় মেমরী এড্রেস পাঠানো হলে সেইটি হল pass by reference. **** প্যারামিটার হিসেবে যদি কোন array পাস করা হয় তবে তা pass by reference হয়। যেমনঃ আমরা যখন কোন ক্যারেক্টার অ্যারে পাস করি তখন ঐ অ্যারের কোন value change করলে তা main() function এও চেঞ্জ থাকে।

ফাংশন এর আলোচনায় recursion কিছুটা জটিল। recursion নিয়ে কাল পারলে একটি সম্পুর্ন টিউন করব। আজ এ পর্যন্তই। গতকালের সমাধানগুলো পেতেএখানে ক্লিক করুন।

আমার টিউন দেখে কেউ যদি উপকৃত হন, তবে আমার কষ্ট সার্থক হচ্ছে বলে মনে করি। আর একটি কথা। আপনাদের মতামত, অভিমত কিংবা সি নিয়ে

ধাপে ধাপে সি ল্যাঙ্গুয়েজ শিখুন

কেমন আছেন সবাই। আশা করি আল্লাহর রহমতে ভাল আছেন। আজ আমরা সি এর অতি গুরুত্বপূর্ন ব্যাপার নিয়ে আলোচনা করব।আর তা হল Recursion.

Recursion কি?

যখন কোন একটি ফাংশন এর মধ্য হতেই ঐ ফাংশনটিকে আবার কল করা হয় যতক্ষন না কোন স্পেসিফিক কন্ডিশন পূরন না হচ্ছে, তখন ঐ ধরনের কলকে recursion এবং ঐ ফাংশনটিকে recursive function বলা হয়। প্রোগ্রাম এর ক্ষেত্রে recursion এর ব্যবহার কোড অনেক ছোট ও সহজ করে তোলে। recursion যদিও সহজ তবে তা আয়ত্ব করতে এবং সঠিক প্রয়োগ ঘটাতে

প্রয়োজন অনেক অনেক প্র্যাকটিস। আজ আমরা অল্প কিছু প্রবলেম recursion এর মাধ্যমে করব এবং তা ভালভাবে বোঝার চেস্টা করব।

Recursion এর উদাহরনঃ

এবার আমরা recursive function এর প্রথম প্রবলেমটা করব। ম্যাথম্যাটিক্স নিয়ে যাদের কিছুটা ধারনা আছে তারা অবশ্যই factorial সম্পর্কে জেনে থাকবেন। এখানে একবার উল্লেখ করছি।কোন সংখ্যার factorial হল 1 থেকে ঐ নাম্বার পর্যন্ত সকল সংখ্যার গুনফল।

n! = 1*2*3*4*....*n

3! = 1*2*3 = 6

5! = 1*2*3*4*5 = 120

exception: 0! = 1

এবার যেকোন নাম্বার এর factorial বের করার জন্য আমরা loop ব্যবহার করে বের করতে পারি।

যেমনঃ

```
for( i=res=1 ; i<=n ; i++ )
res=res*i;
printf(" factorial of %d : %d\n" , n, res );
```

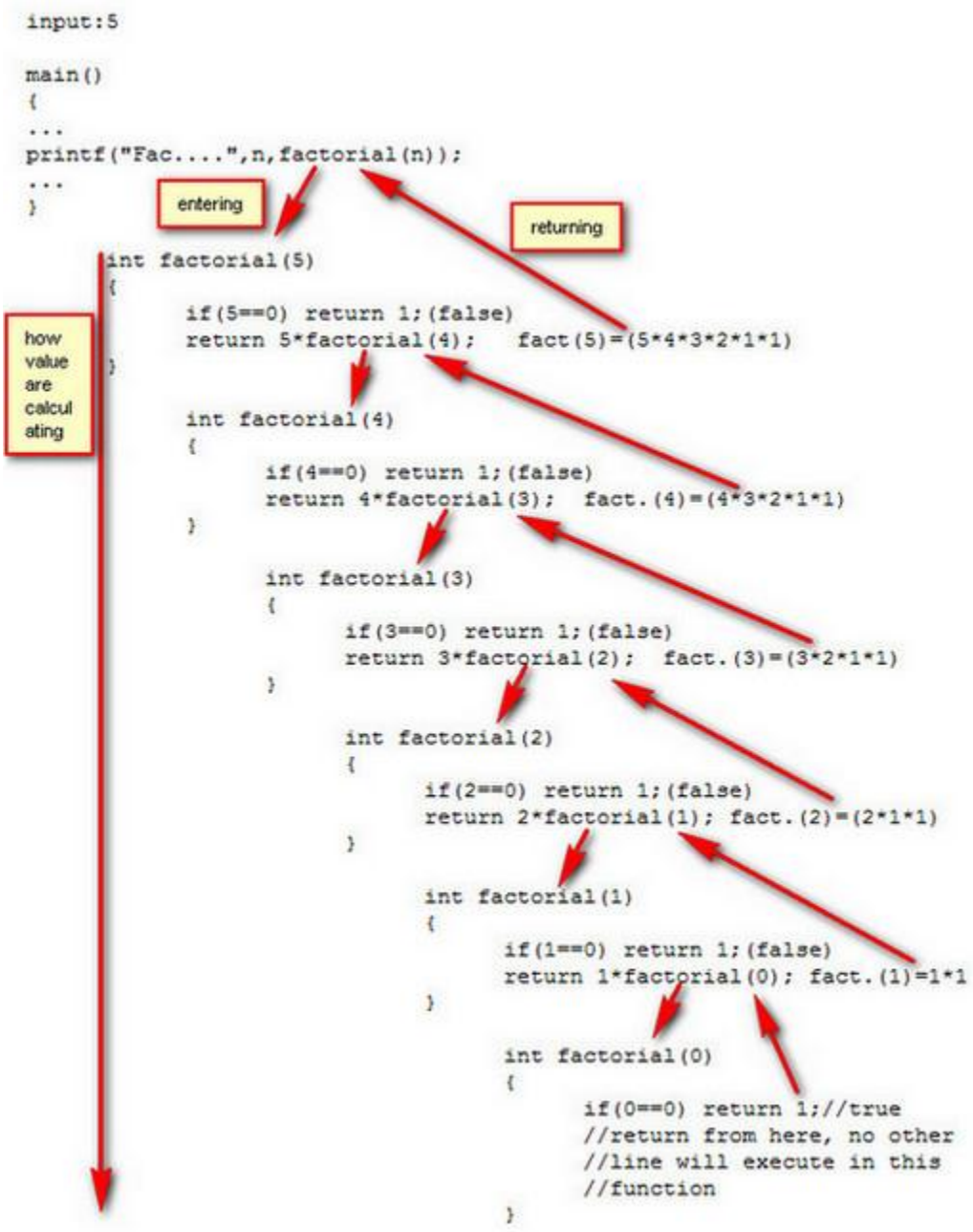
উপরের কোডটি দিয়ে আমরা n এর factorial বের করতে পারি।

এই for loop এর কোডটা আমরা recursion এর মাধ্যমে করতে পারি। উল্লেখ্য, আমরা এখন যে উদাহরনটি করছি তা মূলত recursion বোঝার জন্য। এই প্রোগ্রাম দেখে হয়ত মনে হতে পারে recursion এর কি দরকার। recursion এর আরও অনেক প্রোগ্রাম আমরা দেখব পরে।

```c
#include<stdio.h>

long factorial(long n)
{
    if(n==0) return 1;  //"0! = 1" if condition is for
                        //terminating the recursive call

    return n*factorial(n-1);    //recursive call
}

int main()
{
    long n;

    printf("enter a number:");
    scanf("%d",&n);

    printf("Factorial of %d: %d\n",n,factorial(n));

    return 0;
}
```

যখন কোন ফাংশন কল করা হয়, তখন যেখান থেকে কল করা হয়েছে সেখান থেকে সরাসরি জাম্প করে ফাংশন এ চলে যায়। আর তারপর ঐ ফাংশন এর কাজ শেষ হলে আবার কল এর জায়গায় ফিরে আসে। আর recursion এ যেহেতু একই ফাংশন হতে নিজেকেই আবার কল হয়, তাই ফাংশন কল শুধু depth এ যেতে থাকে।

নিচের ছবিটিতে উপরের উদাহরটি কিভাবে execute হচ্ছে তা দেখানো হল।



যখন প্রোগ্রামটি depth এ যেতে যেতে 0(zero) এর জন্য এক্সিকিউট হচ্ছে তখন ঐ ফাংশন এর স্পেসিফিক কন্ডিশন পূরন হওয়ায় তা ঐখান থেকে return করে যেখান থেকে কল করা হয়েছিল সেখানে ফিরে আসে।

এবার recursion apply করে আরেকটি প্রবলেম করি। আপনি একটি নাম্বার ইনপুট নিয়ে 1 থেকে ঐ নাম্বার পর্যন্ত প্রিন্ট করব এবং এদের যোগফল বের করব।

```c
#include<stdio.h>

int recursion(int n)
{
    int sum=0;
    if(n==1) sum=1;
    else sum=n+recursion(n-1);
    printf("%d ",n);
    return sum;
}

int main()
{
    int n,sum;

    printf("enter a number:");
    scanf("%d",&n);
    printf("number sequentially from 1 to n\n");
    sum=recursion(n);
    printf("\nsum: %d\n",sum);

    return 0;
}
```

আপনারা যদি এর আগের উদাহরনটি কিভাবে কাজ করছে বুঝতে পারেন, ঠিক সেভাবে এটাও depth এ গিয়ে কিভাবে কাজ করছে তা ধরতে পারবেন।

আরেকটি উদাহরন করি। আপনি কোন স্ট্রিং এর reverse করবেন কোন loop ছাড়া। এটি recursion দিয়ে সহজে করা যায়।

```c
#include<stdio.h>

void reverse(int pos,char str[])
{
    if(str[pos]=='\0') return;
    else
        reverse(pos+1,str);
    printf("%c",str[pos]);
}

int main()
{
    char str[100];

    printf("enter a string:");
    scanf("%s",str);
    printf("reverse of the string:");
    reverse(0,str);//string starts from index 0
    printf("\n");

    return 0;
}
```

এইটার সমাধান করার মূল কনসেপ্টটা হল যখন depth এ যেতে যেতে '\০' বা null character পাবে তখন return করার সময় একটি করে ক্যারেক্টার প্রিন্ট করবে। আর তার ফলে স্ট্রিংটিও রিভার্স অর্ডার এ প্রিন্ট হবে।

recursion ব্যবহার করে অনেক ধরনের কাজ খুব সহজে করা যায়। বিভিন্ন algorithm যেমনঃ sorting, searching প্রভৃতি ক্ষেত্রে recursion এর ব্যপক ব্যবহার। এখানে আমি মূলত recursion এর ক্ষেত্রে কি ঘটে তা দেখানোর চেস্টা করেছি। বিভিন্ন algorithm নিয়ে পরবর্তীতে আলোচনা করলে আবারও recursion তুলে ধরব।

কোড লাইব্রেরি

ইনডেক্সঃ

কন্টেস্ট টেমপ্লেট

?

```
01 /*
02 USER: zobayer
03 TASK:
04 */
05
06 #include <cassert>
07 #include <cctype>
08 #include <cmath>
09 #include <cstdio>
10 #include <cstdlib>
11 #include <cstring>
12 #include <iostream>
13 #include <sstream>
14 #include <iomanip>
15 #include <string>
16 #include <vector>
17 #include <list>
18 #include <set>
19 #include <map>
20 #include <stack>
21 #include <queue>
22 #include <algorithm>
```

```
#include <iterator>
#include <utility>
using namespace std;

template< class T > T _abs(T n) { return (n < 0 ? -n : n); }
template< class T > T _max(T a, T b) { return (!(a < b) ? a : b); }
template< class T > T _min(T a, T b) { return (a < b ? a : b); }
template< class T > T sq(T x) { return x * x; }
template< class T > T gcd(T a, T b) { return (b != 0 ? gcd<T>(b, a%b) : a); }
template< class T > T lcm(T a, T b) { return (a / gcd<T>(a, b) * b); }
template< class T > bool inside(T a, T b, T c) { return a<=b && b<=c; }
template< class T > void setmax(T &a, T b) { if(a < b) a = b; }
template< class T > void setmin(T &a, T b) { if(b < a) a = b; }

#define MP(x, y) make_pair(x, y)
#define REV(s, e) reverse(s, e)
#define SET(p) memset(p, -1, sizeof(p))
#define CLR(p) memset(p, 0, sizeof(p))
#define MEM(p, v) memset(p, v, sizeof(p))
#define CPY(d, s) memcpy(d, s, sizeof(s))
#define READ(f) freopen(f, "r", stdin)
#define WRITE(f) freopen(f, "w", stdout)
#define ALL(c) c.begin(), c.end()
#define SIZE(c) (int)c.size()
#define PB(x) push_back(x)
#define ff first
#define ss second
```

```
50 #define i64 long long
51 #define ld long double
52 #define pii pair< int, int >
53 #define psi pair< string, int >
54
55 const double EPS = 1e-9;
56 const double BIG = 1e19;
57 const int INF = 0x7f7f7f7f;
58
59 int main() {
60     //READ("in.txt");
61     //WRITE("out.txt");
62     return 0;
63 }
```

ফাস্ট রিডার

?

```
01 /*
02 In gcc/g++ fread_unlocked() is even faster
03 You can make in non-object-oriented to make faster
04 In contest, who cares dynamic allocation?
05 */
06
07 struct FastRead {
08     char *buff, *ptr;
```

```
09    FastRead(int size) {
10        buff = new char[size];
11        ptr = buff;
12        fread(buff, size, 1, stdin);
13    }
14    ~FastRead() {
15        delete[] buff;
16    }
17    int nextInt()
    {
18        int ret = 0;
19        while(*ptr < '0' || *ptr > '9') ptr++;
20        do { ret = ret * 10 + *ptr++ - '0';
21        } while(*ptr >= '0' && *ptr <= '9');
22        return ret;
23    }
24 };
```


ট্রাই (প্রিফিক্স ট্রি)

```
01 /*
02 Basic trie
03 all operation has complexity O(length)
04 MAX is number of different items
05 */
```

```
06
07 struct trie {
08    trie *next[MAX+1];
09    trie() { for(int i=0; i<=MAX; i++) next[i] = NULL; }
10 };
11
12 void insert(trie *root, int *seq, int len) {
13    trie *curr = root;
14    for(int i = 0; i < len; i++) {
15       if(!curr->next[seq[i]]) curr->next[seq[i]] = new trie;
16       curr = curr->next[seq[i]];
17    }
18    if(!curr->next[MAX]) curr->next[MAX] = new trie;
19 }
20
21 bool found(trie *root, int *seq, int len) {
22    trie *curr = root;
23    for(int i = 0; i < len; i++) {
24       if(!curr->next[seq[i]]) return false;
25       curr = curr->next[seq[i]];
26    }
27    if(!curr->next[MAX]) return false;
28    return true;
29 }
```

মিনিমাম এক্সপ্রেশন (ঝো-জুয়ান)

```
01 /*
```

```
02 Finds alphabetically first representation of a cyclic string in O(length)
03 It concatenates the string with itself,
04 Array must be at least double the maximum size.
05 After use, it reverts the string to original one.
06 */
07
08 inline int minimumExpression(char *s) {
09     int i, j, k, len = strlen(s);
10     memcpy(&s[len], s, len);
11     len <<= 1, i = 0, j = 1, k = 0;
12     while(i + k < len && j + k < len) {
13         if(s[i+k] == s[j+k]) k++;
14         else if(s[i+k] > s[j+k]) { i = i+k+1; if(i <= j) i = j+1; k = 0; }
15         else if(s[i+k] < s[j+k]) { j = j+k+1; if(j <= i) j = i+1; k = 0; }
16     }
17     str[len>>1] = 0;
18     return min(i, j);
19 }
```

জোসেফাস রিকারেন্স

?

```
01 /*
02 The first one is for K = 2 and the second one is general.
03 Note: first function returns 1 based index while second one is 0 based.
04 */
05
06 int f(int n) {
```

```
07    if(n == 1) return 1;
08    return (f((n-(n&1))>>1)<<1) + ((n&1)?1:-1);
09 }
10
11 int f(int n, int k) {
12    if(n == 1) return 0;
13    return (f(n-1, k) + k)%n;
14 }
```

কনভেক্স হাল (গ্রাহাম'স স্ক্যান)

?

```
01 /*
02 ConvexHull : Graham's Scan O(n lg n), integer implementation
03 P[]: holds all the points, C[]: holds points on the hull
04 np: number of points in P[], nc: number of points in C[]
05 to handle duplicate, call makeUnique() before calling convexHull()
06 call convexHull() if you have np >= 3
07 to remove co-linear points on hull, call compress() after convexHull()
08 */
09
10 point P[MAX], C[MAX], P0;
11
12 inline int triArea2(const point &a, const point &b, const point &c) {
13    return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
14 }
15
16 inline int sqDist(const point &a, const point &b) {
```

```
17    return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
18 }
19
20 inline bool comp(const point &a, const point &b) {
21    int d = triArea2(P0, a, b);
22    if(d < 0) return false;
23    if(!d && sqDist(P0, a) > sqDist(P0, b)) return false;
24    return true;
25 }
26
27 inline bool normal(const point &a, const point &b) {
28    return ((a.x==b.x) ? a.y < b.y : a.x < b.x);
29 }
30
31 inline bool issame(const point &a, const point &b) {
32    return (a.x == b.x && a.y == b.y);
33 }
34
35 inline void makeUnique(int &np)
   {
36    sort(&P[0], &P[np], normal);
37    np = unique(&P[0], &P[np], issame) - P;
38 }
39
40 void convexHull(int &np, int &nc) {
41    int i, j, pos = 0;
42    for(i = 1; i < np; i++)
```

```
43      if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x<P[pos].x))
44          pos = i;
45    swap(P[0], P[pos]);
46    P0 = P[0];
47    sort(&P[1], &P[np], comp);
48    for(i = 0; i < 3; i++) C[i] = P[i];
49    for(i = j = 3; i < np; i++) {
50       while(triArea2(C[j-2], C[j-1], P[i]) < 0) j--;
51       C[j++] = P[i];
52    }
53    nc = j;
54 }
55
56 void compress(int &nc) {
57    int i, j, d;
58    C[nc] = C[0];
59    for(i=j=1; i < nc; i++) {
60       d = triArea2(C[j-1], C[i], C[i+1]);
61       if(d || (!d && issame(C[j-1], C[i+1]))) C[j++] = C[i];
62    }
63    nc = j;
64 }
```

পয়েন্ট ইন কনভেক্স পলিগন

?

```
01 /*
02 C[] array of points of convex polygon in ccw order,
```

```
03 nc number of points in C, p target points.
04 returns true if p is inside C (including edge) or false otherwise.
05 complexity O(lg n)
06 */
07
08 inline bool inConvexPoly(point *C, int nc, const point &p) {
09    int st = 1, en = nc - 1, mid;
10    while(en - st > 1) {
11       mid = (st + en)>>1;
12       if(triArea2(C[0], C[mid], p) < 0) en = mid;
13       else st = mid;
14    }
15    if(triArea2(C[0], C[st], p) < 0) return false;
16    if(triArea2(C[st], C[en], p) < 0) return false;
17    if(triArea2(C[en], C[0], p) < 0) return false;
18    return true;
19 }
```

পলিগন এরিয়া (2D)

?

```
01 /*
02 P[] holds the points, must be either in cw or ccw
03 function returns double of the area.
04 */
05
06 inline int dArea(int np) {
07    int area = 0;
```

```
08    for(int i = 0; i < np; i++) {
09        area += p[i].x*p[i+1].y - p[i].y*p[i+1].x;
10    }
11    return abs(area);
12 }
```

স্ট্রংলি কানেকটেড কম্পোনেন্ট (টারজান)

?

```
01 /*
02 SCC (Tarjan) in O(|v| + |e|)
03 Input:
04 G[] is a input directed graph with n nodes in range [1,n]
05 Output:
06 Component[i] holds the component id to which node i belongs
07 components: total number of components in the graph
08 */
09
10 int Stack[MAX], top;
11 int Index[MAX], Lowlink[MAX], Onstack[MAX];
12 int Component[MAX];
13 int idx, components;
14 vector< int > G[MAX];
15
16 void tarjan(int u) {
```

```
17    int v, i;
18    Index[u] = Lowlink[u] = idx++;
19    Stack[top++] = u;
20    Onstack[u] = 1;
21    for(i = 0; i < SZ(G[u]); i++) {
22       v = G[u][i];
23       if(Index[v]==-1) {
24          tarjan(v);
25          Lowlink[u] = min(Lowlink[u], Lowlink[v]);
26       }
27       else if(Onstack[v]) Lowlink[u] = min(Lowlink[u], Index[v]);
28    }
29    if(Lowlink[u] == Index[u]) {
30       components++;
31       do {
32          v = Stack[--top];
33          Onstack[v] = 0;
34          Component[v] = components;
35       } while(u != v);
36    }
37 }
38
39 void findSCC(int n) {
40    components = top = idx = 0;
41    SET(Index); CLR(Onstack); MEM(Lowlink, 0x3f);
42    for(int i = 1; i <= n; i++) if(Index[i]==-1) tarjan(i);
43 }
```

আর্টিকুলেশন পয়েন্ট

?

```
01 /*
02 G[][]: undirected connected graph
03 cut[v] is true if node v is an articulation point / cut-vertex
04 */
05
06 vector< int > G[MAX];
07 int fup[MAX], tin[MAX], used[MAX], cut[MAX], dfstime;
08
09 void dfs(int u, int par = -1) {
10    int i, v, child = 0;
11    used[u] = 1;
12    tin[u] = fup[u] = ++dfstime;
13    for(i = 0; i < G[u].size(); i++) {
14       v = G[u][i];
15       if(v == par) continue;
16       if(used[v]) fup[u] = min(fup[u], tin[v]);
17       else {
18          child++;
19          dfs(v, u);
20          fup[u] = min(fup[u], fup[v]);
21          if(fup[v] >= tin[u]) cut[u] = 1;
22       }
23    }
24    if(par == -1) cut[u] = child > 1;
```

```
25 }
```

ব্রিজ

?

```
01 /*
02 G[][]: undirected graph
03 finds all the bridges in a connected graph and
04 adds those edges to the Bridges[] vector
05 */
06
07 vector< int > G[MAX];
08 vector< pair< int, int > > Bridges;
09 int fup[MAX], tin[MAX], used[MAX], dfstime;
10
11 void dfs(int u, int par) {
12    int i, v;
13    used[u] = 1;
14    tin[u] = fup[u] = ++dfstime;
15    for(i = 0; i < G[u].size(); i++) {
16       v = G[u][i];
17       if(v == par) continue;
18       if(used[v]) fup[u] = min(fup[u], tin[v]);
19       else {
20          dfs(v, u);
21          fup[u] = min(fup[u], fup[v]);
22          if(fup[v] > tin[u]) Bridges.push_back(make_pair(u, v));
23       }
```

```
24    }
25 }
```

স্টেবল ম্যারিজ

?

```
01 /*
02 INPUT:
03 m: number of man, n: number of woman (must be at least as large as m)
04 L[i][]: the list of women in order of decreasing preference of man i
05 R[j][i]: the attractiveness of i to j.
06 OUTPUTS:
07 L2R[]: the mate of man i (always between 0 and n-1)
08 R2L[]: the mate of woman j (or -1 if single)
09 man priority
10 */
11
12 int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM], R2L[MAXW], p[MAXM];
13
14 void stableMarriage() {
15    int i, man, wom, hubby;
16    SET(R2L); CLR(p);
17    for(i = 0; i < m; i++ ) {
18       man = i;
19       while(man >= 0) {
20          while(true) {
21             wom = L[man][p[man]++];
22             if(R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]]) break;
```

```
23          }
24          hubby = R2L[wom];
25          R2L[L2R[man] = wom] = man;
26          man = hubby;
27       }
28    }
29 }
```

ম্যাক্স ফ্লো (ডিনিক)

?

```
01 /*
02 max flow (dinitz algorithm)
03 works on undirected graph
04 can have loops, multiple edges, cycles
05 */
06
07 int src, snk, nNode, nEdge;
08 int Q[MAXN], fin[MAXN], pro[MAXN], dist[MAXN];
09 int flow[MAXE], cap[MAXE], next[MAXE], to[MAXE];
10
11 inline void init(int _src, int _snk, int _n) {
12    src = _src, snk = _snk, nNode = _n, nEdge = 0;
13    SET(fin);
14 }
15
16 inline void add(int u, int v, int _cap) {
17    to[nEdge] = v, cap[nEdge] = _cap, flow[nEdge] = 0;
```

```
18    next[nEdge] = fin[u], fin[u] = nEdge++;
19    to[nEdge] = u, cap[nEdge] = _cap, flow[nEdge] = 0;
20    next[nEdge] = fin[v], fin[v] = nEdge++;
21 }
22
23 bool bfs() {
24    int st, en, i, u, v;
25    SET(dist);
26    dist[src] = st = en = 0;
27    Q[en++] = src;
28    while(st < en) {
29       u = Q[st++];
30       for(i=fin[u]; i>=0; i=next[i]) {
31          v = to[i];
32          if(flow[i] < cap[i] && dist[v]==-1) {
33             dist[v] = dist[u]+1;
34             Q[en++] = v;
35          }
36       }
37    }
38    return dist[snk]!=-1;
39 }
40
41 int dfs(int u, int fl) {
42    if(u==snk) return fl;
43    for(int &e=pro[u], v, df; e>=0; e=next[e]) {
44       v = to[e];
```

```
45      if(flow[e] < cap[e] && dist[v]==dist[u]+1) {
46          df = dfs(v, min(cap[e]-flow[e], fl));
47          if(df>0) {
48              flow[e] += df;
49              flow[e^1] -= df;
50              return df;
51          }
52      }
53  }
54  return 0;
55 }
56
57 i64 dinitz() {
58  i64 ret = 0;
59  int df;
60  while(bfs()) {
61      for(int i=1; i<=nNode; i++) pro[i] = fin[i];
62      while(true) {
63          df = dfs(src, INF);
64          if(df) ret += (i64)df;
65          else break;
66      }
67  }
68  return ret;
69 }
```


মিন কস্ট ম্যাক্স ফ্লো (বেলম্যান ফোর্ড)

?

```
01 /*
02 min cost flow (bellman ford)
03 works only on directed graphs
04 handles multiple edges, cycles, loops
05 */
06
07 int src, snk, nNode, nEdge;
08 int fin[MAXN], pre[MAXN], dist[MAXN];
09 int cap[MAXE], cost[MAXE], next[MAXE], to[MAXE], from[MAXE];
10
11 inline void init(int _src, int _snk, int nodes) {
12     SET(fin);
13     nNode = nodes, nEdge = 0;
14     src = _src, snk = _snk;
15 }
16
17 inline void addEdge(int u, int v, int _cap, int _cost) {
18     from[nEdge] = u, to[nEdge] = v, cap[nEdge] = _cap, cost[nEdge] = _cost;
19     next[nEdge] = fin[u], fin[u] = nEdge++;
20     from[nEdge] = v, to[nEdge] = u, cap[nEdge] = 0, cost[nEdge] = -(_cost);
21     next[nEdge] = fin[v], fin[v] = nEdge++;
22 }
23
24 bool bellman() {
25     int iter, u, v, i;
26     bool flag =
```

```
true;
    MEM(dist, 0x7f);
    SET(pre);
    dist[src] = 0;
    for(iter = 1; iter < nNode && flag; iter++) {
        flag = false;
        for(u = 0; u < nNode; u++) {
            for(i = fin[u]; i >= 0; i = next[i]) {
                v = to[i];
                if(cap[i] && dist[v] > dist[u] + cost[i]) {
                    dist[v] = dist[u] + cost[i];
                    pre[v] = i;
                    flag = true;
                }
            }
        }
    }
    return (dist[snk] < INF);
}

int mcmf(int &fcost) {
    int netflow, i, bot, u;
    netflow = fcost = 0;
    while(bellman()) {
        bot = INF;
        for(u = pre[snk]; u >= 0; u = pre[from[u]]) bot = min(bot, cap[u]);
        for(u = pre[snk]; u >= 0; u = pre[from[u]]) {
```

```
53          cap[u] -= bot;
54          cap[u^1] += bot;
55          fcost += bot * cost[u];
56       }
57       netflow += bot;
58    }
59    return netflow;
60 }
```

ম্যাক্সিমাম বাইপারটাইট ম্যাচিং (ডি.এফ.এস.)

?

```
01 /*
02 G[] is the left-side graph, must be bipartite
03 match(n): n is the number of nodes in left-side set
04 and returns the maximum possible matching.
05 Left[] anf Right[] ar assigned with corresponding matches
06 */
07
08 vector < int > G[MAX];
09 bool visited[MAX];
10 int Left[MAX], Right[MAX];
11
12 bool dfs(int u) {
13    if(visited[u]) return false;
14    visited[u] = true;
15    int len = G[u].size(), i, v;
16    for(i=0; i<len; i++) {
```

```
17      v = G[u][i];
18      if(Right[v]==-1) {
19          Right[v] = u, Left[u] = v;
20          return true;
21      }
22  }
23  for(i=0; i<len; i++) {
24      v = G[u][i];
25      if(dfs(Right[v])) {
26          Right[v] = u, Left[u] = v;
27          return true;
28      }
29  }
30  return false;
31 }
32
   int match(int n)
33 {
34  int i, ret = 0;
35  bool done;
36  SET(Left); SET(Right);
37  do {
38      done = true; CLR(visited);
39      for(i=0; i<n; i++) {
40          if(Left[i]==-1 && dfs(i)) {
41              done = false;
42          }
```

```
43      }
44    } while(!done);
45    for(i=0; i<n; i++) ret += (Left[i]!=-1);
46    return ret;
47 }
```

ম্যাক্সিমাম বাইপারটাইট ম্যাচিং (হপক্রফট কার্প)

?

```
01 /*
02 n: number of nodes on left side, nodes are numbered 1 to n
03 m: number of nodes on right side, nodes are numbered n+1 to n+m
04 G = NIL[0] ? G1[G[1---n]] ? G2[G[n+1---n+m]]
05 */
06
07 bool bfs() {
08    int i, u, v, len;
09    queue< int > Q;
10    for(i=1; i<=n; i++) {
11       if(match[i]==NIL) {
12          dist[i] = 0;
13          Q.push(i);
14       }
15       else dist[i] = INF;
16    }
17    dist[NIL] = INF;
18    while(!Q.empty()) {
19       u = Q.front(); Q.pop();
```

```
20      if(u!=NIL) {
21          len = G[u].size();
22          for(i=0; i<len; i++) {
23              v = G[u][i];
24              if(dist[match[v]]==INF) {
25                  dist[match[v]] = dist[u] + 1;
26                  Q.push(match[v]);
27              }
28          }
29      }
30  }
31  return (dist[NIL]!=INF);
32 }
33
34 bool dfs(int u) {
35  int i, v, len;
36  if(u!=NIL) {
37      len = G[u].size();
38      for(i=0; i<len; i++) {
39          v = G[u][i];
40          if(dist[match[v]]==dist[u]+1) {
41              if(dfs(match[v])) {
42                  match[v] = u;
43                  match[u] = v;
44                  return true;
45              }
46          }
```

```
47        }
48        dist[u] = INF;
49        return false;
50    }
51    return true;
52 }
53
54 int hopcroft_karp() {
55    int matching = 0, i;
56    CLR(match);
57    while(bfs())
58        for(i=1; i<=n; i++)
59            if(match[i]==NIL && dfs(i))
60                matching++;
61    return matching;
62 }
```

ডিসজয়েন্ট সেট

?

```
01 /*
02 disjoint set data-structure
03 implements union by rank and path compression
04 */
05
06 struct DisjointSet {
07     int *root, *rank, n;
```

```
    DisjointSet(int sz)
08  {
09      root = new int[sz+1];
10      rank = new int[sz+1];
11      n = sz;
12  }
13  ~DisjointSet() {
14      delete[] root;
15      delete[] rank;
16  }
17  void init() {
18      for(int i = 1; i <= n; i++) {
19          root[i] = i;
20          rank[i] = 0;
21      }
22  }
23  int find(int u) {
24      if(u != root[u]) root[u] = find(root[u]);
25      return root[u];
26  }
27  void merge(int u, int v) {
28      int pu = find(u);
29      int pv = find(v);
30      if(rank[pu] > rank[pv]) root[pv] = pu;
31      else root[pu] = pv;
32      if(rank[pu]==rank[pv]) rank[pv]++;
33  }
```

```
34 };
```

ডাবল হ্যাশ

?

```
01 /*
02 M > N and should be close, better both be primes.
03 M should be as much large as possible, not exceeding array size.
04 HKEY is the Hash function, change it if necessary.
05 */
06
07 #define NIL -1
08 #define M 1021
09 #define N 1019
10 #define HKEY(x,i) ((x)%M+(i)*(1+(x)%N))%M
11
12 int a[M+1];
13
14 inline int hash(int key) {
15     int i = 0, j;
16     do {
17         j = HKEY(key, i);
18         if(a[j]==NIL) { a[j] = key; return j; }
19         i++;
20     } while(i < M);
21     return -1;
22 }
23
```

```
24 inline int find(int key) {
25     int i = 0, j;
26     do {
27         j = HKEY(key, i);
28         if(a[j]==key) return j;
29         i++;
30     } while(a[j]!=NIL && i < M);
31     return -1;
32 }
```

C++ STL :: vector

ভেক্টরঃ

অ্যারে আমাদের সবারই বিশেষভাবে পরিচিত, এবং সবার খুবই প্রিয় একটা ডাটা-স্ট্রাকচার হল অ্যারে। কোন কোডে অ্যারে ব্যবহার করতে পারলেই কেমন যেন শান্তি শান্তি লাগে... অ্যারের সবচেয়ে আকর্ষনীয় ফিচার মনে হয় তার ইন্ডেক্সিং (মানে র‍্যান্ডম অ্যাকসেস)। কিন্তু যতই দিন যায়, সব কেমন যেন কঠিন হয়ে যায়... তাইতো আজকাল অ্যারে নিয়েও মাঝে মাঝে ঝামেলায় পড়তে হয়, বিশেষতঃ যখন কিনা অ্যারের আকার আকৃতি রানটাইমে পরিবর্তন করার দরকার হয়। কারন, অ্যারে একবার ডিক্লেয়ার করলে রানটাইমে C++ এ তা আর ছোট-বড় করা যায় না। আর এখানেই ভেক্টরের আবির্ভাব।

ভেক্টরকে বলা যেতে পারে একটা ডায়নামিক অ্যারে, দরকার মত যেটার সাইজ বাড়ানো কমানো যায়। আবার অ্যারের মত মাল্টি ডাইমেনশন এবং ইন্ডেক্সিং সুবিধাগুলো ব্যবহার করা যায়।

কিভাবে ব্যবহার করবো?

ভেক্টর C++ STL এর একটা কন্টেইনার ক্লাস। অর্থাৎ এটাও একটা টেমপ্লেট ক্লাস, যার কারনে এটাকে অন্য টেমপ্লেটের সাথে সহজেই ব্যবহার করা যায়। তবে সবার প্রথমে দরকার স্ট্যান্ডার্ড হেডার vector ইনক্লুড করাঃ

?

```
1 #include <vector>
2 using namespace std;
```

এর পরের কাজ হল ভেক্টর ডিক্লারেশন, সেটাও আবার কয়েকভাবে করা যায়, তবে সাধারন স্টাইল হলঃ vector < type > Var; এখানে type হতে পারে যে কোন টাইপ, অন্য কোন ইউজার ডিফাইনড টাইপ, টেমপ্লেট ক্লাস টাইপ, বেসিক টাইপগুলো। ভেক্টর ডিক্লেয়ার করার সাধারন সিন্ট্যাক্সগুলো নিচে দেখানো হলঃ

?

```
01 // constructing vectors
02 #include <iostream>
03 #include <vector>
04 using namespace std;
05
06 int main () {
07   // constructors used in the same order as described above:
08   vector<int> first;                                // empty vector of ints
09   vector<int> second (4,100);                       // four ints with value 100
10   vector<int> third (second.begin(),second.end());  // iterating through second
11   vector<int> fourth (third);                       // a copy of third
12
13   // the iterator constructor can also be used to construct from arrays:
14   int myints[] = {16,2,77,29};
15   vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );
16
17   cout << "The contents of fifth are:";
18   for (unsigned i=0; i < fifth.size(); i++) cout << " " << fifth[i]; cout << endl;
19
20   return 0;
21 }
```

ভেক্টর ডিক্লেয়ার করার সময় [] আর () অপারেটরের মধ্যে পার্থক্যটা খেয়াল করা উচিতঃ

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 int main () {
06    // use of [] and ()
07    vector<int> v1[100]; // creates an array of vectors, i.e. 100 vectors
08    vector<int> v2(100); // creates 1 vector of size 100
09
10    // creating a 2D array 100x2 where each element is a vector,
11    // so in total, it is a 3D structure, as vector itself is 1D
12    vector<int> v3[100][2];
13    return 0;
14 }
```

উপরের কোডে যেমন দেখা গেল... চাইলেই আমরা ভেক্টরের অ্যারে তৈরি করতে পারি (এটা ভেক্টরের অন্যতম একটা গুরুত্বপূর্ন ব্যবহার)। কিন্তু এছাড়াও ভেক্টরে দিয়ে মাল্টিডাইমেনশনাল স্ট্রাকচার তৈরি করা যায়, যেখানে প্রতিটা ডাইমেনশনই ডায়নামিকঃ

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 int main () {
06    // multidimentional vector
07    vector< vector< int > > V2D;
```

```
08    // the above is basically a vector where each element is a vector,
09    // we can also increase the level of nesting if needed.
10
11    // demonstrate a triangular structure..
12    vector< int > temp;
13    for(int i = 0; i < 10; i++) {
14       temp.clear();
15       for(int j = 0; j <= i; j++) {
16          temp.push_back(i+1);
17       }
18       V2D.push_back(temp);
19    }
20    return 0;
21 }
```

হুম, তো আমরা খালি একটা ভেক্টরের মধ্যে আরেকটা ভেক্টর পুশ করলাম, ফলাফল 2D ভেক্টর। আগেই বলা হয়েছে, ভেক্টর একটা টেমপ্লেট ক্লাস, তাই, যে কোন টাইপ নিয়ে এটা বানানো যায়। যেমন চাইলেই আমরা কিউএর একটা ভেক্টর বানাতে পারিঃ vector< queue > Vq;

ভেক্টর অ্যাকসেসঃ

ভেক্টর দুইভাবে অ্যাকসেস করা যায়, ইটারেটরের সাহায্যে, ইন্ডেক্সিং-এর সাহায্যে। ইটারেটরের সাহায্যে করলে কিঞ্চিত ফাস্ট হয়, ভেক্টরের ইটারেটর একটা র‍্যান্ডম অ্যাকসেস ইটারেটর।

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 int main () {
06    vector< vector< int > > V2D;
```

```
07
08  // creating a 2D triangular structure
09  for(int i = 0; i < 10; i++) {
10      vector< int > temp;
11      for(int j = 0; j <= i; j++) {
12          temp.push_back(i);
13      }
14      V2D.push_back(temp);
15  }
16
17  // using iterator
18  cout << "using iterator:\n";
19  vector< vector< int > > :: iterator outer;
20  vector< int > :: iterator inner;
21  for(outer = V2D.begin(); outer != V2D.end(); outer++) {
22      for(inner = outer->begin(); inner != outer->end(); inner++) {
23          cout << *inner << ' ';
24      }
25      cout << '\n';
26  }
27
28  // using index
29  cout << "\nusing indexes:\n";
30  for(unsigned i = 0; i < V2D.size(); i++) {
31      for(unsigned j = 0; j < V2D[i].size(); j++) {
32          cout << V2D[i][j] << ' ';
33      }
```

```
34      cout << '\n';
35   }
36   return 0;
37 }
```

সাধারনত কেউ ভেক্টরে ইটারেটর ব্যবহার করে না, কারন ইন্ডেক্সের ব্যবহার অনেক সহজ, এবং অধিকাংশ C++ কোডারের পয়েন্টারের প্রতি আজন্মের ভয়। কিন্তু ইটারেটরের ব্যবহার আরো বেশি সিগ্নিফিক্যান্ট, এবং অর্থবহ। অবশ্য কিভাবে ব্যবহার করতে হবে সেটা কোডারের ব্যক্তিগত ইচ্ছা। যার যেটায় সুবিধা সেটাই ব্যবহার করা উচিত। উপরের কোডে ভ্যালুগুলো চাইলে অ্যারের মত করে মডিফাইও করা যাবে, যেমন, V2D[i][j] = 100; লিখে দিলেই ওই পজিশনটার মান ১০০ হয়ে যাবে।

পুশব্যাক, পপব্যাক, সাইজ, এম্পটি

আগের কোডটায় আমরা দেখলাম পুশব্যাক দিয়ে ভেক্টরে ভ্যালু ইন্সার্ট করা হচ্ছে, তাই আর নতুন করে সেটা দেখনোর কিছু নাই, push_back() ফাংশনের সাহায্যে ভেক্টরের শেষে একি ধরনের একটা আইটেম অ্যাড করা যায়, আর pop_back() দিয়ে ভেক্টরের শেষ থেকে একটা আইটেম হাওয়া করে দেওয়া যায়। অন্যান্য STL মেম্বারের মত ভেক্টরের size() আর empty() ফাংশন দুইটাও আইডেন্টিক্যাল। নিচের কোডে এগুলোর ব্যবহার দেখানো হলঃ

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 int main () {
06   vector< int > simple;
07
08   for(int i = 1; i < 10; i++) {
09      simple.push_back(i*100);
10      cout << "inserting: " << simple.back() << endl;
11   }
```

```
12    cout << "------------------\n";
13
14    while(!simple.empty()) {
15       cout << "size: " << simple.size();
16       cout << " last element: " << simple.back() << endl;
17       simple.pop_back();
18    }
19    cout << "vector empty\n";
20    return 0;
21 }
```

রিসাইজ, ইরেজ, ক্লিয়ার এবং রিসাইজ নিয়ে কিছু কাহিনীঃ

ভেক্টরের সাইজ মডিফায় করা যায় এরকম কিছু মেম্বার হল resize(), erase(), clear(), এদের মধ্যে clear() এর ব্যবহার অন্য যে কোন কন্টেইনার ক্লাসের মতই, সব এলিমেনত ডিলিট করে দেয়, ফলাফম সাইজ ০ হয়ে যায়। আর resize() ফাংশন দিয়ে ভেক্টরের সাইজ পরিবর্তন করা যায়। erase() এর কাজ কোন এলিমেন্ট বা একটা রেঞ্জ ডিলিট করা যায়। erase() এর দুইটা ফরম্যাট আছে, erase(iterator pos), erase(iterator first, iterator last), অর্থাৎ কোন ভ্যালুর সাপেক্ষে ডিলিট করা যায় না, তার ইটারেটর পজিশন দিতে হবে, আর দ্বিতীয় ধরনে ভেক্টরের [first, last) এই রেঞ্জের সব আইটেম ডিলিট করে দিবে। বলাই বাহুল্য, আজগুবি ইটারেটর দিলে রানটাইম এররর খেয়ে বসে থাকতে হবে...

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 int main () {
06    unsigned int i;
07    vector<unsigned int> myvector;
08    // set some values (from 1 to 10)
```

```
09    for (i=1; i<=10; i++) myvector.push_back(i);
10    // erase the 6th element
11    myvector.erase (myvector.begin()+5);
12    // erase the first 3 elements:
13    myvector.erase (myvector.begin(),myvector.begin()+3);
14    cout << "myvector contains:";
15    for (i=0; i<myvector.size(); i++)
16       cout << " " << myvector[i];
17    cout << endl;
18
19    // clear all
20    myvector.clear();
21    cout << "size: " << myvector.size() << endl;
22
23    // resize and then check size
24    myvector.resize(10);
25    cout << "size: " << myvector.size() << endl;
26    return 0;
27 }
```

কিছু বিষয়ে এখানে সতর্ক হতে হবে, যেমনঃ resize() ফাংশনটা ভেক্টরের আগের অবস্থাত কোন তোয়াক্কা না করেই তার সাইজ চেঞ্জ করবে, ফলে কিছু এলিমেন্ট বাদও পড়তে পারে, আবার কিছু জায়গা খালিও থাকতে পারে। কিন্তূ ভেক্টরের সাইজ চেক করলে নতুন সাইজ এ পাওয়া যাবে, যদিও, সেটা হয়তো একটা খালি ভেক্টর ছিল। তাই, resize() এর পরে push_back() ব্যবহার করলে কোডার যাই আশা করুক না কেন, সে রিসাজকৃত স্পেসের পরেই পুশ করবে। যেমন, মনে করি, ভেক্টরে ছিল ১৮ টা আইটেম, আমি সেটাকে রিসাইজ করলাম ৫০ এ। তখন push_back() করলে সে ১৯ তম পজিশনে করবে না, করবে ৫১ তম পজিশনে (মানে ইন্ডেক্স ৫০)। একই ভাবে যদি ১২ রে রিসাইজ করতাম, তাহলেও সে ১৯ এ না করে ১৩ তে পুশ করতো। সাধারনত clear() এর অল্টারনেট হিসাবে resize() ব্যবহার করা যায়, তবে এর সাথে push_back() ব্যবহার না করাই ভাল। নিচের কোডঃ

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 int main () {
06    int n, a;
07    vector< int > v;
08    while(cin >> n) {
09       v.resize(n);
10       for(a = 0; a < n; a++) {
11          cin >> v[a];
12       }
13       for(a = 0; a < n; a++) {
14          cout << v[a] << endl;
15       }
16    }
17    return 0;
18 }
```


erase() এর ব্যাপারেও একটা কথা আছে, তা হল, এর কমপ্লেক্সিটি লিনিয়ার, তাই রান্ডম ইনসার্ট আর ডিলিটের জন্য ভেক্টর সুবিধাজনক না। x.clear() করা আর x.erase(x.begin(), x.end()) একই কথা। তাই কোডে খুব বেশি clear না মারাই ভাল।

ভেক্টরের ইটারেটর পাব কিভাবে?

উপরের কোডগুলোতে প্রায়ই দেখা গেছে begin() আর end() নামের দুটি ফাংশন। এরা যথা ক্রমে ভেক্টরের শুরু আর শেষের ইটারেটর রিটার্ন করে। মনে রাখতে হবে, STL এর যে কোন মেম্বারই একটা কমন রুল ফলো করে, তা হল, এখানে রেঞ্জ ডিফাইন করা হয় [first, last) দিয়ে, তার মানে last টা ইনক্লুডেড না। একই ভাবে

end() ইটারেটর হল লাস্ট আইটেমের পরের ইটারেটর টা। এদের উলটা ফাংশন ২টা হলঃ rbegin(), rend(), যথাক্রমে রিভার্স বিগিন আর রিভার্স এন্ড ইটারেটর রিটার্ন করে।

অনেক হল...

ভেক্টর আসলে অনেক বিশাল একটা ব্যাপার, এটা কিভাবে C++ এ ইমপ্লিমেন্ট করা হয়, সেটাও আরেক মজার ব্যাপার। এক পোস্টে সব বলা সম্ভব না, আর দরকারও নাই, গুগল আছে কি করতে... তবে এই ফাংশন গুলার বাইরে কোনটাই লাগে না। এগুলার এক্সপার্ট মানেই ভেক্টরের এক্সপার্ট, তাও শেষে ভেক্টরের একটা বহুল ব্যবহার দেখাই, সেটা হল, অ্যাডজাসেনসি লিস্ট দিয়ে গ্রাফ রিপ্রেজেন্টেশনঃ

?

```
01 #include <iostream>
02 #include <vector>
03 using namespace std;
04
05 const int MAX = 100;
06 typedef pair< int, int > Edge; // to, weight
07
08 int main () {
09    int n, e, i, j, u, v, w;
10    vector< Edge > G[MAX]; // adjacency list for MAX vertices
11    while(cin >> n >> e) {
12       // n nodes in range [0,n), e edges
13       for(i = 0; i < n; i++) G[i].clear(); // forget previous info
14       for(i = 0; i < e; i++) {
15          // directed edge from u to v with cost w
16          cin >> u >> v >> w;
17          G[u].push_back(Edge(v, w));
18       }
19       // now show the
```

graph

```
20      for(i = 0; i < n; i++) {
21          cout << "Degree of " << i << ": " << G[i].size() << endl;
22          cout << "Adjacents:\n";
23          for(j = 0; j < (int)G[i].size(); j++) {
24              cout << ' ' << G[i][j].first << "(" << G[i][j].second << ")\n";
25          }
26          cout << endl;
27      }
28  }
29  return 0;
30 }
```

রেফারেন্সঃ

ভেক্টরের কারনে কাজ অনেক সহজ হয়ে যায়, তবে ভেক্টর কিছুটা স্লো আর বেশি মেমরি নেয়। যারা অপ্টিমাইজেশন পছন্দ করেন তাদের জন্য এটা খুব একটা মজার কিছু না।

ডিটেইলসঃ http://www.cplusplus.com/reference/stl/vector/

কিছু কথাঃ

যারাই কিনা সি/সি++ নিয়ে বেশকিছুদিন নাড়াচাড়া করছেন, তারা প্রায় সবাই সি এর একটা দারুন জিনিস স্ট্রাকচার-এর সাথে পরিচিত, আর, আরেকটু অ্যাডভান্সডরা তো মনে হয় অলরেডি সি++ এর ক্লাস নামের জিনিসটা মোয়া বানিয়ে ফেলেছে।

সি একটা ফাটাফাটি ল্যাঙ্গুয়েজ আর সি++ এর কথা তো বলাই বাহুল্য। যারা অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর সুবাতাস পেয়েছেন তারা এটা আরো ভাল করে জানেন। আমরা জানি, সি++ এ কিছু প্রিমিটিভ ডাটা টাইপ ডিফাইন করা আছে, যাদের উপরে আমরা খুব সহজেই বিভিন্ন অপারেশন চালাতে পারি, কিন্তু মাঝে মাঝে এই রকমের ডাটা টাইপের ব্যবহার আমাদের কে ঝামেলায় ফেলতে পারে। যেমন, মনে করা যাক আমাকে একটা 2D গ্রিডের কিছু পয়েন্ট স্টোর করতে হবে। শুধু int টাইপের অ্যারে দিয়ে এটা মেইনটেইন করাটা বেশ মুশকিলের ব্যাপার। কিন্তু যদি এরকম একটা ডাটা টাইপ থাকতো 'point' নামে, যা কিনা (x, y) আকারে কো-অর্ডিনেট রাখতে পারতো!!! সি এ সেই ব্যাবস্থা করেই দেয়া আছে, যেন প্রোগ্রামাররা চাইলেই ইচ্ছা মতো ডাটা টাইপ

বানিয়ে নিতে পারেন, আর সি++ এটাকে আরেক ডিগ্রি এগিয়ে নিয়ে গেছে। এখানে প্রোগ্রামার চাইলে তার বানানো ডাটা টাইপের আচার আচরণও বলে দিতে পারেন।

কিন্তু, প্রশ্ন হল, এটা কন্টেস্ট প্রোগ্রামিং এর জন্য কতটা সুইটেবল?

পেয়ার কি?

কন্টেস্ট প্রোগ্রামিং-এ সাধারনতঃ খুব কমপ্লেক্স ডাটা টাইপ বানাতে হয় না। সেখানে বেশি প্রয়োজন খুব দ্রুত আর নির্ভুল কোডিং। যেমন, প্রায়ই দেখা যায়, একটা গ্রাফের এজ গুলো স্টোর করতে হবে, বা জিয়োমেট্রির প্রবলেমের জন্য কো-অর্ডিনেট নিয়ে কাজ করতে হবে, কখনো বা দেখা যায় কিছু স্ট্রিং-এর জন্য কিছু নাম্বার দিতে হবে, অথবা একগাদা ডাটা বিভিন্ন ক্রাইটেরিয়ার ভিত্তিতে সর্ট বা সার্চ করতে হবে। সাধরনতঃ এসব ক্ষেত্রে প্রোগ্রামার যদি নিজে থেকে ডাটাটাইপ বানাতে যায়, কোন সন্দেহ নাই তাতে তার মূল্যবান কিছু সময় নষ্ট হবে। যেমন, নিচের প্রবলেমটা দেখিঃ

আমাকে একগাদা 2D পয়েন্ট থাকবে, আমাকে সেগুলা সর্ট করতে হবে। এখন, আমি চাইলেই একটা স্ট্রাকচার বানাতে পারিঃ

?

```
1 struct point { int x, y; } P[128];
```

অর্থাৎ, P[] হল একটা পয়েন্ট টাইপের অ্যারে, এখন এটাকে সর্ট করতে হবে। কাজ তো সোজাই, অ্যালগোরিদমের হেডারটা ইনক্লুড করে sort() মেরে দিলেই হয়... কিন্তু আসলে এটা এত সিম্পল না, কারন, sort() একটা টেমপ্লেট ফাংশন, মানে যে কোন ডাটাটাইপের জন্য কাজ করবে, কিন্তু তার আগে তাকে ডাটাটাইপের ভাবগতি জানাতে হবে। আমি 'struct point' দিয়ে কি বুঝাতে চাই, এটা তার বুঝার কোন কারনই নাই যদি না আমি বলে দেই। তার মানে খালি sort() কে ডাকলেই চলবে না, তার সাথে অপারেটর ওভারলোড বা কম্পেয়ার ফাংশন লিখে বুঝিয়ে দিতে হবে যে আমি আসলে কি চাই। আর এখানেই std::pair এর প্লাস পয়েন্ট।

std::pair জিনিশটা তেমন কিছুই না, জাস্ট দুইটা ভ্যালু কে একসাথে করে রাখে, যেখানে ভ্যালু দুইটা যে কোন টাইপের হতে পারে, ডিফারেন্ট টাইপেরও হতে পারে। পেয়ার ক্লাসটা ডিফাইন করা থাকে std <utility> নামের হেডারে।

?

```
1 #include <utility>
2 using namespace std;
```

pair ক্লাসের ডেফিনিশনঃ

?

```
1 template <class T1, class T2> struct pair {
2    T1 first;
3    T2 second;
4    pair(): first(T1()), second(T2()) {}
5    pair(const T1 &x, const T2 &y): first(x), second(y) {}
6    template <class U, class V> pair(const pair<U, V> &p): first(p.first), second(p.second) {}
7 };
```

যাদের টেমপ্লেট সম্পর্কে আইডিয়া নাই, তাদের জন্য অল্প কথায়, টেমপ্লেট একটা সিস্টেম যেখানে কোন অ্যাকচুয়াল টাইপের জন্য ডিজাইন থাকে না, বরং যে কোন টাইপের জন্য তার একটা ইন্সট্যান্স তৈরি করা যায়। টেমপ্লেট শিখতে হলে এই পেজটি দেখা যেতে পারে।

pair এর সুবিধা হল, এটা টেমপ্লেট টাইপ, তাই STL algorithm এর ফাংশন গুলার জন্য সাধারনতঃ pair অবজেক্ট গুলার আলাদা করে কোন পরিচয় দেওয়ার দরকার পড়ে না। যেমন আগের প্রবলেমে জাস্ট sort() কল দিলেই চলে, সে অটমেটিক প্রথমে পেয়ারের প্রথম মেম্বার, তার পর ২য় টা, তার পর ৩য় টা, এভাবে বাকি গুলা কম্পেয়ার করে দেখবে। প্রোগ্রামারকে এর জন্য কিছুই বলতে হবে না।

কিভাবে ব্যবহার করে?

pair নিয়ে কাজ করতে চাইলে <utility> হেডার ইনক্লুড করা উচিৎ, অবশ্য যে কোন STL হেডার ইনক্লুড করলেই pair ব্যবহারের সুবিধা পাওয়া যায়। আর pair টাইপের অবজেক্ট সহজে ক্রিয়েট করার জন্য <utility> হেডারের make_pair() ফাংশন ব্যবহার করা যায়। আর pair-এর ১ম আর ২য় এলিমেন্টকে যথাক্রমে .first আর .second মেম্বার দিয়ে অ্যাক্সেস করা যায়। নিচে একটা উদাহরন দেখানো হলঃ

?

```
01 #include <iostream>
02 #include <string>
03 #include <utility>
04 using namespace std;
05
06 int main() {
```

```
07    // simple constructions
08    pair< int, int > px, py;
09    pair< int, int > p1(23, 43);
10    pair< int, int > p2 = pair< int, int >(234, 534);
11    px = p1;
12    py.first = p2.first * px.second, py.second = p2.second * px.first;
13    cout << "py: (" << py.first << ", " << py.second << ")\n";
14
15    // bit more complex
16    pair< pair< int, int >, pair< int, int > > p3;
17    p3 = pair< pair<int, int>, pair< int, int > > (px, py);
18    cout << "p3: ((";
19    cout << p3.first.first << ", " << p3.first.second << "), (";
20    cout << p3.second.first << ", " << p3.second.second << "))\n";
21
22    // using make_pair()
23    pair< double, pair< string, int > > p4;
24    p4 = make_pair(3.14159, make_pair("pi", 5) );
25    cout << "this is " << p4.second.first << ", value: " << p4.first;
26    cout << " precision: " << p4.second.second << " digits\n";
27    return 0;
28 }
```

pair নিয়ে সবচেয়ে বেশি যে কথাটা শোনা যায় তা হল, মানুষজন নাকি .first.second.second.first.... এরকম চেইন লিখতে লিখতে টায়ার্ড হয়ে যায়। কিন্তু একটু কাজ করেই এইটাকে সহজ করে ফেলা যায়, যেমন, নিচের কোডে pair কে #define করে নেওয়া হয়েছেঃ

?

```
01 #include <iostream>
02 using namespace std;
03
04 #define pii pair< int, int >
05 #define ppi pair< pii, int >
06 #define ff first
07 #define ss second
08
09 int main() {
10     ppi p1;
11     pii p2;
12     cin >> p2.ff >> p2.ss;
13     p1 = ppi( p2, p2.ss * p2.ff );
14     cout << "entry: " << p1.ff.ff << ", " << p1.ff.ss << endl;
15     cout << "product: " << p1.ss << endl;
16     return 0;
17 }
```

অন্যান্য STL কন্টেইনারের ডাটা টাইপ হিসাবেও pair ব্যবহার করা যায়। যেমন, বি.এফ.এস. অ্যালগরিদমে প্রায়ই কিউতে একজোড়া নাম্বার রাখতে হতে পারে, অথবা ডায়াকস্ট্রার অ্যালগরিদমে পুরা একটা এজকে প্রাইওরিটি কিউতে রাখার ব্যাবস্থা করতে হয়। এটাও খুব সহজেই করা যায়ঃ

?

```
1 #include <queue>
2 using namespace std;
3
4 #define pii pair< int, int >
5 #define edge pair< int, pii >
6 // edge.first is weight, edge.second is a pair indicating endpoints
```

```
7 queue< pii > Q;
8 priority_queue< edge, vector< edge >, greater< edge > > PQ;
```

সতর্কতাঃ

অন্যান্য সব STL অবজেক্টের মত, এখানেও একটা ব্যাপার খেয়াল করতে হবে তা হলঃ উপরের উদাহরন গুলাতে দেখা যায় কন্সট্রাকশন শেষ করার সময় মাঝে মাঝে পর পর দুইটা > ব্যবহার করতে হয়, (যেমন প্রাইওরিটি কিউ এর উদাহরনে শেষে .....greater< edge > >, এখানে শেষ ২টা > এর মাঝখানে একটা স্পেস ব্যবহার করা হয়েছে, এটা কিন্তু সৌন্দর্য বর্ধনের জন্য না। এটা না দিলে অনেক সময় কম্পাইলারের মাথা গরম হয়ে যায়। কারন সি++ এ >> আরো কয়েকটা অপারেটরের কাজ করে। আর যেসব যায়গায় দেখা যায় pair ইউজ করলে আরো ঝামেলা হয়, সেখানে নরমাল স্ট্রাকচার ব্যবহার করাটাই বেশি ভাল। এটা জেনে রাখা ভাল, আর জানলেই যে বসাতে হবে এমনও কোন কথা নাই।

ব্যবহারঃ

সাধারনতঃ STL এর টেমপ্লেট ক্লাসগুলোর ওভারলোডিং সুবিধা নেওয়ার জন্য এবং ছোটো খাটো স্ট্রাকচারের শর্টহ্যান্ড হিসাবে pair ব্যবহার করা হয়ে থাকে। এছাড়া std::map এ হাশিং এর সময় pair ব্যবহার করা হয়। প্রোগ্রামিং প্রবলেমগুলাতে গ্রাফ আর জিওমেট্রিক রিপ্রেজেন্টেশনে pair অনেক ব্যবহার করা হয়।
আশা করি যারা আগ্রহি হবে তারা এটাকে নিয়ে আরো কিছুক্ষন ঘাটাঘাটি করবে, কারন ঘাটাঘাটি খোচাখুচি করাই প্রোগ্রামিং শেখার সবচেয়ে ভাল টেকনিক।
বিস্তারিতঃ http://www.cplusplus.com/reference/std/utility/pair/
Categories: ডাটা স্ট্রাকচার, প্রোগ্রামিং ট্যাগসমূহ :c++, data structure, pair, programming, stl

C++ STL :: priority_queue

প্রায়োরিটি কিউঃ

সব ডাটা-স্ট্রাকচারই যে ধোয়া তুলসি পাতা, এইটা বলা যায় না, বিশেষতঃ যখন “জোর যার মুল্লুক তার” টাইপের এই ডাটা-স্ট্রাকচারটা প্রায়ই ব্যবহার করতে হয়, C++ STL এর priority_queue, এটা একটা বাইনারি হিপ ডাটা-স্ট্রাকচার (ডিফল্টঃ ম্যাক্স হিপ)। সহজ বাংলায়, হিপ হল এমন একটা ট্রি ডাটা-স্ট্রাকচার যেখানে সবচেয়ে বড় (ম্যাক্স হিপ) বা সবচেয়ে ছোট (মিন হিপ) এলিমেন্টটা সবসময় রুটে থাকবে। তাই, যদি এমন একটা ঘটনা ঘটে যে, একদল লোক লাইন দিছে কাঙ্গালি ভোজে, এমন টাইমে মাস্তান টাইপের এক ফকির আসছে, আর লোকজন ভয় পেয়ে তাকে লাইনের সামনে দাঁড়া করায় দিবে, তখন প্রায়রিটি কিউ ছাড়া উপায় নাই, অর্থাৎ খালি আগে আসলেই হবে না, যথেষ্ট পরিমানে ভাব নিয়ে আসতে হবে।
এইটা খুব সহজেই করা যায়, বার বার চেক করে যে নেক্সট কার প্রায়রিটি বেশি, কিন্তু এইভাবে করাটা আন-এফিসিয়েন্ট, তার চেয়ে হিপের মত একটা ট্রি ডাটা-স্ট্রাকচার ব্যবহার করে অনেক কম সময়ে এই কাজ করা

যায়, আর সেই কাজটাকে আর সহজ করে দেওয়ার জন্যই আছে priority_queue, queue এর মত এটাও একটা অ্যাডাপ্টার ক্লাস, তার মানে হল, যে সব STL কন্টেইনার front(), push_back(), pop_back() এই অ্যাকসেস দেয়, তাদেরকে priority_queue এর ইন্টারনাল কন্টেইনার হিসাবে ব্যবহার করা যাবে। আর এটা ব্যবহার করতে চাইলে std <queue> হেডারটা প্রোগ্রামে ইনক্লুড করতে হবেঃ

?

```
1 #include <queue>
2 using namespace std;
```

কনস্ট্রাকশনঃ

priority_queue বেশ কয়েকভাবে বানানো যায়, বাই ডিফল্ট এটা ম্যাক্স হিপ ইম্পলিমেন্ট করে আর এলিমেন্ট কম্পেয়ার (ছোট-বড় বুঝার জন্য) করার জন্য <queue> এর less<type_t> ক্লাস ব্যাবহার করে। চাইলে এটাকে অন্য কোন কন্টেইনার যেমন vector কে ইন্টারনাল কন্টেইনার হিসাবে ডিফাইন করে দেওয়া যায়, আর বিল্ট ইন বা ওভারলোডেড টেমপ্লেট টাইপ ছাড়াও এটা বানানো যায়, সেক্ষেত্রে ডাটার নিজের ডিফল্ট কম্পেয়ারিজন ক্লাস ব্যবহার করতে হবেঃ < কে ওভারলোড করে, অথবা এক্সপ্লিসিট কম্পেয়ারিজন ক্লাসে () কে ওভারলোড করে। আর, মিন হিপ বানানোও সহজ, <queue> এ ডিফাইন করা greater<type_t> ক্লাস ব্যবহার করে খুব সহজেই করা যায়। যেমনঃ

?

```
01 // constructing priority queues
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 class mycomparison {
07     bool reverse;
08 public:
09     mycomparison(const bool& revparam=false) { reverse = revparam; }
10     bool operator() (const int& lhs, const int&rhs) const {
11         if (reverse) return (lhs>rhs);
```

```
12        else return (lhs<rhs);
13    }
14 };
15
16 int main () {
17    int myints[]= {10,60,50,20};
18
19    // default construction
20    priority_queue<int> first;
21    priority_queue<int> second (myints,myints+3);
22
23    // using greater<> to create min heap
24    priority_queue< int, vector<int>, greater<int> > third (myints,myints+3);
25
26    // using "mycomparison" comparison class
27    priority_queue< int, vector<int>, mycomparison > fourth;
28
29    typedef priority_queue<int,vector<int>,mycomparison> mypq_type;
30    mypq_type fifth (mycomparison());
31    mypq_type sixth (mycomparison(true));
32
33    return 0;
34 }
```

কমপ্লেক্সিটিঃ ইনিশিয়াল এলিমেন্টের সাপেক্ষে লিনিয়ার, ইনিশিয়ার এলিমেন্ট অ্যাসাইন না করলে কন্সট্যান্ট।

পুশ, পপ, টপঃ

priority_queue এর এলিমেন্ট কে অ্যাকসেস করার জন্য ৩ টা ফাংশন ডিফাইন করা আছে, push(), pop() আর top(). কিউতে কোন এলিমেন্ট অ্যাড করতে চাইলে push() ফাংশনটা ব্যবহার করতে হয়, top() হিপের

বর্তমান রুটকে রিটার্ন করে, আর pop() সেটা ট্রি থেকে ডিলিট করে। অর্থাৎ top() আর pop() একত্রে ম্যাক্স হিপের জন্য extract_max() আর মিন হিপের জন্য extract_min() [see: Introduction To Algorithms -- CLRS -- MIT Press] এর কাজ করে। নিচে উদাহরণ দেওয়া হলঃ

?

```
01 // priority_queue::push/pop/top
02 #include <iostream>
03 #include <queue>
04 #include <ctime>
05 #include <cstdlib>
06 using namespace std;
07
08 int main () {
09   priority_queue<int> mypq;
10   srand(time(NULL));
11
12   cout << "Pushing some random values...\n";
13   for(int n, i = 0; i < 10; i++) {
14     n = rand();
15     cout << " " << n;
16     mypq.push(n);
17   }
18   cout << endl;
19
20   cout << "Popping out elements...\n";
21   while (!mypq.empty()) {
22     cout << " " << mypq.top();
23     mypq.pop();
```

```
24    }
25    cout << endl;
26    return 0;
27 }
```

কমপ্লেক্সিটিঃ push() লগারিদমিক, top() কন্সট্যান্ট, কিন্তু pop() লগারিদমিক।

কিউ এর সাইজ চেক করাঃ

priority_queue এর বর্তমান এলিমেন্ট কতগুলা আছে বা, কিউ খালি কিনা এটা টেস্ট করার জন্য ২ টা ফাংশন ডিফাইন করা আছে, size() আর empty(). কিউ থেকে top() আর pop() ব্যবহার করার আগে অবশ্যই কিউ খালি কিনা চেক করা উচিৎ, তা না হলে রান টাইম এরর জেনারেট হতে পারে। আর, কিউ খালি কিনা এটা empty() মেথড দিয়েই চেক করা উচিত, soze()==0 দিয়ে নয়। STL এর সব কন্টেইনার ক্লাসেই এদের ব্যবহার একি রকমঃ

?

```
01 // priority_queue ::size/empty
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 int main () {
07    priority_queue<int> myints;
08    cout << "0. size: " << (int) myints.size() << endl;
09
10    for (int i=0; i<10; i++) myints.push(i);
11    cout << "1. size: " << (int) myints.size() << endl;
12
13    myints.pop();
14    cout << "2. size: " << (int) myints.size() << endl;
15
```

```
16    while(!myints.empty()) myints.pop();
17    cout << "3. size: " << (int) myints.size() << endl;
18
19    return 0;
20 }
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ব্যবহারঃ

বাইনারি হিপ তৈরিতে, ডায়াকস্ট্রা আর প্রিম অ্যালগরিদমে ব্যবহার করা হয়, টেমপ্লেট ক্লাস হওয়ায় যে কোন ডাটা টাইপের জন্য খুব দ্রুত কোড ইমপ্লিমেন্ট করা যায়।

বিস্তারিতঃ http://www.cplusplus.com/reference/stl/priority_queue/

Categories: ডাটা স্ট্রাকচার, প্রোগ্রামিং ট্যাগসমূহ :c++, data structure, priority queue, programming, stl

C++ STL :: queue

কিউঃ

আগে আসলে আগে পাবেন, এই রকমের একটা ডাটা স্ট্রাকচার হল কিউ, যাকে আমরা বলি ফার্স্ট ইন ফার্স্ট আউট (FIFO)। এটা C++ STL এর সবচেয়ে বেশি ব্যবহৃত কন্টেইনার ক্লাস। queue এর সামনের দিক থেকে ডাটা এক্সট্র্যাক্ট করা হয়, আর ইনসার্ট করা হয় তার বিপরীত দিক থেকে। তাই, যে সব STL কন্টেইনার push_back() আর pop_front() সাপোর্ট করে সেগুলো দিয়ে কিউ ইমপ্লিমেন্ট করা যায়, যেমন list আর deque. অবশ্য queue এর ডিফল্ট কন্টেইনার হল deque, যদি কিছু বলে না দেয়া হয়। stack এর মত queue ও একটা অ্যাডাপ্টার ক্লাস।

queue ব্যবহার করতে চাইলে std <queue> হেডারটা প্রোগ্রামে ইনক্লুড করতে হবেঃ

?

```
1 #include <queue>
2 using namespace std;
```

কন্সট্রাকশনঃ

অন্যান্য কন্টেইনার ক্লাসের মত queue এর সাধারণ কন্সট্রাক্টর queue< type_t > myqueue; এই ধরনের। এছাড়া এক্সপ্লিসিট কন্টেইনার ডিক্লেয়ার করেও queue কন্সট্রাক্ট করা যায়, যেমনঃ

?

```
01 // constructing queues
02 #include <deque>
03 #include <list>
04 #include <queue>
05 using namespace std;
06
07 int main ()
08 {
09   deque< int > mydeck(3,100); // deque with 3 elements
10   list< int > mylist(2,200); // list with 2 elements
11
12   // implicit declaration
13   queue< int > first; // empty queue
14   queue< int > second(mydeck); // from a mydeck
15   queue< int > third(second); // from another queue
16
17   // explicit declaration
18   queue< int, list< int > > fourth; // empty queue
19   queue< int, list< int > > fifth(mylist); // from mylist
20   queue< int, deque< int > > sixth; // empty queue
21   queue< int, deque< int > > seventh(mydeck); // from mydeck
22
23   // initialization with operator=
24   queue< int > eighth = first; // initialized with first
25
```

```
26    return 0;
27 }
```

কমপ্লেক্সিটিঃ কন্সট্রাক্টরের সাপেক্ষে কন্সট্যান্ট, অর্থাৎ কন্টেইনারের উপরে নির্ভর করে।

পুশ আর পপঃ

queue এ ডাটা ইন্সার্ট আর এক্সট্র্যাক্ট করার জন্য ২ টা মেম্বার ফাংশন আছে, push() আর pop()। push() এর কাজ হল queue এর শেষে এলিমেন্ট ইন্সার্ট করা আর pop() এর সাহায্যে queue এর সামনে থেকে কোন এলিমেন্ট বের করে দেয়া। যেমনঃ

?

```
01 //
   queue::push/pop
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 int main()
07 {
08    queue< int > Q;
09
10    // push 5 integers
11    for(int i=1; i<=5; i++) Q.push(i);
12
13    // pop 5 integers
14    // will be popped in the same order they were pushed
15    for( ; !Q.empty() ; )
16    {
17       cout << Q.front() << endl;
18       Q.pop();
```

```
19    }
20
21    return 0;
22 }
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ফ্রন্ট আর ব্যাক

queue তে এলিমেন্ট এক্সেসের জন্য ২ টা ফাংশন হল front() আর back(); front() দিয়ে queue এর ফার্স্ট এলিমেন্ট এক্সেস করা যায়, আর back() দিয়ে লাস্ট ইন্সার্ট করা এলিমেন্ট কে পাওয়া যায়। front() আর back() এর এলিমেন্ট কে স্ট্যান্ডার্ড অপারেটর গুলর সাহায্যে মডিফাই করা যায়। যেমনঃ

?

```
01 // queue::front/back
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 int main ()
07 {
08    queue<int> myqueue;
09
10    myqueue.push(77);
11    myqueue.push(16);
12    cout << "myqueue.front() = " << myqueue.front() << endl;
13    cout << "myqueue.back() = " << myqueue.back() << endl;
14
15    // modify front element
16    myqueue.front() -= myqueue.back();
```

```
17    cout << "myqueue.front() is now " << myqueue.front() << endl;
18
19    // modify back element
20    myqueue.back() += myqueue.front();
21    cout << "myqueue.back() is now " << myqueue.back() << endl;
22
23    return 0;
24 }
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

কিউ কি খালি?

queue এ এই মুহূর্তে কত গুলা এলিমেন্ট আছে সেটা জানা যায় size() ফাংশনের মাধ্যমে, আর empty() ফাংশন টা boolean, queue খালি থাকলে true দেয়, না হলে false; queue খালি কি না, সেটা চেক করা হয় empty() দিয়ে, কখনই size() এর ভ্যালু ০ কিনা এটা দেখে queue খালি কিনা, সেই টেস্ট করা উচিৎ না, আর queue তে pop() করার আগে আবশ্যই দেখে নিতে হবে queue এ কোন এলিমেন্ট আছে কিনা, তা না হলে run-time error হতে পারে। নিচের কোডে size() আর empty() এর প্রয়োগ দেখানো হলঃ

?

```
01 // queue::size/empty
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 int main ()
07 {
08    queue<int> Q;
09
10    // just push some elements
11    for(int i = 0; i < 5; i++) Q.push(i*i);
```

```
12
13    cout << "Queue has " << Q.size() << " elements" << endl;
14
15    Q.pop(); // not a good way, first check for Q.empty()
16
17    if(!Q.empty()) Q.pop(); // this is the porper way
18
19    while(!Q.empty()) Q.pop(); // pop all element
20
21    if(Q.size()==0) cout << "Q is empty" << endl; // not a good way
22    if(Q.empty()) cout << "Q is empty" << endl; // this is the proper way
23
24    return 0;
25 }
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ব্যবহারঃ

FIFO ডাটা স্ট্রাকচার হিসাবে, BFS ট্রাভার্সাল, বিভিন্ন গ্রাফ এলগরিদমে queue ইম্পলিমেন্ট করা হয়।

বিস্তারিতঃ http://www.cplusplus.com/reference/stl/queue/

C++ STL :: stack

স্ট্যাকঃ

STL কন্টেইনারদের মধ্যে সম্ভবত সবচেয়ে সিম্পল ডাটা স্ট্রাকচার হল stack, এটা একটা লাস্ট ইন ফার্স্ট আউট (LIFO) ডাটা স্ট্রাকচার, মানে হল যে সবার শেষে আসবে, সে সবার আগে ভাগবে... সোজা কথায় এই কন্টেইনারের শুধুমাত্র একটা দিকেই ডাটা ইন্সার্ট বা এক্সট্র্যাক্ট করা হয়। আর STL এ stack তার ডিফল্ট ইন্টার্নাল ডাটা স্ট্রাকচার হিসাবে ব্যবহার করে STL এরই deque কন্টেইনার, তবে চাইলে vector বা list ও ব্যবহার করা যেতে পারে। যে সব কন্টেইনার push_back() আর pop_back() মেথড ২ টা সাপোর্ট করে সেগুলোকেই stack এর কন্টেইনার ক্লাস হিসাবে ব্যবহার করা যায়। stack আসলে একটা অ্যাডাপ্টার ক্লাস, অর্থাৎ, এটা তৈরি করা হয় এর ইন্টারনাল কন্টেইনারের স্পেসিফিক কিছু ফাংশনকে এলিমেন্ট একসেসের অনুমতি দিয়ে।

stack ব্যবহার করতে চাইলে সর্বপ্রথম কাজটা হল std <stack> হেডারটা প্রোগ্রামে ইনক্লুড করাঃ

?

```
1 #include <stack>
2 using namespace std;
```

কন্সট্রাক্টরঃ

অন্যান্য STL কন্টেইনার ক্লাসের মত stack এরও সাধারণ কন্সট্রাক্টর stack< type_t > myStack; এই রকমের, তবে আরো অনেকভাবে ডিক্লেয়ার করা যায়। যেমনঃ

?

```
01 // constructing stacks
02 #include <list>
03 #include <vector>
04 #include
   <deque>
05 #include <stack>
06 using namespace std;
07
08 int main ()
09 {
10   // using default container deque
11
12   stack< int > first; // empty stack
13   deque< int > mydeque(3, 100); // deque with 3 elements
14   stack< int > second(mydeque); // from mydeque
15   stack< int > third(second); // from another stack second
16
17   // explicit container declarations
```

```
18
19    stack< int, deque< int > > fourth; // empty stack using deque
20    deque< int > newdeque(10, 100); // deque with 10 elements
21    stack< int, deque< int > > fifth(newdeque); // from newdeque
22
23    stack< int, vector< int > > sixth;  // empty stack using vector
24    vector< int > myvector(2, 200); // vector with 2 elements
25    stack< int, vector< int > > seventh(myvector); // from myvector
26
27    stack< int, list< int > > eighth; // empty srack using list
28    list< int > mylist(4, 100); // list with 4 elements
29    stack< int, list< int > > ninth(mylist); // from mylist
30
31    // can refer to some other stack
32    stack< int > tenth = first; // declaration time initialization
33
34    return 0;
35 }
```

কমপ্লেক্সিটিঃ কন্টেইনার কন্সট্রাকশনের সাপেক্ষে কন্সট্যান্ট, অতএব কি ধরণের কন্টেইনার ব্যবহার করা হচ্ছে তার উপরে নির্ভর করে।

এলিমেন্ট একসেসঃ

stack ক্লাসের এলিমেন্ট গুলাকে একসেস করার জন্য ৩ টা মেম্বার ফাংশন আছেঃ

1. top()
2. push()
3. pop()

push() ফাংশনটার কাজ stack এর শেষে কোন এলিমেন্ট ইন্সার্ট করা, আর pop() দিয়ে লাস্ট এলিমেন্ট টা বের করে দেয়া। top() এর সাহায্যে কারেন্টলি stack এ সবার উপরের এলিমেন্ট কে পাওয়া যায়। top() এলিমেন্টকে স্ট্যান্ডার্ড অপারেটরদের সাহায্যে মডিফাই করা যায়।

আর, stack এর এলিমেন্ট কাউন্ট করার জন্য ২ টা ফাংশন আছেঃ

1. size()
2. empty()

size() ব্যবহার করে জানতে পারি এই মুহূর্তে stack এ কতগুলো এলিমেন্ট আছে, আর empty() একটা boolean ফাংশন, stack খালি থাকলে এটা true দেয়, না হলে false, stack এ pop() মেথডটা ব্যবহার করতে চাইলে আগে অবশ্যই চেক করে নিতে হবে stack এ কিছু আছে কিনা, তা না হলে run-time error হতে পারে।

নিচে এই ফাংশন গুলার কাজ দেখানো হলঃ

?

```
01 // stack::push/pop/top/size/empty
02 #include <iostream>
03 #include <stack>
04 using namespace std;
05
06 int main ()
07 {
08   stack< int > mystack;
09   // lets push and pop some values
10   for (int i=0; i<5; i++) mystack.push(i);
11
12   cout << "Popping out elements...";
13   while (!mystack.empty())
14   {
15     cout << " " << mystack.top();
16     mystack.pop();
17   }
18   cout << endl;
```

```
19
20    // changing the value of top
21    mystack.push(100);
22    mystack.top() += 1000;
23    cout << "Now top is: " << mystack.top() << endl;
24    mystack.pop();
25
26    // not a good way to check if stack is empty
27    if(mystack.size() == 0) cout << "Stack is empty" << endl;
28    // better we do this
29    if(mystack.empty()) cout << "Stack is empty" << endl;
30
31    return 0;
32 }
```

কমপ্লেক্সিটিঃ প্রতিটা ফাংশনের কম্পলেক্সিটি কন্সট্যান্ট।

ব্যবহারঃ

সাধারণত এক্সপ্রেশন / গ্রামার প্রসেসিং, রিকারসিভ এলগরিদমের নন রিকারসিভ প্রয়োগ, DFS ট্রাভার্সাল, LIFO অপারেশনে stack ব্যবহার করা হয়। STL এর stack একটা টেমপ্লেট ক্লাস, তাই যে কোন ডাটা টাইপের জন্য খুব দ্রুত stack ইম্পলিমেন্ট করা যায় STL ব্যবহার করে।

বিস্তারিতঃ http://www.cplusplus.com/reference/stl/stack/

C++ STL :: queue

কিউঃ

আগে আসলে আগে পাবেন, এই রকমের একটা ডাটা স্ট্রাকচার হল কিউ, যাকে আমরা বলি ফার্স্ট ইন ফার্স্ট আউট (FIFO)। এটা C++ STL এর সবচেয়ে বেশি ব্যবহৃত কন্টেইনার ক্লাস। queue এর সামনের দিক থেকে ডাটা এক্সট্র্যাক্ট করা হয়, আর ইনসার্ট করা হয় তার বিপরীত দিক থেকে। তাই, যে সব STL কন্টেইনার push_back() আর pop_front() সাপোর্ট করে সেগুলো দিয়ে কিউ ইমপ্লিমেন্ট করা যায়, যেমন list আর deque. অবশ্য queue এর ডিফল্ট কন্টেইনার হল deque, যদি কিছু বলে না দেয়া হয়। stack এর মত queue ও একটা অ্যাডাপ্টার ক্লাস।

queue ব্যবহার করতে চাইলে std <queue> হেডারটা প্রোগ্রামে ইনক্লুড করতে হবেঃ

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

কন্সট্রাকশনঃ

অন্যান্য কন্টেইনার ক্লাসের মত queue এর সাধারণ কন্সট্রাক্টর queue< type_t > myqueue; এই ধরনের। এছাড়া এক্সপ্লিসিট কন্টেইনার ডিক্লেয়ার করেও queue কন্সট্রাক্ট করা যায়, যেমনঃ

```
// constructing queues
#include <deque>
#include <list>
#include <queue>
using namespace std;

int main ()
{
        deque< int > mydeck(3,100); // deque with 3 elements
        list< int > mylist(2,200); // list with 2 elements

        // implicit declaration
        queue< int > first; // empty queue
        queue< int > second(mydeck); // from a mydeck
        queue< int > third(second); // from another queue

        // explicit declaration
        queue< int, list< int > > fourth; // empty queue
        queue< int, list< int > > fifth(mylist); // from mylist
        queue< int, deque< int > > sixth; // empty queue
        queue< int, deque< int > > seventh(mydeck); // from mydeck

        // initialization with operator=
        queue< int > eighth = first; // initialized with first
```

```
        return 0;
}
```

কমপ্লেক্সিটিঃ কন্সট্রাক্টরের সাপেক্ষে কন্সট্যান্ট, অর্থাৎ কন্টেইনারের উপরে নির্ভর করে।

পুশ আর পপঃ

queue এ ডাটা ইন্সার্ট আর এক্সট্র্যাক্ট করার জন্য ২ টা মেম্বার ফাংশন আছে, push() আর pop()। push() এর কাজ হল queue এর শেষে এলিমেন্ট ইন্সার্ট করা আর pop() এর সাহায্যে queue এর সামনে থেকে কোন এলিমেন্ট বের করে দেয়া। যেমনঃ

```
// queue::push/pop
#include <iostream>
#include <queue>
using namespace std;

int main()
{
        queue< int > Q;

        // push 5 integers
        for(int i=1; i<=5; i++) Q.push(i);

        // pop 5 integers
        // will be popped in the same order they were pushed
        for( ; !Q.empty() ; )
        {
                cout << Q.front() << endl;
                Q.pop();
        }

        return 0;
}
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ফ্রন্ট আর ব্যাক

queue তে এলিমেন্ট এক্সেসের জন্য ২ টা ফাংশন হল front() আর back(); front() দিয়ে queue এর ফার্স্ট এলিমেন্ট এক্সেস করা যায়, আর back() দিয়ে লাস্ট ইন্সার্ট করা এলিমেন্ট কে পাওয়া যায়। front() আর back() এর এলিমেন্ট কে স্ট্যান্ডার্ড অপারেটর গুলর সাহায্যে মডিফাই করা যায়। যেমনঃ

```
// queue::front/back
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
	queue<int> myqueue;

	myqueue.push(77);
	myqueue.push(16);
	cout << "myqueue.front() = " << myqueue.front() << endl;
	cout << "myqueue.back() = " << myqueue.back() << endl;

	// modify front element
	myqueue.front() -= myqueue.back();
	cout << "myqueue.front() is now " << myqueue.front() << endl;

	// modify back element
	myqueue.back() += myqueue.front();
	cout << "myqueue.back() is now " << myqueue.back() << endl;

	return 0;
}
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

কিউ কি খালি?

queue এ এই মুহূর্তে কত গুলা এলিমেন্ট আছে সেটা জানা যায় size() ফাংশনের মাধ্যমে, আর empty() ফাংশন টা boolean, queue খালি থাকলে true দেয়, না হলে false; queue খালি কি না, সেটা চেক করা হয় empty() দিয়ে, কখনই size() এর ভ্যালু ০ কিনা এটা দেখে queue খালি কিনা, সেই টেস্ট করা উচিৎ না, আর queue তে pop() করার আগে আবশ্যই দেখে নিতে হবে queue এ কোন এলিমেন্ট আছে কিনা, তা না হলে run-time error হতে পারে। নিচের কোডে size() আর empty() এর প্রয়োগ দেখানো হলঃ

```
// queue::size/empty
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
	queue<int> Q;

	// just push some elements
	for(int i = 0; i < 5; i++) Q.push(i*i);

	cout << "Queue has " << Q.size() << " elements" << endl;

	Q.pop(); // not a good way, first check for Q.empty()

	if(!Q.empty()) Q.pop(); // this is the porper way

	while(!Q.empty()) Q.pop(); // pop all element

	if(Q.size()==0) cout << "Q is empty" << endl; // not a good way
	if(Q.empty()) cout << "Q is empty" << endl; // this is the proper way

	return 0;
}
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ব্যবহারঃ

FIFO ডাটা স্ট্রাকচার হিসাবে, BFS ট্রাভার্সাল, বিভিন্ন গ্রাফ এলগরিদমে queue ইম্পলিমেন্ট করা হয়।

বিস্তারিতঃ http://www.cplusplus.com/reference/stl/queue/

C++ STL :: priority_queue

প্রায়োরিটি কিউঃ

সব ডাটা-স্ট্রাকচারই যে ধোয়া তুলসি পাতা, এইটা বলা যায় না, বিশেষতঃ যখন "জোর যার মুল্লুক তার" টাইপের এই ডাটা-স্ট্রাকচারটা প্রায়ই ব্যবহার করতে হয়, C++ STL এর priority_queue, এটা একটা বাইনারি হিপ ডাটা-স্ট্রাকচার (ডিফল্টঃ ম্যাক্স হিপ)। সহজ বাংলায়, হিপ হল এমন একটা ট্রি ডাটা-স্ট্রাকচার যেখানে সবচেয়ে বড় (ম্যাক্স হিপ) বা সবচেয়ে ছোট (মিন হিপ) এলিমেন্টটা সবসময় রুটে থাকবে। তাই, যদি এমন একটা ঘটনা ঘটে যে, একদল লোক লাইন দিছে কাঙ্গালি ভোজে, এমন টাইমে মাস্তান টাইপের এক ফকির আসছে, আর লোকজন ভয় পেয়ে তাকে লাইনের সামনে দাঁড়া করায় দিবে, তখন প্রায়রিটি কিউ ছাড়া উপায় নাই, অর্থাৎ খালি আগে আসলেই হবে না, যথেষ্ট পরিমানে ভাব নিয়ে আসতে হবে।
এইটা খুব সহজেই করা যায়, বার বার চেক করে যে নেক্সট কার প্রায়রিটি বেশি, কিন্তু এইভাবে করাটা আন-এফিসিয়েন্ট, তার চেয়ে হিপের মত একটা ট্রি ডাটা-স্ট্রাকচার ব্যবহার করে অনেক কম সময়ে এই কাজ করা যায়, আর সেই কাজটাকে আর সহজ করে দেওয়ার জন্যই আছে priority_queue, queue এর মত এটাও একটা অ্যাডাপ্টার ক্লাস, তার মানে হল, যে সব STL কন্টেইনার front(), push_back(), pop_back() এই অ্যাকসেস দেয়, তাদেরকে priority_queue এর ইন্টারনাল কন্টেইনার হিসাবে ব্যবহার করা যাবে। আর এটা ব্যবহার করতে চাইলে std <queue> হেডারটা প্রোগ্রামে ইনক্লুড করতে হবেঃ

?

```
1 #include <queue>
2 using namespace std;
```

কনস্ট্রাকশনঃ

priority_queue বেশ কয়েকভাবে বানানো যায়, বাই ডিফল্ট এটা ম্যাক্স হিপ ইম্পলিমেন্ট করে আর এলিমেন্ট কম্পেয়ার (ছোট-বড় বুঝার জন্য) করার জন্য <queue> এর less<type_t> ক্লাস ব্যাবহার করে। চাইলে এটাকে অন্য কোন কন্টেইনার যেমন vector কে ইন্টারনাল কন্টেইনার হিসাবে ডিফাইন করে দেওয়া যায়, আর বিল্ট ইন বা ওভারলোডেড টেমপ্লেট টাইপ ছাড়াও এটা বানানো যায়, সেক্ষেত্রে ডাটার নিজের ডিফল্ট কম্পেয়ারিজন ক্লাস ব্যবহার করতে হবেঃ < কে ওভারলোড করে, অথবা এক্সপ্লিসিট কম্পেয়ারিজন ক্লাসে () কে ওভারলোড

করে। আর, মিন হিপ বানানোও সহজ, <queue> এ ডিফাইন করা greater<type_t> ক্লাস ব্যবহার করে খুব সহজেই করা যায়। যেমনঃ

?

```
01 // constructing priority queues
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 class mycomparison {
07    bool reverse;
08 public:
09    mycomparison(const bool& revparam=false) { reverse = revparam; }
10    bool operator() (const int& lhs, const int&rhs) const {
11       if (reverse) return (lhs>rhs);
12       else return (lhs<rhs);
13    }
14 };
15
16 int main () {
17    int myints[]= {10,60,50,20};
18
19    // default construction
20    priority_queue<int> first;
21    priority_queue<int> second (myints,myints+3);
22
23    // using greater<> to create min heap
```

```
24    priority_queue< int, vector<int>, greater<int> > third (myints,myints+3);
25
26    // using "mycomparison" comparison class
27    priority_queue< int, vector<int>, mycomparison > fourth;
28
29    typedef priority_queue<int,vector<int>,mycomparison> mypq_type;
30    mypq_type fifth (mycomparison());
31    mypq_type sixth (mycomparison(true));
32
33    return 0;
34 }
```

কমপ্লেক্সিটিঃ ইনিশিয়াল এলিমেন্টের সাপেক্ষে লিনিয়ার, ইনিশিয়ার এলিমেন্ট অ্যাসাইন না করলে কন্সট্যান্ট।

পুশ, পপ, টপঃ

priority_queue এর এলিমেন্ট কে অ্যাকসেস করার জন্য ৩ টা ফাংশন ডিফাইন করা আছে, push(), pop() আর top(). কিউতে কোন এলিমেন্ট অ্যাড করতে চাইলে push() ফাংশনটা ব্যবহার করতে হয়, top() হিপের বর্তমান রুটকে রিটার্ন করে, আর pop() সেটা ট্রি থেকে ডিলিট করে। অর্থাৎ top() আর pop() একত্রে ম্যাক্স হিপের জন্য extract_max() আর মিন হিপের জন্য extract_min() [see: Introduction To Algorithms -- CLRS -- MIT Press] এর কাজ করে। নিচে উদাহরণ দেওয়া হলঃ

?

```
01 // priority_queue::push/pop/top
02 #include <iostream>
03 #include <queue>
04 #include <ctime>
05 #include <cstdlib>
06 using namespace std;
07
08 int main () {
```

```
09    priority_queue<int> mypq;
10    srand(time(NULL));
11
12    cout << "Pushing some random values...\n";
13    for(int n, i = 0; i < 10; i++) {
14       n = rand();
15       cout << " " << n;
16       mypq.push(n);
17    }
18    cout << endl;
19
20    cout << "Popping out elements...\n";
21    while (!mypq.empty()) {
22       cout << " " << mypq.top();
23       mypq.pop();
24    }
25    cout << endl;
26    return 0;
27 }
```

কমপ্লেক্সিটিঃ push() লগারিদমিক, top() কন্সট্যান্ট, কিন্তু pop() লগারিদমিক।

কিউ এর সাইজ চেক করাঃ

priority_queue এর বর্তমান এলিমেন্ট কতগুলা আছে বা, কিউ খালি কিনা এটা টেস্ট করার জন্য ২ টা ফাংশন ডিফাইন করা আছে, size() আর empty(). কিউ থেকে top() আর pop() ব্যবহার করার আগে অবশ্যই কিউ খালি কিনা চেক করা উচিৎ, তা না হলে রান টাইম এরর জেনারেট হতে পারে। আর, কিউ খালি কিনা এটা empty() মেথড দিয়েই চেক করা উচিত, soze()==0 দিয়ে নয়। STL এর সব কন্টেইনার ক্লাসেই এদের ব্যবহার একি রকমঃ

?

```
01 // priority_queue ::size/empty
02 #include <iostream>
03 #include <queue>
04 using namespace std;
05
06 int main () {
07   priority_queue<int> myints;
08   cout << "0. size: " << (int) myints.size() << endl;
09
10   for (int i=0; i<10; i++) myints.push(i);
11   cout << "1. size: " << (int) myints.size() << endl;
12
13   myints.pop();
14   cout << "2. size: " << (int) myints.size() << endl;
15
16   while(!myints.empty()) myints.pop();
17   cout << "3. size: " << (int) myints.size() << endl;
18
19   return 0;
20 }
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ব্যবহারঃ

বাইনারি হিপ তৈরিতে, ডায়াকস্ট্রা আর প্রিম অ্যালগরিদমে ব্যবহার করা হয়, টেমপ্লেট ক্লাস হওয়ায় যে কোন ডাটা টাইপের জন্য খুব দ্রুত কোড ইমপ্লিমেন্ট করা যায়।

বিস্তারিতঃ http://www.cplusplus.com/reference/stl/priority_queue/

বিভিন্ন অনলাইন জাজ ও কন্টেস্ট সাইট সমূহ

- ☆ ইউ.ভি.এ. অনলাইন জাজ (uva)
- ☆ স্ফেয়ার অনলাইন জাজ (spoj)
- ☆ ইউসাকো ট্রেইনিং প্রোগ্রাম গেটওয়ে (usaco)
- ☆ এ.সি.এম. আই.সি.পি.সি. লাইভ আর্কাইভ (cli)
- ☆ সারাতভ অনলাইন কন্টেস্টার (sgu)
- ☆ কোড ফোর্সেস (cf)
- ☆ টপ কোডার (tc)
- ☆ কোড শেফ (cc)
- ☆ গুগল কোড জ্যাম (gcj)
- ☆ ক্রোয়েশিয়ান ওপেন কন্টেস্ট ইন ইনফরমেটিক্স (coci)
- ☆ ইন্টারনেট প্রবলেম সলভিং কন্টেস্ট (ipsc)
- ☆ তিয়ানজিন ইউনিভার্সিটি অনলাইন জাজ (tju)
- ☆ সি.এস.ই. ডি.ইউ. অনলাইন জাজ (cseduoj)
- ☆ প্রোজেক্ট ইউলার (pe)

ফোরাম গুলো

- ☆ ইউ.ভি.এ. ইলেকট্রনিক বোর্ড
- ☆ স্পজ কমিউনিটি ফোরাম
- ☆ টপ কোডার ফোরাম
- ☆ ইউসাকো ফোরাম
- ☆ উবুনতু ফোরামস

কিভাবে শুরু করব? কিভাবে প্র্যাক্টিস করব?

- ☆ বিগেনারদের জন্য ফাহিম ভাই
- ☆ কন্টেস্ট টিপস
- ☆ Momtchil এর এডভাইস
- ☆ ইন্সপায়ারিং চিঠি : আব্দুল্লাহ আল মাহমুদ
- ☆ ইন্সপায়ারিং চিঠি : মাহবুবুল হাসান শান্ত
- ☆ ইন্সপায়ারিং চিঠি : শাহরিয়ার মনজুর
- ☆ ইন্সপায়ারিং চিঠি : তামিম শাহরিয়ার সুবিন

Tanbir Ahmad Razib

Mobile No:→ 01738 -359 555

E-Mail: → tanbir.cox@gmail.com

Facebook: → http://facebook.com/tanbir.cox

e-books Page: → http://facebook.com/tanbir.ebooks

Web Site : → http://tanbircox.blogspot.com

I share new interesting & Useful Bangla e-books(pdf) everyday on my facebook page & website .

Keep on eye always on my facebook page & website & update ur knowledge .

If You think my e-books are useful , then please share & Distribute my e-book on Your facebook & personal blog .