Argument passing techniques

- Pass By Value
- Pass By Pointer/Address
- Pass By Reference

Call By Value

- It is a default mechanism for argument passing.
- Any changes made in the formal argument are not reflected back to actual argument, rather they remain local to the block which are lost once the control is returned back to calling program

EXAMPLE

```
#include<iostream>
Using namespce std;
void swap(int,int);
int main()
{
int a=10,b=20;
cout<<“before swap values are :”<<a<<b;
swap(a,b);
cout<<“after swap values are”<<a<<b;
}
void swap(int x, int y) {
x=x+y;
y=x-y;
x=x-y;
cout<<In functions values are”<<x<<y;
}
```

Call By pointer/address

- In this instead of passing value, address are passed.
- Here formal arguments are pointers to the actual arguments
- Hence change made in the argument are permanent.

Example

```
void swap (int *,int *) ;
int main()
{
int a,b;
cout<<enter numbers”;
cin>>a>>b;
swap(&a,&b);
cout<<a<<b;
}
void swap(int *p,int *q)
{
int t;
t= *p;
*p= *q;
*q= t;
cout<<“In function” <<*p<<*q;
}
```

Call By Reference(Using Reference Variables with Functions)

- **To create a second name for a variable in a program, you can generate an alias, or an alternate name**

- **In C++ a variable that acts as an alias for another variable is called a reference variable, or simply a reference**

Declaring Reference Variables

- You declare a reference variable by placing a type and an ampersand in front of a variable name, as in `int &cash;` and assigning another variable of the same type to the reference variable

```
int someMoney;
int &cash = someMoney;
```

- A reference variable refers to the same memory address as does a variable, and a pointer holds the memory address of a variable

EXAMPLE REFERENCE VARIABLE

```
void main()
{
int i=10;
int &j=i; // j is a reference variable of I
cout<<“value”<<i<<“\t”<<j;
j=20;
cout<<“modified value”<<i<<“\t”<<j;
getch();
}
```

Output:-

Value 10 10

modified value 20 20

Example

```
#include<iostream.h>
 #include<conio.h>
 void swap (int &a, int &b)
 {
/* &a and &b are reference variables */
     int temp;
temp=a;
a=b;
b=temp;
}
main()
{
int i=5,j=10;
cout<<"Before swapping I = "<<i<<" J = "<<j<<endl;
swap(i,j);
cout<<"After swapping I = "<<i<<" J = "<<j<<endl;
 }
```

Recursion

- A recursive function is a function that calls itself either directly or indirectly through another function
- We use recursion where we want same statements or similar statements to be executed repeatedly as in case of iterative statements
- Whenever we are not sure about the number of times steps are to be performed, that kind of problem can be solved using recursive function

- **Requirements of Recursion:**

1. Function must call itself again and again
2. It must have an exit condition

- Major applications of recursion is game programming where a series of steps can be solved with recursion

Example: reverse using recursion

```
int reverse(int);
int main()
{
int n,r;
cout<<“enter number”;
cin>>n;
r=reverse(n);
cout<<“reverse of “<<n<<“is”<<r;
}
int reverse (int n)
{
int r=0, d;
if(n==0)
return 0;
else
{
d=n%10;
n=n/10;
r=r*10+d;
reverse(n);
} }
return (n);
}
```

Advantages of recursion

1. It make program code compact which is easier to write and understand.
2. It is used with the data structures such as linklist, stack, queues etc.
3. It is useful if a solution to a problem is in repetitive form.
4. The compact code in a recursion simplifies the compilation as less number of lines need to be compiled.

Disadvantages

1. Consume more storage space as recursion calls and automatic variables are stored in a stack.
2. It is less efficient in comparison to normal program in case of speed and execution time
3. Special care need to be taken for stopping condition in a recursion function
4. If the recursion calls are not checked ,the computer may run out of memory.