C++ PROVIDES SEVERAL MECHANISMS FOR PASSING PARAMETERS TO FUNCTIONS BY REFERENCE. A FULL DISCUSSION OF THESE METHODS REQUIRES THE INTRODUCTION OF POINTER VARIABLES.

THE FOLLOWING EXAMPLE ILLUSTRATES THE DIFFERENCE BETWEEN PASSING BY VALUE AND PASSING BY REFERENCE.

EX (DOES NOT WORK)

```
void swap (int a, int b)
{
    int temp;

    temp=a;
    a=b;
    b=temp;
}
```

THEN IN FUNCTION main:

```
int x=2, y=3;

swap(x,y);
cout << "x= " << x << "and y= "
     << y << endl;
```

WHAT WILL BE PRINTED BY THE ABOVE cout STATEMENT ?

(1) x = 2 and y = 3

OR

(2) x = 3 and y = 2

THE ANSWER IS (1), SINCE VARIABLES X AND Y ARE PASSED TO FUNCTION SWAP BY VALUE AND NOT BY REFERENCE. IN OTHER WORDS, SWAP RECEIVES COPIES OF X AND Y, KNOWN locally AS a AND b. CHANGES MADE TO a AND b DO NOT AFFECT THE ORIGINALS X AND Y.

IN FACT FUNCTION SWAP, AS WRITTEN, DOES NOTHING USEFUL.

ARRAY VARIABLES ON THE OTHER HAND ARE PASSED BY REFERENCE, AS THE FOLLOWING EXAMPLE SHOWS.

EX (DOES WORK)

```
void exchange(double L[], int i, int j)
{
    double temp;
    temp = L[i];
    L[i] = L[j];
    L[j] = temp;
}
```

WE CAN USE THIS FUNCTION IN THE SelectionSort PROGRAM TO EXCHANGE list[imax] WITH list[top] BY DOING

exchange(list, imax, top)

IN FUNCTION main. THIS EXAMPLE WORKS BECAUSE THE ARRAY list IS PASSED BY REFERENCE, NOT BY VALUE. i.e. list IS NOT COPIED TO THE FORMAL PARAMETER L. INSTEAD L BECOMES A LOCAL ALIAS, OR REFERENCE TO THE ARRAY list.

EXERCISE:
RE-WRITE THE SelectionSort.CPP EXAMPLE TO INCORPORATE THIS FUNCTION.

ANOTHER WAY TO PASS BY REFERENCE IN C++ (AND C) IS TO USE SO-CALLED POINTER TYPES, WHICH ARE IN SOME SENSE DUAL TO THE SIMPLE DATA TYPES STUDIED SO FAR.

EACH DATA TYPE IN C/C++ HAS A CORRESPONDING POINTER TYPE.

TYPE	POINTER TYPE	NAME
int	int *	POINTER TO int
char	char *	Pointer TO char
double	double *	Pointer TO double
bool	bool *	Pointer TO bool

WE ILLUSTRATE WITH TYPES int AND int *

RECALL THAT THE DECLARATION

```
int a;
```

ALLOCATES SPACE IN MEMORY SUFFICIENT TO STORE A SIGNED INTEGER TO BE REFERED TO BY THE SYMBOLIC NAME a.

THIS SPACE CAN ALSO BE REFERED TO BY ITS ADDRESS, WHICH IS A NUMERIC CODE FOR ITS LOCATION IN MEMORY.

THE ADDRESS OF AN int VARIABLE (OR ANY OTHER DATA TYPE) IS ACCESSED BY THE ADDRESS-OF OPERATOR: & . i.e. GIVEN THE ABOVE DECLARATION, THE EXPRESSION

```
&a
```

EVALUATES TO THE NUMERIC ADDRESS REFERED TO BY THE SYMBOLIC NAME a.

A VARIABLE OF TYPE int * STORES THE ADDRESS OF A VARIABLE OF TYPE int. THUS THE DECLARATION

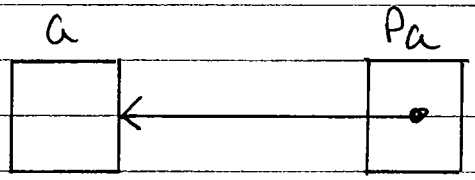
```
int * pa;
```

ALLOCATES pa TO BE A VARIABLE WHICH WILL STORE THE ADDRESS OF A LOCATION IN MEMORY WHICH ITSELF STORES AN INTEGER.

GIVEN THE ABOVE DECLARATIONS, WE CAN MAKE THE ASSIGNMENT

```
pa = &a;
```

THE POINTER VARIABLE pa THEN 'POINTS TO' THE int VARIABLE a.



THE int VALUE IN a CAN BE ACCESSED VIA pa BY USING THE INDIRECTION OPERATOR *, ALSO CALLED THE DEREFERENCE OPERATOR, OR VALUE-AT OPERATOR.

THUS THE EXPRESSION

```
*pa
```

EVALUATES TO WHATEVER THE VALUE OF a IS.

Ex.
```
int a;
int* pa;

a = 10;
pa = &a;

cout << &a << '=' << pa << endl;
cout << a << = << *pa << endl;
```

IF THE COMPILER DECIDES TO LET THE SYMBOLIC NAME a REFER TO THE ADDRESS 66224 (FOR INSTANCE), THEN THE OUTPUT WILL BE:

```
66224 = 66224
10 = 10
```

THE EXPRESSION *pa CAN ALSO BE USED TO CHANGE THE VALUE OF a, BY PLACING *pa ON THE LEFT SIDE OF AN ASSIGNMENT. CONTINUING THE LAST EXAMPLE

```
*pa = 20;
cout << a << '=' << *pa << endl;
```

CAUSES THE OUTPUT:

```
20 = 20
```

IN FACT WE CAN EXCHANGE TWO VALUES WITHOUT USING THEIR SYMBOLIC NAMES:

Ex.
```
int a, b, temp;
int* p1;
int * p2;

a = 10;
b = 20;
p1 = &a;
p2 = &b;

Cout << "a=" << a << " b=" << b << endl;

tempP = *p1;
*p1 = *p2;
*p2 = tempP;

cout << "a=" << a << " b=" << b << endl;
```

OUTPUT:

```
a=10 b=20
a=20 b=10
```

THE POINTER VARIABLES p1, p2 CAN THEMSELVES BE ALTERED TO POINT ELSWHERE. CONTINUING...

```
p1 = p2;
cout << "a=" << a << endl;
cout << "b=" << b << endl;
cout << "*p1" << *p1 << endl;
cout << "*p2" << *p2 << endl;
Cout << "p1=" << p1 << endl;
cout << "p2=" << p2 << endl;
```