Day / Date

POST MID - "FUNCTIONS"

IMPORTANCE OF FUNCTIONS :-

(i) Built for the specific operation;
(ii) Used in different programs by different people;
(iii) Enhances readibility;
(iv) Allows programs to be modularized.
* Funct. arguments can be all;
consts e.g sqrt (4)
variables e.g sqrt (x)
expressions eg sqrt (sqrt (x)) ; or
sqrt (3-6x)

LIST OF PREDEFINED FUNCTS :-

	name	at	input type	output type	use
(i)	abs (x)	<cstdlib>	int	int	(-7) = 7
(ii)	ceil (x)	<cmath>	double	double	(56.34) = 57.00
(iii)	cos (x)	<cmath>	double	double	
(iv)	exp (x)	<cmath>	double	double	for any x if gives e^x
(v)	Fabs (x)	<cmath>	double	double	(-7.5) = 7.5
(vi)	Floor (x)	<cmath>	double	double	56.34 = 56.00
(vi)	Pow (x, y)	<cmath>	double	"	x^y
(vii)	tolower (x)	<cctype>	int	int	give lowercase
(viii)	to upper (x)	<cctype>	int	int	give uppercase

Day / Date

ASPECTS OF FUNCT DEFINITION:-

Customized functs are made consisting of steps;

(i) Take in data.

(ii) Perform operation.

(iii) return somevalue.

return value type functname(↑list Parametre)

```
{
    declarations &
    statements
}
```

Generating Random Numbers:-

For this we use funct from <cstdlib>

→ Rand() ; is used to generate a random number.

(from 0 / 32767)

```
int main()
    {
        int number = rand()
        cout<< Number;
        return 0;
    }
```

output = 54 (or anyother but consecutively same 54/other again and again)

→ We can also seed the rand funct but a single seed gives a single random number again and again.

```
intmain()
 {
  srand(6);
```

Day / Date

```
int number = rand();
cout << number;
}
```

This time the function will not generate 54 but rather 41 but for the seed '6' 41 will be generated consecutively.

→ Inorder to consecutively change the random numbers generated we can use time(0) from <ctime> to be seeded in srand.

```
int main
{
srand(time 0);
intnumber = rand();
cout<< number;
}
```

Scaling RANDOM NUMBERS:-

If we seed the random number by time(0) we might get a very large random, so we can scale or limit the range of random number.

```
int main
{
srand(time0);
int number = rand()%.6+1;
cout <<number;
return 0;
}
```

Call by reference or call by address / location.

Day / Date

VOID FUNCTION:-

void is often used when we avoid returning any value to the funct. However we can perform similar operation from it.

Example:-

"The grading example from the slides".

DESIGNING SQUARE ROOT FUNCT:-

To design the square root funct (we obv take some value from the user to square root upon.)

(i) we set low value = 1.0 & high value = (user input)

(ii) then we let another variable
estimate = (high + low) / 2

(iii) if estimate * estimate > user input
∴ estimate = new high = high.

(iv) or it would be (<)
∴ estimate = new low = low.

(v) always return (high+low/2);

Day / Date

inline functs
reduces compilation time.

TASKS :-

(i) even/odd numbers. ✓ (iii) Calculator, ✓
*(ii) Prime numbers. ✓ *(iv) dices / game
*(v) largest / smallest ✓ *(vi) Print last digits ✓

PRIMIE NUMBERS :-

```
int checkPrime (int num)
 {
    if (num < 1)
     { return 0; }

   for (int i = 2 ; i <= num ; i++)
    {  if (num % i == 0)
        {
          return 0;
        }
    return 1;
      }

int main()
  {
   int number = 5
  if (checkPrime(5))
     {  cout << " The number is Prime" << endl;
      }
  else
     { cout << The no. is not Prime" << endl;
     }  ---
```

Galaxy

in default argument we set up a value ; which is only used when no value is passed by the user.

LAST DIGIT PRINT:-

```
Int main ()
 {
    int a = 256.
 int last digit = a%.10.

   cout << "The last digit is" << last digit <<
                                        endl;
 }
```

6 Ans!.

FIRST DIGIT PRINT:-

```
Int main ()
{
    int a/number = 256
  while (a/number >= 10)
{  number = number/10.
  }

   cout <<number;
}
```

MODULUS (FUNCTION):-

fmod from <cmath>

```
double remander (double a, double b)
{
    return fmod(a,b);
}
```

Galaxy

Day / Date

19th Dec - 2023.

Array :-

(i) A collection structure of related data items.

(ii) Static - entity ; has the same size throughout the program.

(iii) The variables stored in the adjacent memory locations and have the same name & type.

Example `int test [5]`

- `int` ↓ data type of array.
- `test` ↓ name of array
- `[5]` ↘ size of array (capacity ... declarator).

(iv) Size of the array is the number of bytes allocated for the array; i.e.

(number of elements) * (bytes needed for each element.)

Example:

```
double vol [DSIZE] // holds 10 doubles.
                   // array is 80 bytes.
```

(v) `test [0]` ↓ =1.0 First element, `test [1]` ↓ =2.0 second element, `test [2]` ↓ =3.0 third element. array of 3 elements ie 3 indexed variable

(i) The largest index is one less than the size of the array and the first index is zero.

Day / Date

(vi) Error ; when array called without index / subscript.

Error is generated when no. of elements of array > size of array.

(vii) Array subscript / index can be

cin test [3]; int const.

cin test [i]. int variable.

cin test [i+j]; int expression.

* cin test [test[0]] int expression.

(viii) Array loads & displays one element at a time.

(ix) Const variables must be initialized when inputted as they cannot alter later.

(x) Note (string in array)

char First [ ] = "hello";

cout << First; (= hello)

cout << First [0] (= h)

cout << //second First [1] (= e)

Day / Date

(xi) User input - unknown size array printing;

```
for (int i=0 ; string[i] != '\0' ; i++)
```

(xii) value name of array is the address of 1st element.

(xiii) Passing arrays to functions).

Example:-

```
void myfunction ( int b[] , int arraysize);
```

or simply

```
void myfunct ( int [] , int];
```

↓ represents the array.

↓ represents the size of our array.

(xiv) If we want a char array to print space; it is to be done at the cout statement and not in the array itself.

Error in our compilation needs.

```
char string[20] = {"hello there"};
cout << string;      (=hello // now the array thinks that the space comes and no more chars in it).
```

(xv) Array as a whole is termed / considered as call by reference in a funct

(xv) Individual array index is not call by reference.

Day / Date

(xvi) int number [3] = {1, 2, 3}

for (int i=0 ; i < size/3 ; i++).

{ cout << "\n" << number [i]++;

this is post increment

* if more couts are used afterward the output will be:

2
3
4.

∴ output rn.

=> {1, 2, 3}

(xvii) int number [3] = {1, 2, 3}

for (int i=0 ; i<size/3 ; i++)

{

cout << number [i+1].

}.

1
3 } Ans

(xviii) In array we enter rows first & then colams

Example b [2] [2].

Day / Date

IMP. OF POINTERS:-

Pointers can directly access the 'memory' & thus decreases the time of execution.
The use can reduce "excessive memory allocation."

POINTER CONCEPTS:-

```
int *ptr      // initiallizing the pointers
ptr = dat or ptr = &dat[0]  // giving address o
int a = *ptr + 5; //                the first index of
*ptr += 1 //                        array
```

getting the value stored at dat[
to add with 5.

permanently increasing the value of it.

*ptr(++) → increment of one on the address.

TYES OF CONST. POINTER:-

```
int *const myptr = &x
const int *myptr = &x
const int *const myptr = &x
```

(i) *(data+4) <=> data[4]

(ii) Ptr[1] = Ptr[2] = Ptr[3]
=> Data[1] = Data[2] = Data[3]

Day / Date

SWAPPING:-

```
int main()
{
   int x=5, y=7;
   swapit (&x, &y).      endl
   cout <<"x = " << x <<";
    cout <<"y = " << y<<";
                        endl
}
                int
void swapit (*x, int*y)
  {
      int temp
       temp = *x;
       *x = *y;
       *y = temp;
  }
```

BUBBLE SORT & SWAPPING:- COMPLETE ON YOUR OWN.

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

void bubblesort (int *, const int);
void swap ( int *const element1ptr, int *const
                                    element
                                    2ptr).

int main ()
{  int arraysize = 10;
   int a [10] = { 2, 6, 4, 8, 10, 12, 89,
                  68 ...}.
```

* after pointing a variable with pointers both the variable & pointer can point/change the variable's stored value.

Day / Date

Pointer to pointer

dereferenced

ANALYZE CAREFULLY:-

cout for pointer funcs & call by ref func

dong this with the dereference the dereference will generate an error itself and

```
int * badmull (int x, int y)
{
    int *out = x*y;
    return out;
}

int & badmull (int x, int y)
{
    int out = x*y;
    return *out;
}

int main ()
{
 int x = 5;
 int y = 5;

 int *result = *badmull (5, 5);

 int &result2 = badmull (5, 5);
```

Day / Date

DYNAMIC MEMORY :-

The memory allocations while writing the codes are divided into four types.

Heap →

This memory doesn't grow when the program runs!

stack → here all the funct calls & local var

static/global → here all the global variables etc. are stored.

code (TEXT) → here the general inst. are stored.

* at stack by default some memory is allocated but if our program exhaust that memory it is called as 'stack overflowing' & the program will crash.

* At a heap, the memory allocated is not restricted unlike stack therefore called free pool memory / heap / dynamic memory.

* (although heap & stack are data structures but not considered as such now).

Day / Date

* to use heap thus we use new while initializing and to access an address in heap we need to require/acquire the use of pointers in the code.

```
strncpy(destination, source, sizeof(destination) -1);
destination[sizeof(destination)-1] = '\0';
```