The CreAdult workshop module:
A Four-Phase Curriculum
Technology Literacy

1. **Duration:** *180 min (classroom, excluded exploitation) or 5 hours – self-paced*
2. **Learning Outcomes:** *After completion of the module, participants should be able to*
    - *define basic terminology in algorithms and computer programming*
    - *have knowledge about the fundamental concept of algorithms and programming*
    - *classify and explain the stages of software development life cycle.*
    - *apply light computer programs including decision structures, loop, function and array*
    - *and, they should have a good sense of the beauty of programming*

3. **Materials required:** *Projector to watch the videos, internet connection, online C++ compiler (cpp.sh)*
4. **The four phases of the CreAdult curriculum:**

Energizer:

1. *Mention about why algorithms and programming is the core skill of the 21st Century*
2. *Watch the Introduction video about the algorithms*

Main activity:

1. *Deliver the lecture including the following topics using the lecture presentation.*
    - *Software development life cycle (SDLC)*
    - *Hello World, First Program in C++*
    - *Variables in C/C++*
    - *Control structures; decision making structures*
    - *Control structures; loops*
    - *Functions*
    - *Arrays*
2. *Implement the code in each topic of the module utilizing online C++ compiler (cpp.sh).*

Evaluation:

1. *Implement the interactive task 1.*
2. *Implement the interactive task 2.*
3. *Implement the interactive task 3.*
4. *Implement the interactive task 4.*
5. *Implement the interactive task 5.*

Exploitation:

1. *Fulfil the home task 1, 2 and 3 in the document titled "Exploitation - Technology". Find the outputs with given inputs in the document.*

The Guidelines for adult education professionals
Technology Literacy

Energizer:
The educator starts presenting the topic by explaining why algorithms and programming are the core skill of the 21st Century. (See the introduction video transcript).

The educator opens the video from the link https://www.youtube.com/watch?v=cDA3_5982h8 (Exact Instructions Challenge - THIS is why my kids hate me. | Josh Darnit)
After the video is watched the educator

- *opens a discussion on why algorithms and programming are the core skill of the 21st Century and,*

Hints and Tips for educator:

- *For the discussion part, encourage participants to try to find examples of algorithms and programming in the universe.*

Main activity:
The educator deliver lecture using the presentation. The participants shall be actively involved during the presentation and shall fulfil all the Interactive tasks.

Hints and Tips for educator:
For each topic,

- *First, explain the theory part of the topic in the presentation.*
- *Perform the corresponding exercise using the online compiler at cpp.sh*
- *Ask the participants to implement and test the same exercise on their own utilizing the compiler. Encourage them to change the value of the variable in the code to get different output*

Evaluation:
After delivering lecture in the main activity, the educator asks the participants to implement the interactive tasks 1, 2, 3, 4 and 5 on their own using the online compiler at cpp.sh. The main idea is to understand function of the concept given during the main activity.

Hints and Tips for educator:

- *Ensure that all participants open the online compiler at cpp.sh*
- *Distribute the code of the program to be implemented*
- *Make sure that the participants are actively involved in the process and*
- *Help them during the implementation part*

Exploitation:
The participants are encouraged to implement the codes in the file titled "Exploitation - Technology"

Hints and Tips for educator:

- *On the next day ask the participants whether they could run the codes or not.*
- *They can also work in groups of max 3 persons while implementing the codes.*

Annexes: Content for the Workshop Module: Technology Literacy

A-Introduction Video Transcript
B-Lecture Video Transcript
C-Gamification-Based Interactive Tasks
D-Readings and References

A. Introduction Video Transcript

Hi! Welcome to **Technology Literacy Module**.
Algorithms and Programming is the Core Skill of the 21st Century
Why?

- *No programming = No life to modern generation*
- *It's big business*
- *Career opportunities*
- *The biggest reason to learn to code—to create value with your own ideas*

Steve Jobs underlines very important reality.
"Everybody in this country should learn how to program a computer…because it teaches **you how to think.**"
*He does not mean that everybody in the country should be computer programmer but to learn how to program a computer. The aim is to become "***a problem solver***".*
So, "Algorithms and Programming is **the way of thinking** *which all should have this skill in the 21st Century.*
Now we will try to understand fundamental concept of algorithms and programming regardless of programming languages.
But before that – let's watch the first video from external resources section.
https://www.youtube.com/watch?v=cDA3_5982h8 (Exact Instructions Challenge - THIS is why my kids hate me. | Josh Darnit)

B. Lecture Video Transcript

The main objective of this module is to review the fundamental concept of algorithms and programming regardless of programming languages. During the module, C++ programming language which is the most important one in the programming language world.

The followings are the topics of the module,

1. *Software development life cycle (SDLC)*
2. *Hello World, First Program in C++*
3. *Variables in C/C++*
4. *Control structures; decision making structures*
5. *Control structures; loops*
6. *Functions*
7. *Arrays*

Let's start with the SDLC

1. What is Software Development Life Cycle (SDLC)?

SDLC is a methodology made up of 6 phases: analysis, design, coding, testing, deployment and maintenance.

Analysis
Goal: *To gather requirements and define the direction of the software engineering process.*
Mistakes made during the analysis phase are the most expensive to fix later. This is the only time to build a foundation for your future product:

Design
Goal: *To convert requirements into detailed software architecture.*
The vital thing here is a vast understanding of the context for the existence of software engineering services. The more detailed the description of how the application needs to be created would be, the less additional input developers will require at the next coding stage.

Coding
Goal: *To translate the design of the system into code.*
Using the design description, programmers code the modules using the chosen programming language. The coding tasks are divided between the software engineering team members according to their skillset. Front-end developers create codes for displaying an application or product UI and the elements users need to interplay with the site. Their Back-end counterparts are in charge of the technical side of a product.

Testing
Goal: *Code verification and bugs detection.*
Software is completely free of bugs and compliant after the testing operations.

Deployment,
Goal: *Software delivery to a target device.*
Software deployment refers to the process of running a product on a server or device and can be summarized in three general phases: preparation, testing and deployment

Maintenance
Goal: *Ongoing security monitoring and update.*
The process of software development is a never-ending cycle as the plan rarely turns out perfect when it meets reality. In most cases, product maintenance is the continuous phase intended to keep the software stable and up to date.

2. Hello World, First Program in C++

It is customary to start with the "Hello World" in the programming world. Let's write our first program by following this tradition.

```
//A first program in C++
#include <iostream>
using namespace std;

int main (void)
{
        cout << "Hello world!" << endl;
        return 0;
}
```

Let me explain all line in the program.
#include <iostream>

- *is a directive to the C preprocessor. Lines beginning with # are processed by the preprocessor before the program is compiled.*

using namespace std;

- *allows reference to string, cout, and endl without writing std::string, std::cout, and std::endl*

int main (void)

- *is a part of every C++ program. The parentheses after* **main** *indicate that main is a program building block called a* **function***.*

cout << "Hello world!" << endl;

- *instructs the computer to perform an* **action***, namely to print on the screen the* **string** *of characters marked by the quotation marks.*

return 0;

- *is included at the end of every main function.*

3. Variables in C/C++

A **variable** *is a location in memory where a value can be stored for use by a program.*
int *integer1;*
int *integer2;*
int *sum;*
These definitions specify that the variables integer1, integer2 and sum are of type int, which means that these variables will hold **integer** *values, i.e., whole numbers such as 7, –11, 0, 31914 and the like. Programming languages have different kind of variable to store numeric and alfa-numeric values such as* **float** *and* **char***.*
Now let's code a program that calculates the salary of a salesperson in the company.

PROBLEM STATEMENT 1

According to management policy of a company, a salesperson is payed a fixed salary and a premium. Develop a program to calculate the salary of a salesperson in the company.

ANALYSIS

According to management policy of a company,

salary = fixedSalary + premium

Variables can be defined as shown in the table below

Variable	Variable Name	Type of Variable	Input/ Output
Salary of a salesperson	*salary*	*float*	*O*
Fixed salary	*fixedSalary*	*float*	*I*
Premium	*premium*	*float*	*I*

DESIGN

1. *Define variables*
   Use the above table to define variables and their types.
2. *Take inputs from user.*
   Fixed salary and premium
3. *Calculate the salary*
   salary = fixedSalary + premium
4. *Output the salary*

IMPLEMENTATION – CODE

```
//Calculating monthly salary of a salesperson.
#include <iostream>
using namespace std;
```

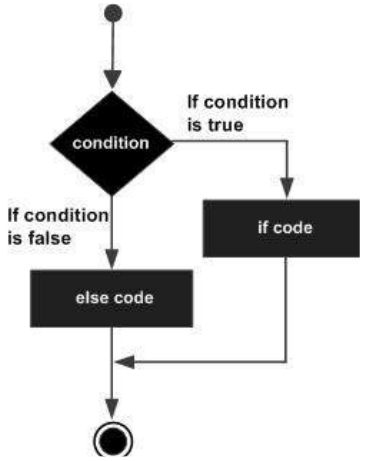
```
int main ()
{
        //1. Define variables
        float salary, fixedSalary, premium;

        //2. Take inputs from user.
        cout<< "Enter fixed salary: ";
        cin>>fixedSalary;
        cout<< "Enter premium: ";
        cin>>premium;

        //3. Calculate the salary
        salary = fixedSalary + premium;

        //4. Output the salary
        cout<< "Monthly salary: " << salary <<endl;

        return 0;
}
```

TESTING

Please test the code providing proper inputs according to the requirement of the program using the online compiler at cpp.sh

4. Control structures; decision making structures

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false



PROBLEM STATEMENT 2

According to management policy of a company, a salesperson who sells three different goods gets paid a fixed salary and a premium depending on total earnings from sales and rate of premium. And, the rate of premium is doubled if the total earnings exceed a level determined by the administrator of the company. Develop a program to calculate the salary of a salesperson in the company.

ANALYSIS

1. *salary= fixedSalary + premium*

   1.1. *premium= totalEarnings*rateOfPremium*

      1.1.1. *if (totalEarnings>= level)*
      rateOfPremium= 2 rateOfPremium*

      1.1.2. *totalEarnings = volume1 * unitPrice1 + volume2 * unitPrice2 + volume3 * unitPrice3*

Variable	Variable name	Type	Input/ output
Salary	*salary*	*float*	*c/o*
Fixed salary	*fixedSalary*	*float*	*i*
Premium	*premium*	*float*	*c*
Rate of premium	*rateOfPremium*	*float*	*i/d*
Total earnings	*totalEarnings*	*float*	*c*
Volume 1	*volume1*	*integer*	*i*
Volume 2	*volume2*	*integer*	*i*
Volume 3	*volume3*	*integer*	*i*
Unit price 1	*unitPrice1*	*float*	*i/d*
Unit price 2	*unitPrice2*	*float*	*i/d*
Unit price 3	*unitPrice3*	*float*	*i/d*
Reward level	*level*	*float*	*I*

DESIGN

1. *Define variables*
   Use the table above to define variables

2. *Take input from user*
   fixedSalary, volume1, volume2, volume3

3. *Calculate salary*
   3.1 Calculate total earnings
   1.1.2
   3.2 Determine rate of premium
   1.1.1
   3.3 Compute premium
   1.1
   3.4 Calculate salary
   1

4. *Output salary*
   salary

IMPLEMENTATION-CODE

```
//Calculate salary V1
#include <iostream>
using namespace std;
int main ()
{
        //1. Define variables
        float salary, fixedsalary, premium, fixedSalary, rateOfPremium, totalEarnings, unitPrice1,
        unitPrice2, unitPrice3, level;
        int volume1, volume2, volume3;

        //2. Take input from user
        cout<< "Please enter your fixed salary: "; cin>>fixedSalary;

        cout<< "Please enter rate of premium: "; cin>>rateOfPremium;

        cout << "Please enter number of item1 sold: "; cin>>volume1;
        cout << "Please enter number of item2 sold: "; cin>>volume2;
        cout << "Please enter number of item3 sold: "; cin>>volume3;

        cout << "Please enter unit price 1: "; cin>>unitPrice1;
        cout << "Please enter unit price 2: "; cin>>unitPrice2;
        cout << "Please enter unit price 3: "; cin>>unitPrice3;

        cout << "Please enter reward level: "; cin>>level;

        //3. Calculate salary
        //3.1 Calculate total earnings

        totalEarnings = volume1 * unitPrice1 + volume2 * unitPrice2 + volume3 * unitPrice3;

       //3.2   Determine rate of premium
        if (totalEarnings>= level)
                rateOfPremium = 2* rateOfPremium;

        //3.3   Calculate premium
        premium= totalEarnings * rateOfPremium;

        //3.2   Calculate salary
        salary= fixedSalary + premium;

        //4.    Output salary
        cout<< "Salary is "<<salary;

        return 0;
}
```

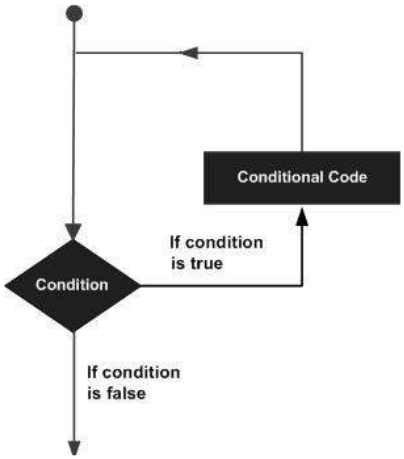
TESTING

Please test the code providing proper inputs according to the requirement of the program using the online compiler at cpp.sh

5. Control structures; loops

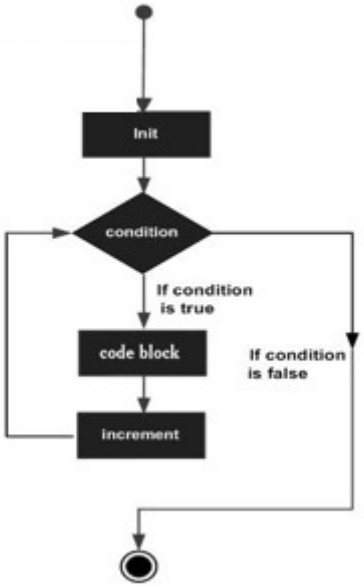
There may be a situation, when you need to execute a block of code **several number of times**.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages:



C/C++, as well as the other programming languages provide the following types of loop to handle looping requirements.
for loop *executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.*
while loop *repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.*
do-while loop, *like a while statement, except that it tests the condition at the end of the loop body.*
nested loops, *you can use one or more loop inside any another* **while, for or do-while** *loop.*
In this module, we are going to handle **for loop***.*

A **for loop** *is a repetition control structure that allows you to efficiently write a loop that needs to execute* **a specific number** *of times.*



Here is the flow of control in a for loop:

- *The* **init** *step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.*

- *Next, the* **condition** *is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.*
- *After the body of the for loop executes, the flow of control jumps back up to the* **increment** *statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.*
- *The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.*

PROBLEM STATEMENT 3

According to the management policy of a company, a salesperson is paid a fixed salary and a premium. Develop a program to calculate the salaries of some number of salespeople in the company and total payment for the salaries.

ANALYSIS

the num of sales person= 3
total salary = 0
sales person no (SP #): 1 2 3
iter.1: SP# 1;
take fixed salary and premium for SP1
calculate salary of SP1 (salary1 = fixedSalary1 + premium1)
accumulate salary of SP1 in total salary (totalSalary = totalSalary + salary1)

iter.2: SP# 2;
take fixed salary and premium for SP2
calculate salary of SP2
accumulate salary of SP2 in total salary

iter.3: SP# 3;
take fixed salary and premium for SP3
calculate salary of SP3 (salary3 = fixedSalary3 + premium3)
accumulate salary of SP3 in total salary (totalSalary = totalSalary + salary3)

Variable	Variable Name	Type of Variable	Input/ Output
Salary	*salary*	*float*	*O*
Fixed Salary	*fixedSalary*	*float*	*I*
Premium	*premium*	*float*	*C*
Total Salary	*totalSalary*	*float*	*O*

DESIGN

1. Define and initialize variables
use table above
totalSalary=0
2. Take input from user
numofSP,
3. Calculate salaries of salespeople and total salary

For each salesperson

- *calculate salary for* **current** *salesperson*
  - *Take fixedSalary and premium for* **current** *salesperson*
  - *Calculate salary for* **current** *salesperson: salary = fixedSalary+premium*
  - *Output salary of* **current** *salesperson*
- *calculate total salary*
  - *Accumulate salary of* **current** *salesperson in total salary: totalSalary=totalSalary + salary*

4. Output total Salary

IMPLEMENTATION – CODE

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

int main ()
{
    //Define variables
    float salary, fixedSalary, premium, totalPayment = 0;
    int number Person = 3;

    //For each salesperson
    for(int salesperson = 1;  salesperson<= numberPerson; salesperson ++)
    {
        //Calculate salary for current salesperson
        //Take fixedSalary and premium for current salesperson
        cout << "Sales Person-" << salesperson << endl ;
        cout << "Enter Fixed Salary:";    cin >> fixedSalary;
        cout << "Enter Premium:";    cin >> premium;

        //Calculate Salary
        salary = fixedSalary + premium;

        //Output Salary of current salesperson
        cout <<"The salary of salesperson - " << salesperson << " is: "<< salary << endl ;

        // Calculate total salary
        totalPayment = totalPayment+ salary;

    }

    // Output total Salary
    cout<<"Total payment is: " << totalPayment;
    return 0;
}
```

TESTING

Please test the code providing proper inputs according to the requirement of the program using the online compiler at cpp.sh

6. Functions

The function in C++ language is also known as procedure or subroutine in other programming languages. To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

There are two types of functions in C programming:

1. Library Functions: *are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.*

2. User-defined functions: *are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.*

PROBLEM STATEMENT 4

Let's code a program to find maximum number of three numbers

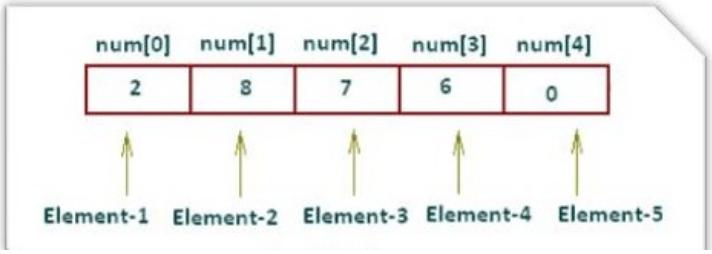
```
#include <iostream>
using namespace std;

int findMax(int,int,int);   //function prototype
int main ()
{
        //Define variables
        int num1, num2, num3, maxNum;

        //Take numbers
        cout<<"Enter first,second and third number"<<endl;
        cin>>num1;
        cin>>num2;
        cin>>num3;

        maxNum=findMax(num1,num2,num3);
        cout<<"Max is: "<<maxNum;
        return 0;
}

int findMax(int n1, int n2, int n3)
{
         int maxNum;
         if(n1>n2)
                maxNum=n1;
          else
                maxNum=n2;

         if(maxNum<n3)
                maxNum=n3;

        return maxNum;
}
```

TESTING

Please test the code providing proper inputs according to the requirement of the program using the online compiler at cpp.sh

7. Arrays

An array is a collection of elements of the same type that are referenced by a common name. All the elements of an array occupy a set of contiguous memory locations. First element at position 0.



Why we need to use arrays?

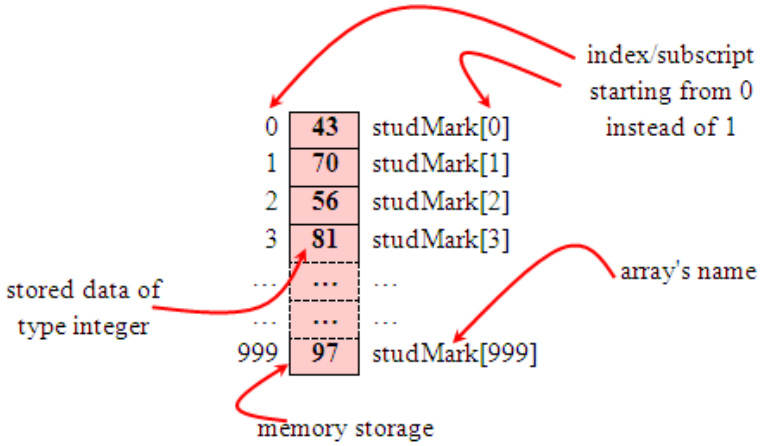
Example:
Let's assume that we have a list of 1000 students' marks of an integer type.
If using the basic data type (int), we will declare something like the following

```
int main(void)
{
        // declaration of 1000 variables
        int studMark1, studMark2, studMark3, studMark4,
        …,
        …,
        studMark998, stuMark999, studMark1000;


        // rest of the program
        …
        …
        return 0;
}
```

Using an array, we just declare like this,

```
int main(void)
{
        // declaration of an array of 1000 elements
        int  studMark[1000];

        // rest of the program
        …

        return 0;
}
```

This will reserve 1000 contiguous memory locations for storing the students' marks. Graphically, this can be depicted as in the following figure.



Declaring Array

When declaring arrays, we need to specify

- *Name*
- *Type of array*
- *Number of elements*

int arrayType arrayName[numberOfElements];
for examples:
int c[10];
float myArray[3284];

PROBLEM STATEMENT 5
Now let me write a program to calculate total of four sold products with some given value

```
#include <iostream>
using namespace std;
int main ()
{
        //Define variable
        // those four numbers in braces are randomly chosen by us
        int saleProduct[4]={45,25,13,56};

        int total=0;

        //Calculate total sales
        for(int i=0; i<4; i++)
        {
                total=total + saleProduct[i];
        }

        //Output total
        cout<<"Total is: "<<total;

        return 0;
}
```

TESTING
Please test the code providing proper inputs according to the requirement of the program using the online compiler at cpp.sh

C. Gamification-Based Interactive Tasks

Interactive task 1.

Find the value of the "salary" in problem statement1 with the following values of the variables
fixedSalary = 5000
premium = 1000
Implement the problem by coding in online C++ compiler (cpp.sh)

Output:

```
Monthly salary:
    a. 6000
    b. 7000
    c. 5000
```

Interactive task 2.

What is the value of the salary after running the code in problem statement 2?

Variable	Variable name	Value
Salary	*salary*	*?*
Fixed salary	*fixedSalary*	*3000*
Rate of premium	*rateOfPremium*	*0.10 (zero **dot** ten)*
Volume 1	*volume1*	*100*
Volume 2	*volume2*	*200*
Volume 3	*volume3*	*300*
Unit price 1	*unitPrice1*	*10*
Unit price 2	*unitPrice2*	*8*
Unit price 3	*unitPrice3*	*5*
Reward level	*level*	*5000*

Output

```
salary is
    a. 4090
    b. 3410
    c. 2010
```

Interactive task 3

Use the code in problem statement 3 to find the total salary that the company needs to pay for the 3 employees. All employees have the same fixed salary and premium.
fixed salary = 3500
and premium = 500

Output

```
Total payment is:
    a. 11000
    b. 10000
    c. 12000
```

Interactive task 4

What is the output of the program in problem statement 4 with the inputs 39, 1, 41?

Output

```
Max is:
    a. 41
    b. 42
    c. 81
```

Interactive task 5

What is the output of the program in problem statement 5, if the following array is used in the code?
int saleProduct[4]={45,20,15,10};

Output

```
Total is:
    a. 80
    b. 90
    c. 100
```

D. Reading

Walter Savitch, Kenrick Mock-Absolute C++
Stephen Prata, "C Primer Plus"
Deitel, "C How to Program"
Günay Karlı, "C'de Problem ÇözmeMantığı"