CS252M

Submitted by:

Vijeeth J Poojary

231EE263

NITK

Submitted to:

Dr. Radhika B S

Course Instructor-CS252M

NITK Surathkal.

National Institute of Technology Karnataka [NITK], Surathkal.

NH 66, Srinivasnagar, Surathkal, Mangalore, Karnataka-575025

Activation Records

In computer science, a function call is a fundamental operation. To manage the data associated with each call, the system uses a data structure on the **call stack** known as an **activation record** (or a **stack frame**).

- **What it is:** An activation record is a block of memory allocated on the call stack every time a function is called. It serves as the function's private, temporary workspace for that specific invocation.
- **Its Purpose:** It holds all the information necessary for the function to execute correctly and, crucially, to return control to the caller when it is finished. Because each function call gets its own record, a function can even call itself (recursion) without its data getting mixed up.
- **What it Contains:**
    - **Arguments:** The values or references passed into the function by the caller (e.g., `health`, `&armor`).
    - **Local Variables:** Variables declared inside the function's scope, which are only accessible to that function (e.g., `damage` and `healAmount` in `battleRound`).
    - **Return Address:** The memory location in the calling function where the program should resume after the current function completes. This is how the program knows where to go back to.
    - **Saved Frame Pointer:** A pointer to the previous function's activation record. This links the stack frames together in a chain, allowing debuggers to trace back the sequence of calls.
- **Lifecycle:** The management of activation records is straightforward. An activation record is **pushed** (created) onto the top of the stack when a function is called, and it is **popped** (destroyed) when the function returns. This "Last-In, First-Out" (LIFO) behavior makes the stack a highly efficient structure for managing nested function calls.

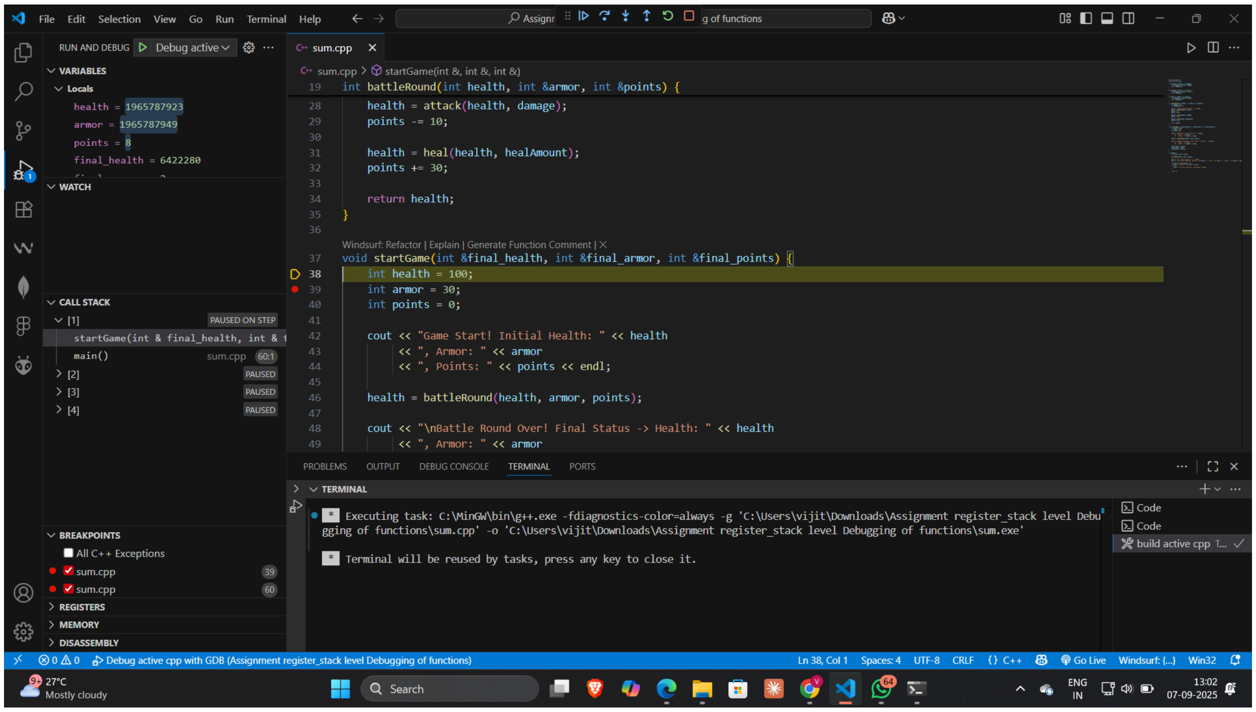
Program

```
#include<iostream>

using namespace std;


int attack(int health, int damage) {

    int newHealth = health - damage;

    return newHealth;

}


int defend(int health, int armor) {

    int newHealth = health + armor/2;

    return newHealth;

}


int heal(int health, int amount) {

    int newHealth = health + amount;

    return newHealth;

}


int battleRound(int health, int &armor, int &points) {

    int damage = 30;
    int healAmount = 5;


    cout << "\nEntering Battle Round..." << endl;

    health = defend(health, armor);

    armor -= 5;

    points += 20;


    health = attack(health, damage);

    points -= 10;


    health = heal(health, healAmount);

    points += 30;


    return health;

}


void startGame(int &final_health, int &final_armor, int &final_points) {

    int health = 100;

    int armor = 30;

    int points = 0;


    cout << "Game Start! Initial Health: " << health
```

```
          << ", Armor: " << armor
          << ", Points: " << points << endl;

    health = battleRound(health, armor, points);

    cout << "\nBattle Round Over! Final Status -> Health: " << health
          << ", Armor: " << armor
          << ", Points: " << points << endl;

    final_health = health;
    final_armor = armor;
    final_points = points;
}

int main() {
    int health, armor, points;

    startGame(health, armor, points);

    cout << "\n--- Back in main() ---" << endl;
    cout << "Final Stats Received - Health: " << health << ", Armor: " <<
armor << ", Points: " << points << endl;
```

```
    if(points >= 50 && health > 0) {

        cout << "You won. Play again" << endl;

    } else {

        cout << "You Lose. Game Over. Play again" << endl;

    }



    return 0;

}
```

Screen Shots



```
health = 1965787923
armor = 1965787949
points = 8
final_health = 6422280
```

```
[1]
  startGame(int & final_health, int & f
  main()                                  sum.cpp 60:1
[2]
[3]
[4]
```

```
sum.cpp 39
sum.cpp 60
```

```
int battleRound(int health, int &armor, int &points) {
    health = attack(health, damage);
    points -= 10;

    health = heal(health, healAmount);
    points += 30;

    return health;
}

Windsurf: Refactor | Explain | Generate Function Comment | X
void startGame(int &final_health, int &final_armor, int &final_points) {
    int health = 100;
    int armor = 30;
    int points = 0;

    cout << "Game Start! Initial Health: " << health
         << ", Armor: " << armor
         << ", Points: " << points << endl;

    health = battleRound(health, armor, points);

    cout << "\nBattle Round Over! Final Status -> Health: " << health
         << ", Armor: " << armor
```

```
 *  Executing task: C:\MinGW\bin\g++.exe -fdiagnostics-color=always -g 'C:\Users\vijit\Downloads\Assignment register_stack level Debu
gging of functions\sum.cpp' -o 'C:\Users\vijit\Downloads\Assignment register_stack level Debugging of functions\sum.exe'

 *  Terminal will be reused by tasks, press any key to close it.
```

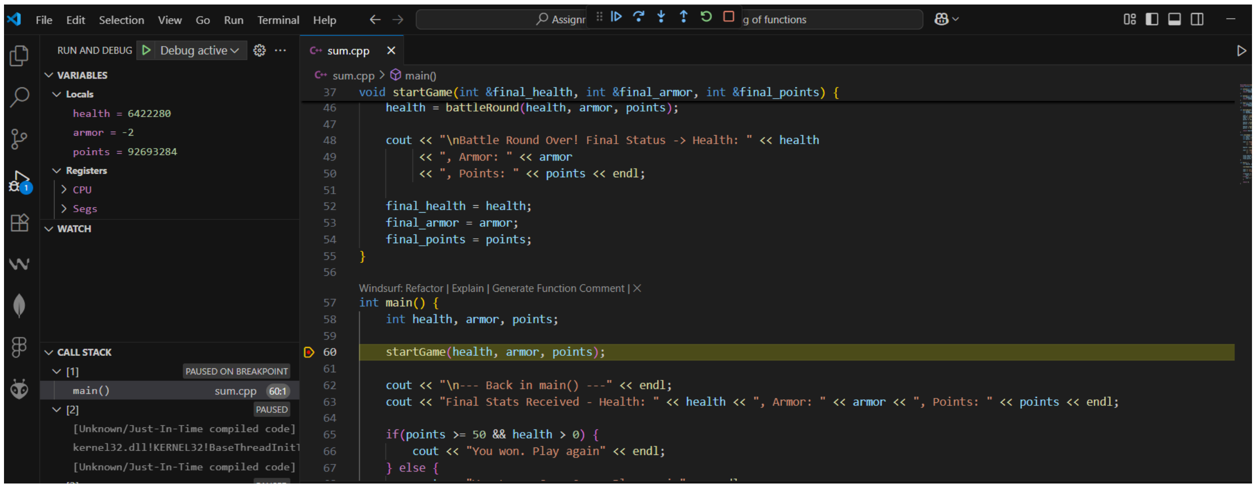
```
0 0  Debug active cpp with GDB (Assignment register_stack level Debugging of functions)   Ln 38, Col 1  Spaces: 4  UTF-8  CRLF  {} C++  Go Live  Windsurf: {...}  Win32
```

Explanation of Activations:

1) Main Function()

The program execution begins in the `main` function. The call stack contains only one activation record (frame #0) for `main`. This is the base of the stack, and all other function calls will be pushed on top of it.

As we can see the variables health, armor and points have been initialized but has garbage values from memory.



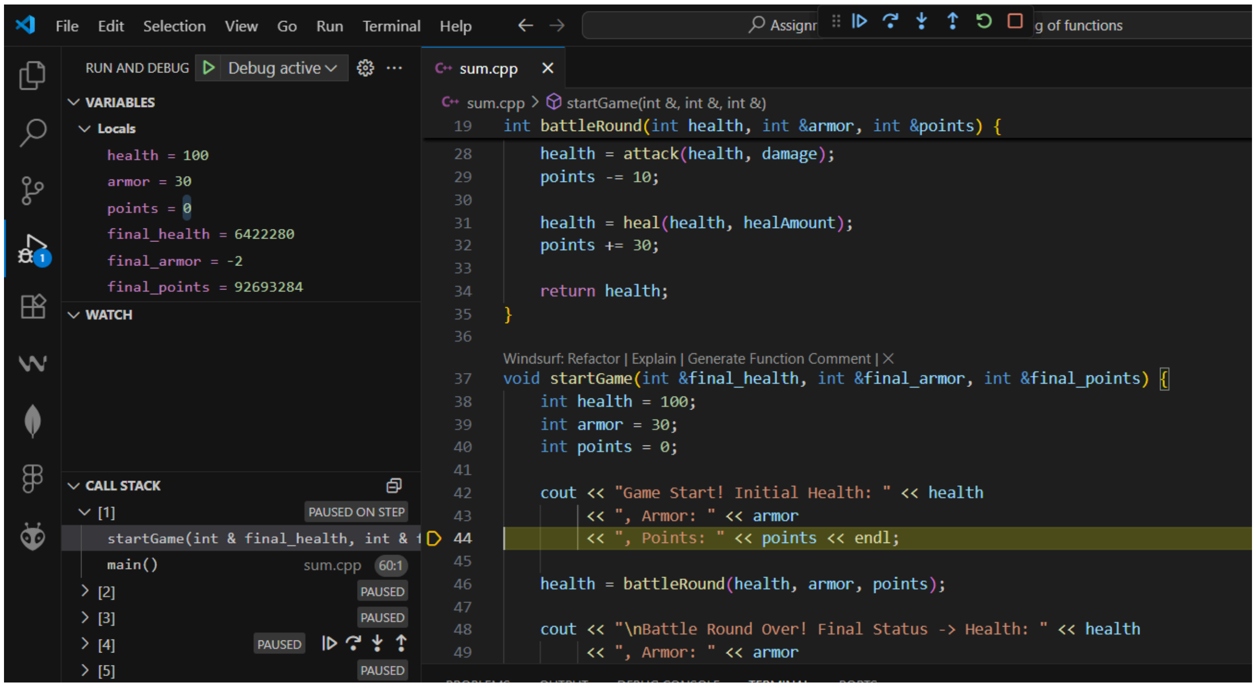
```
VARIABLES
  Locals
    health = 6422280
    armor = -2
    points = 92693284
  Registers
    > CPU
    > Segs
WATCH

CALL STACK
  [1]                                      PAUSED ON BREAKPOINT
    main()                                 sum.cpp 60:1
  [2]                                      PAUSED
    [Unknown/Just-In-Time compiled code]
    kernel32.dll!KERNEL32!BaseThreadInitT
    [Unknown/Just-In-Time compiled code]
```

```cpp
void startGame(int &final_health, int &final_armor, int &final_points) {
    health = battleRound(health, armor, points);

    cout << "\nBattle Round Over! Final Status -> Health: " << health
         << ", Armor: " << armor
         << ", Points: " << points << endl;

    final_health = health;
    final_armor = armor;
    final_points = points;
}

int main() {
    int health, armor, points;

    startGame(health, armor, points);

    cout << "\n--- Back in main() ---" << endl;
    cout << "Final Stats Received - Health: " << health << ", Armor: " << armor << ", Points: " << points << endl;

    if(points >= 50 && health > 0) {
        cout << "You won. Play again" << endl;
    } else {
```

2) startGame()

The `main` function calls `startGame`. A new activation record for `startGame` (frame #0) is pushed onto the top of the stack. The frame for `main` (now frame #1) is below it. The arguments for `startGame` are references, so GDB shows the memory addresses of the `health`, `armor`, and `points` variables in `main.`

`Now we can see that the variables health,armor and points have been assigned its proper values after running the assignment operation.`


File Edit Selection View Go Run Terminal Help
RUN AND DEBUG
Debug active
sum.cpp
sum.cpp > startGame(int &, int &, int &)
Windsurf: Refactor | Explain | Generate Function Comment | ×

VARIABLES
Locals

```
health = 100
armor = 30
points = 0
final_health = 6422280
final_armor = -2
final_points = 92693284
```

WATCH

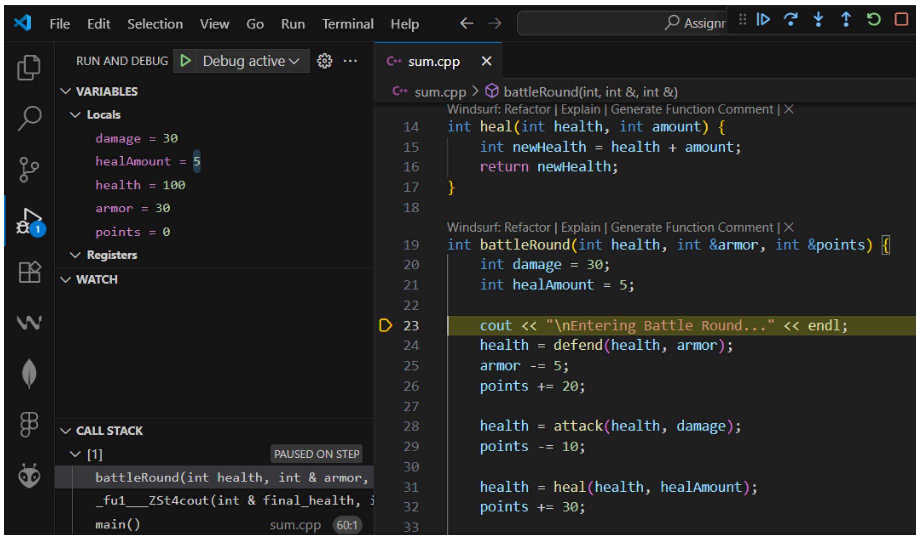
CALL STACK

```
[1]                                         PAUSED ON STEP
    startGame(int & final_health, int & f
    main()                           sum.cpp  60:1
[2]                                         PAUSED
[3]                                         PAUSED
[4]                                         PAUSED
[5]                                         PAUSED
```

```cpp
int battleRound(int health, int &armor, int &points) {
    health = attack(health, damage);
    points -= 10;

    health = heal(health, healAmount);
    points += 30;

    return health;
}

void startGame(int &final_health, int &final_armor, int &final_points) {
    int health = 100;
    int armor = 30;
    int points = 0;

    cout << "Game Start! Initial Health: " << health
         << ", Armor: " << armor
         << ", Points: " << points << endl;

    health = battleRound(health, armor, points);

    cout << "\nBattle Round Over! Final Status -> Health: " << health
         << ", Armor: " << armor
```

3) battleRound()

From `startGame`, `battleRound` is called. The stack grows again, with a new activation record for `battleRound` pushed on top. The chain has increased with three active frames. We can use `info locals` at this point to see the local variables `damage = 30` and `healAmount = 5` which exist only within this frame.


File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Debug active
sum.cpp

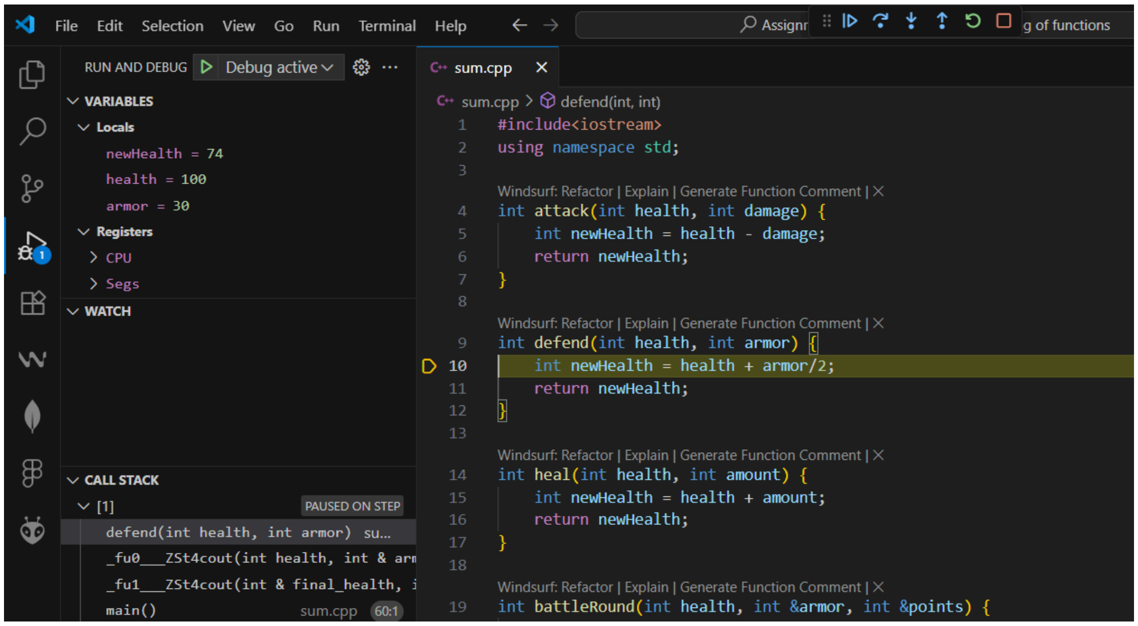
```
VARIABLES
  Locals
    damage = 30
    healAmount = 5
    health = 100
    armor = 30
    points = 0
  Registers
WATCH

CALL STACK
  [1]                                   PAUSED ON STEP
    battleRound(int health, int & armor,
    _fu1___ZSt4cout(int & final_health, i
    main()                      sum.cpp  60:1
```

```
sum.cpp > battleRound(int, int &, int &)
Windsurf: Refactor | Explain | Generate Function Comment | X
int heal(int health, int amount) {
    int newHealth = health + amount;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | X
int battleRound(int health, int &armor, int &points) {
    int damage = 30;
    int healAmount = 5;

    cout << "\nEntering Battle Round..." << endl;
    health = defend(health, armor);
    armor -= 5;
    points += 20;

    health = attack(health, damage);
    points -= 10;

    health = heal(health, healAmount);
    points += 30;
```

4) Defend()

From `battleRound`, `defend` is called. The stack grows again, with a new activation record for `battleRound` pushed on top.

Here the variables are yet to be updated.

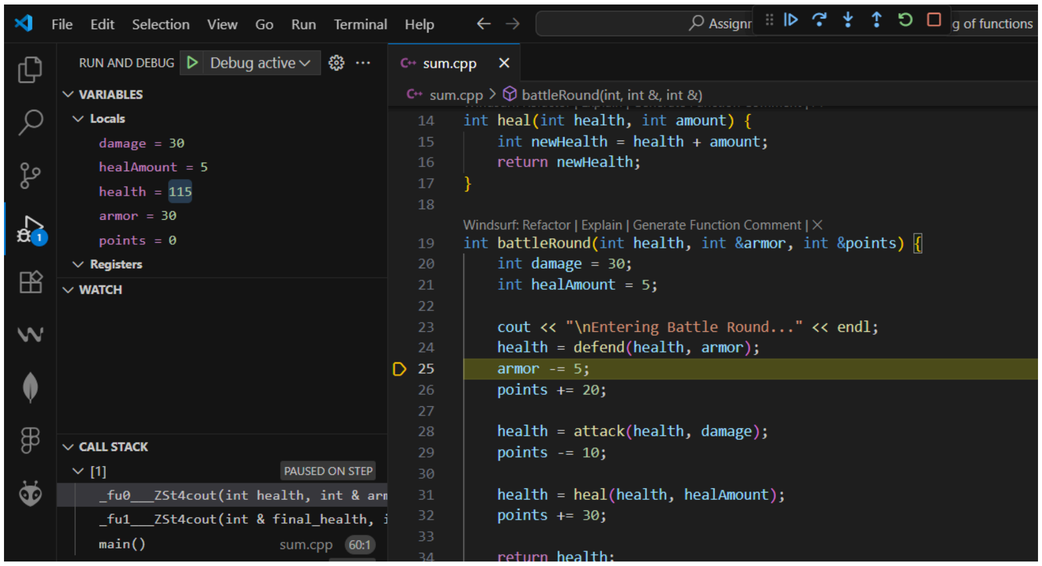

File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Debug active
sum.cpp

```
VARIABLES
  Locals
    newHealth = 74
    health = 100
    armor = 30
  Registers
    > CPU
    > Segs
WATCH

CALL STACK
  [1]                                   PAUSED ON STEP
    defend(int health, int armor)  su...
    _fu0___ZSt4cout(int health, int & arm
    _fu1___ZSt4cout(int & final_health, i
    main()                      sum.cpp  60:1
```

```
sum.cpp > defend(int, int)
#include<iostream>
using namespace std;

Windsurf: Refactor | Explain | Generate Function Comment | X
int attack(int health, int damage) {
    int newHealth = health - damage;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | X
int defend(int health, int armor) {
    int newHealth = health + armor/2;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | X
int heal(int health, int amount) {
    int newHealth = health + amount;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | X
int battleRound(int health, int &armor, int &points) {
```

5) Return from Defend()

Since the function Defend calls return the activation defend gets popped from the stack and returning to its previous instruction.

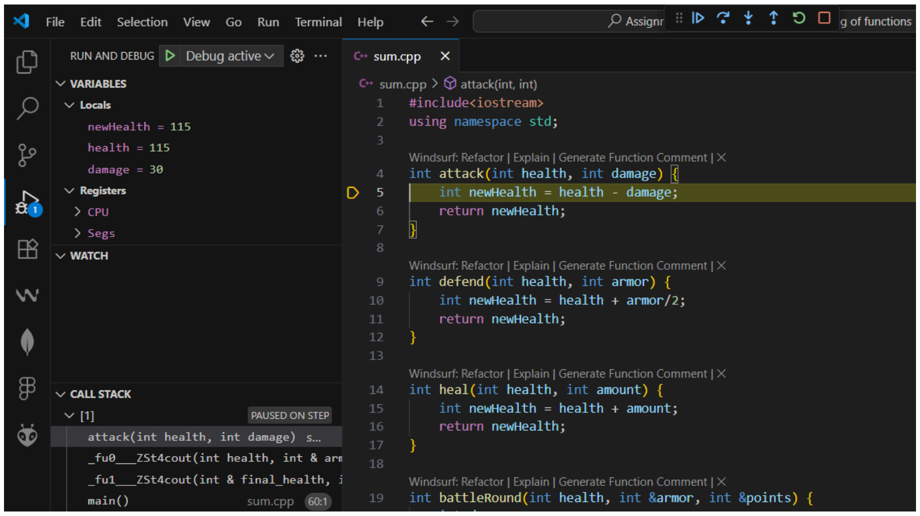

File Edit Selection View Go Run Terminal Help
Assignr
g of functions
RUN AND DEBUG
Debug active
sum.cpp
sum.cpp > battleRound(int, int &, int &)

```
VARIABLES
  Locals
    damage = 30
    healAmount = 5
    health = 115
    armor = 30
    points = 0
  Registers
WATCH

CALL STACK
  [1]                                   PAUSED ON STEP
    _fu0___ZSt4cout(int health, int & arm
    _fu1___ZSt4cout(int & final_health, i
    main()                         sum.cpp  60:1
```

```
int heal(int health, int amount) {
    int newHealth = health + amount;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
int battleRound(int health, int &armor, int &points) {
    int damage = 30;
    int healAmount = 5;

    cout << "\nEntering Battle Round..." << endl;
    health = defend(health, armor);
    armor -= 5;
    points += 20;

    health = attack(health, damage);
    points -= 10;

    health = heal(health, healAmount);
    points += 30;

    return health;
```

6) Attack()

Now from battleRound, defend was returned and a new function attack is called. The stack grows again, with a new activation record for battleRound pushed on top.

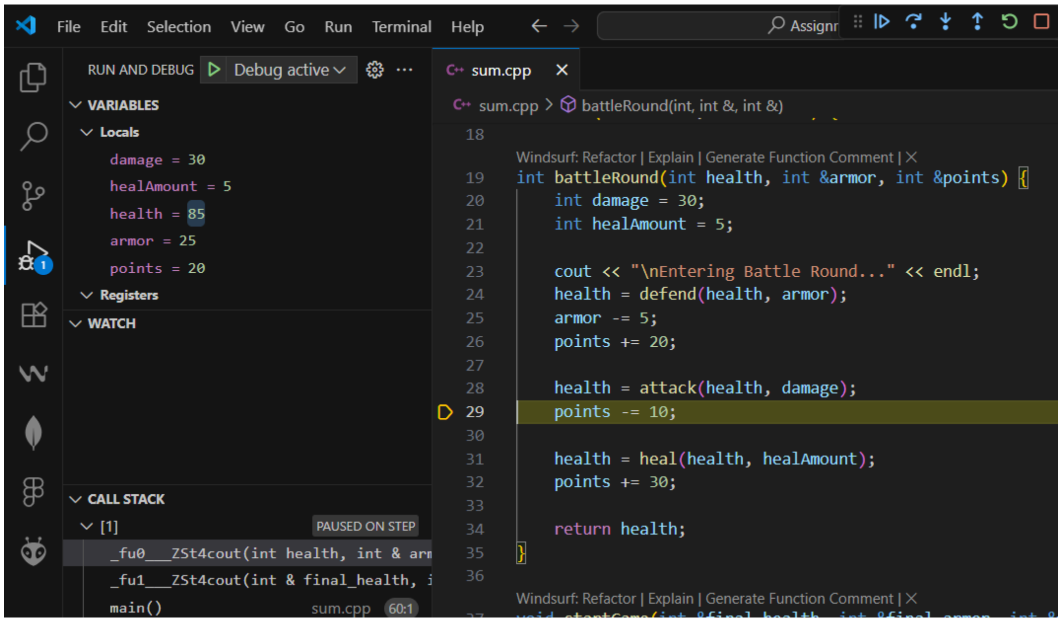

File Edit Selection View Go Run Terminal Help
Assignr
g of functions
RUN AND DEBUG
Debug active
sum.cpp
sum.cpp > attack(int, int)

```
VARIABLES
  Locals
    newHealth = 115
    health = 115
    damage = 30
  Registers
    > CPU
    > Segs
WATCH

CALL STACK
  [1]                                   PAUSED ON STEP
    attack(int health, int damage)  s...
    _fu0___ZSt4cout(int health, int & arm
    _fu1___ZSt4cout(int & final_health, i
    main()                         sum.cpp  60:1
```

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

Windsurf: Refactor | Explain | Generate Function Comment | ×
int attack(int health, int damage) {
    int newHealth = health - damage;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
int defend(int health, int armor) {
    int newHealth = health + armor/2;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
int heal(int health, int amount) {
    int newHealth = health + amount;
    return newHealth;
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
int battleRound(int health, int &armor, int &points) {
```

7) Return of attack()

Now the function attack got popped out from the stack as it returned.



VARIABLES
Locals

```
damage = 30
healAmount = 5
health = 85
armor = 25
points = 20
```

CALL STACK [1] PAUSED ON STEP

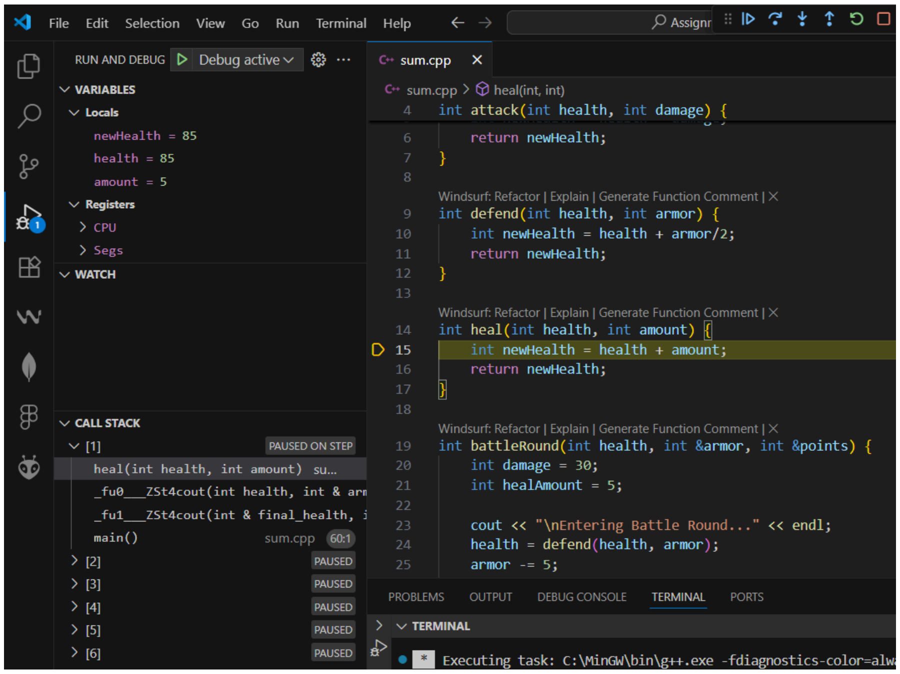
```
_fu0___ZSt4cout(int health, int & arm
_fu1___ZSt4cout(int & final_health, i
main()                    sum.cpp 60:1
```

sum.cpp > battleRound(int, int &, int &)

```cpp
int battleRound(int health, int &armor, int &points) {
    int damage = 30;
    int healAmount = 5;

    cout << "\nEntering Battle Round..." << endl;
    health = defend(health, armor);
    armor -= 5;
    points += 20;

    health = attack(health, damage);
    points -= 10;

    health = heal(health, healAmount);
    points += 30;

    return health;
}
```

8) Heal()

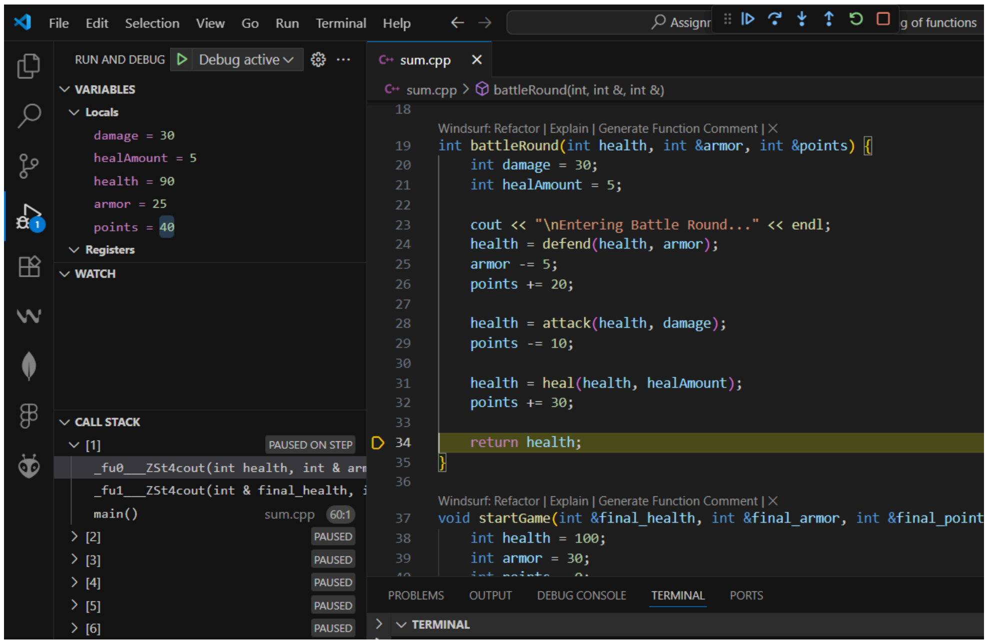
SImilar thing is happening in heal function also.



```
VARIABLES
  Locals
    newHealth = 85
    health = 85
    amount = 5
  Registers
    CPU
    Segs
WATCH
CALL STACK
  [1]                                   PAUSED ON STEP
    heal(int health, int amount)  su...
    _fu0___ZSt4cout(int health, int & arm
    _fu1___ZSt4cout(int & final_health, i
    main()                sum.cpp  60:1
  [2]  PAUSED
  [3]  PAUSED
  [4]  PAUSED
  [5]  PAUSED
  [6]  PAUSED
```

```cpp
int attack(int health, int damage) {
    return newHealth;
}

int defend(int health, int armor) {
    int newHealth = health + armor/2;
    return newHealth;
}

int heal(int health, int amount) {
    int newHealth = health + amount;
    return newHealth;
}

int battleRound(int health, int &armor, int &points) {
    int damage = 30;
    int healAmount = 5;

    cout << "\nEntering Battle Round..." << endl;
    health = defend(health, armor);
    armor -= 5;
```

```
Executing task: C:\MinGW\bin\g++.exe -fdiagnostics-color=alw
```

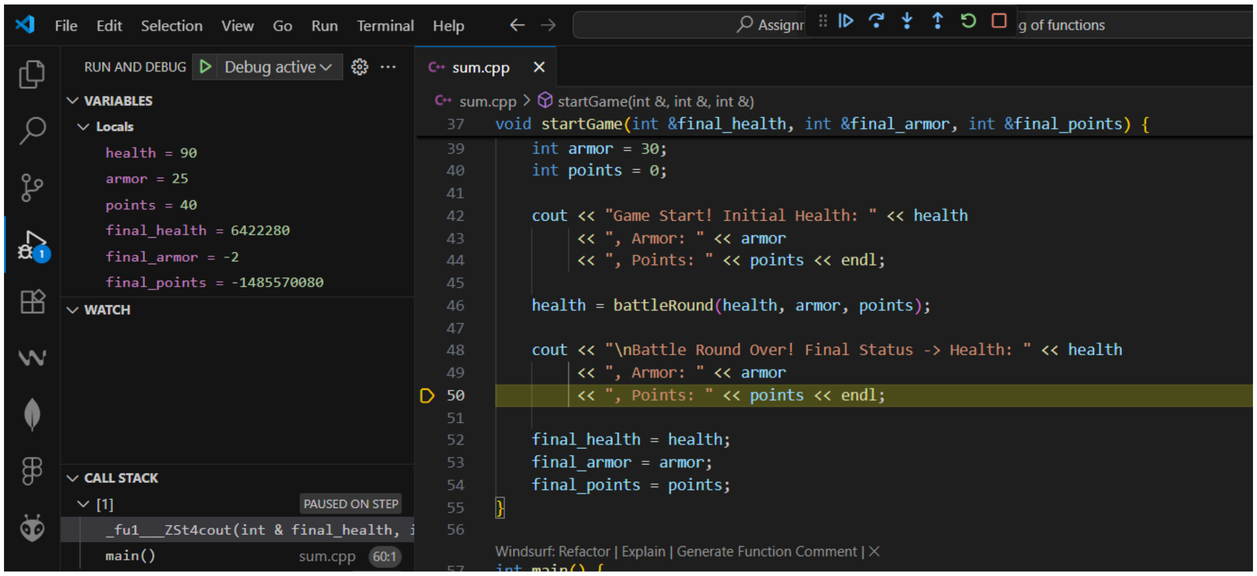


```
VARIABLES
  Locals
    damage = 30
    healAmount = 5
    health = 90
    armor = 25
    points = 40
  Registers
WATCH
CALL STACK
  [1]                                   PAUSED ON STEP
    _fu0___ZSt4cout(int health, int & arm
    _fu1___ZSt4cout(int & final_health, i
    main()                sum.cpp  60:1
  [2]  PAUSED
  [3]  PAUSED
  [4]  PAUSED
  [5]  PAUSED
  [6]  PAUSED
```

```cpp
int battleRound(int health, int &armor, int &points) {
    int damage = 30;
    int healAmount = 5;

    cout << "\nEntering Battle Round..." << endl;
    health = defend(health, armor);
    armor -= 5;
    points += 20;

    health = attack(health, damage);
    points -= 10;

    health = heal(health, healAmount);
    points += 30;

    return health;
}

void startGame(int &final_health, int &final_armor, int &final_point
    int health = 100;
    int armor = 30;
```

9)return of battlegound()

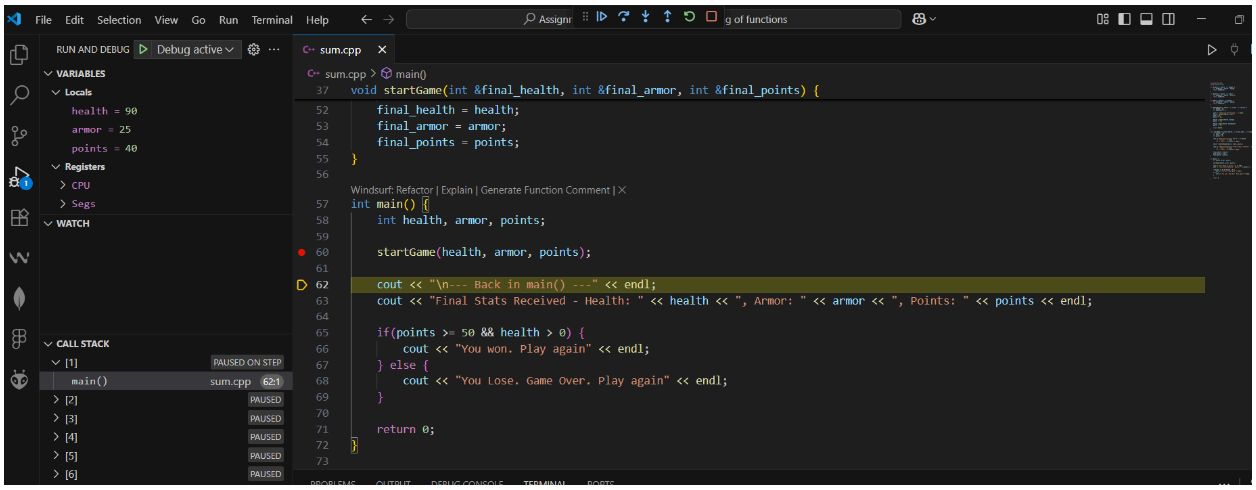
The `battleRound` function has completed its execution. Its activation record has been popped (destroyed) from the stack, and control has returned to the `startGame` function. The updated values for `health`, `armor`, and `points` (which were passed by reference) are now set within the `startGame` frame.



```
VARIABLES
  Locals
    health = 90
    armor = 25
    points = 40
    final_health = 6422280
    final_armor = -2
    final_points = -1485570080
WATCH
CALL STACK
  [1]                                         PAUSED ON STEP
    _fu1___ZSt4cout(int & final_health, i
    main()                                    sum.cpp  60:1
```

```cpp
void startGame(int &final_health, int &final_armor, int &final_points) {
    int armor = 30;
    int points = 0;

    cout << "Game Start! Initial Health: " << health
         << ", Armor: " << armor
         << ", Points: " << points << endl;

    health = battleRound(health, armor, points);

    cout << "\nBattle Round Over! Final Status -> Health: " << health
         << ", Armor: " << armor
         << ", Points: " << points << endl;

    final_health = health;
    final_armor = armor;
    final_points = points;
}

Windsurf: Refactor | Explain | Generate Function Comment | ✕
```

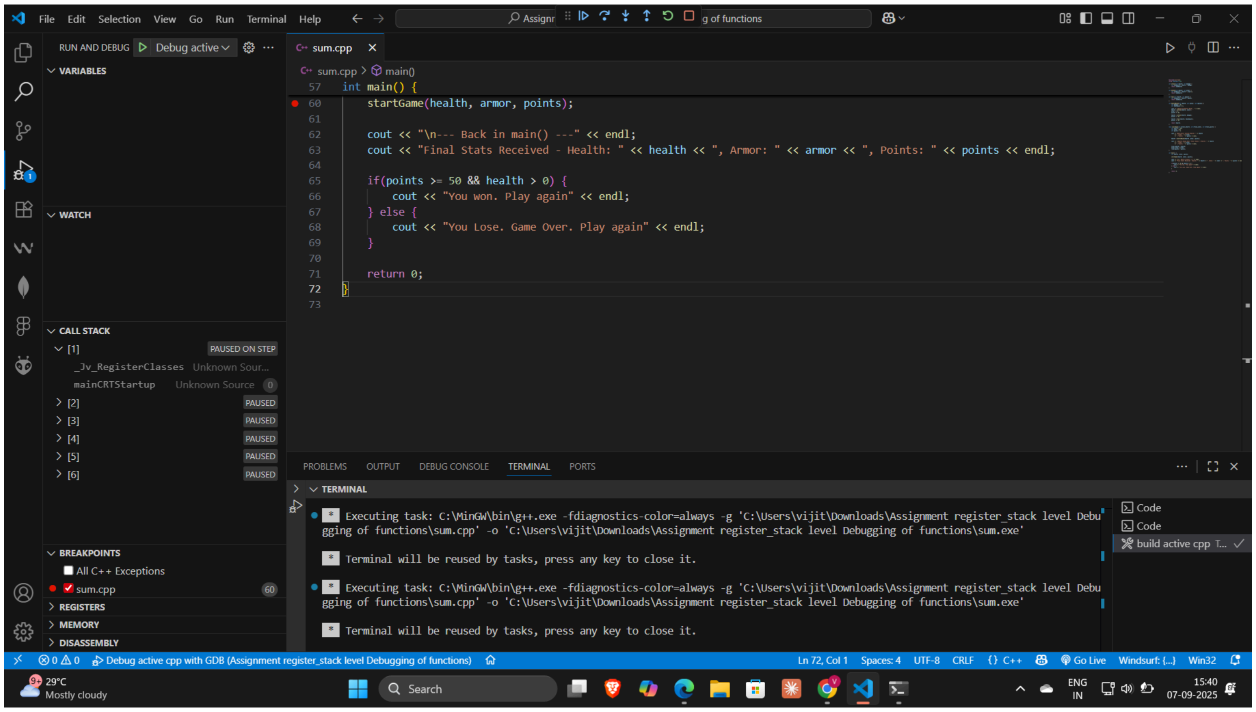
Return of startgame()



```
VARIABLES
  Locals
    health = 90
    armor = 25
    points = 40
  Registers
    CPU
    Segs
WATCH
CALL STACK
  [1]                     PAUSED ON STEP
    main()                sum.cpp  62:1
  [2]                     PAUSED
  [3]                     PAUSED
  [4]                     PAUSED
  [5]                     PAUSED
  [6]                     PAUSED
```

```cpp
void startGame(int &final_health, int &final_armor, int &final_points) {
    final_health = health;
    final_armor = armor;
    final_points = points;
}

Windsurf: Refactor | Explain | Generate Function Comment | ✕
int main() {
    int health, armor, points;

    startGame(health, armor, points);

    cout << "\n--- Back in main() ---" << endl;
    cout << "Final Stats Received - Health: " << health << ", Armor: " << armor << ", Points: " << points << endl;

    if(points >= 50 && health > 0) {
        cout << "You won. Play again" << endl;
    } else {
        cout << "You Lose. Game Over. Play again" << endl;
    }

    return 0;
}
```

Return of Main()



```
int main() {
    startGame(health, armor, points);

    cout << "\n--- Back in main() ---" << endl;
    cout << "Final Stats Received - Health: " << health << ", Armor: " << armor << ", Points: " << points << endl;

    if(points >= 50 && health > 0) {
        cout << "You won. Play again" << endl;
    } else {
        cout << "You Lose. Game Over. Play again" << endl;
    }

    return 0;
}
```

```
 *  Executing task: C:\MinGW\bin\g++.exe -fdiagnostics-color=always -g 'C:\Users\vijit\Downloads\Assignment register_stack level Debugging of functions\sum.cpp' -o 'C:\Users\vijit\Downloads\Assignment register_stack level Debugging of functions\sum.exe'

 *  Terminal will be reused by tasks, press any key to close it.

 *  Executing task: C:\MinGW\bin\g++.exe -fdiagnostics-color=always -g 'C:\Users\vijit\Downloads\Assignment register_stack level Debugging of functions\sum.cpp' -o 'C:\Users\vijit\Downloads\Assignment register_stack level Debugging of functions\sum.exe'

 *  Terminal will be reused by tasks, press any key to close it.
```

As the `main` function returns and pops off the stack, it outputs all the required values, and in the process, all activation records and local variables are destroyed, restoring memory to its initial state.