Help

```
#include <iostream>
#include <cstdlib>
#include <cstring>

using namespace std;

#include "
href../../../../common/math/ImportanceSampling_jl/src/MountainOption_h_src.pdfma
#include "
href../../../../common/math/ImportanceSampling_jl/src/Option_h_src.pdfmath/Impor

/*_____________ Atlas BaseOption _____________*/

AtlasOption::AtlasOption(const Param &P)
  : BaseOption(P)
{
  label = "atlas";

  P.extract("number best", nb_best);
  P.extract("number worst", nb_worst);
  final = pnl_vect_new();

}

AtlasOption::~AtlasOption()
{
  pnl_vect_free(&final);
}

void AtlasOption::print() const
{
  cout << "**** Atlas Option Characteristics *****" << endl;
  cout << " nb worst to remove : " << nb_worst << endl;
  cout << " nb best to remove : " << nb_best << endl;
  BaseOption::print();
}

double AtlasOption::payoff(const PnlMat *path_val)
```

```
{
  pnl_mat_get_row(final, path_val, path_val->m - 1);
  PnlVect S0 = pnl_vect_wrap_mat_row(path_val, 0);
  pnl_vect_div_vect_term(final, &S0);
  pnl_vect_qsort(final, 'i');
  double sum = 0.0;
  for (int i = nb_worst ; i < size - nb_best ; i++)
    sum += GET(final, i);

  return sum / (size - nb_best - nb_worst) * 1.05;
}

/*_____________ Altiplano BaseOption _____________*/

AltiplanoOption::AltiplanoOption(const Param &P)
  : BaseOption(P)
{
  label = "altiplano";

  P.extract("start window", start_window);
  P.extract("end window", end_window);
  P.extract("nominal", nominal);
  P.extract("barrier", barrier, size);

  if (start_window < 0 || end_window > maturity)
    {
      perror("inconsistency between window and maturity");
      exit(1);
    }

}


void AltiplanoOption::print() const
{
  cout << "**** Altiplano Option Characteristics ****" << endl;
  BaseOption::print();
  cout << " nominal : " << nominal << endl;
  cout << " barrier : " << barrier << endl;
  cout << " start window : " << start_window << endl;
  cout << " end window : " << end_window << endl;
```

```
}


double AltiplanoOption::payoff(const PnlMat *path_val)
{
  int nTimeSteps = path_val->m - 1;
  try
    {
      for (int i = (int) floor(start_window * nTimeSteps / maturity) ; i <= (int
        for (int j = 0 ; j < size ; j++)
          if (MGET(path_val, i, j) <= barrier * MGET(path_val, 0, j))
            throw "altiplano knocked out";

    }
  catch (const char *message)
    {
      /* the sample has been knocked out */
      if (strcmp(message, "altiplano knocked out") == 0)
        {
          double sum = 0.0;
          for (int j = 0 ; j < size ; j++)
            sum += MGET(path_val, nTimeSteps, j) / MGET(path_val, 0, j);

          sum /= size;
          return (1 + max(sum - 1.0, 0.0));
        }
    }
  catch (...)
    {
      perror("unknown exception");
    }

  /* the sample has NOT been knocked out */
  return nominal;

}
/*_____________ Everest BaseOption _________________*/

EverestOption::EverestOption(const Param &P)
  : BaseOption(P)
{
```

```
  label = "everest";
  final = pnl_vect_new();
}

EverestOption::~EverestOption()
{
  pnl_vect_free(&final);
}


void EverestOption::print() const
{
  cout << "**** Everest Option Characteristics ***" << endl;
  BaseOption::print();
}

double EverestOption::payoff(const PnlMat *path_val)
{
  pnl_mat_get_row(final, path_val, path_val->m - 1);
  PnlVect S0 = pnl_vect_wrap_mat_row(path_val, 0);
  pnl_vect_div_vect_term(final, &S0);
  double min_payoff = GET(final, 0);
  for (int i = 1 ; i < size ; i++)
    {
      if (GET(final, i) < min_payoff)
        min_payoff = GET(final, i);
    }

  return (1 + min_payoff);
}
```