Program jest symulacją turnieju rozgrywanego między 4096 zawodnikami, walki toczą się na śmierć i życie, przedstawie tutaj opis funkcji, klas i zmniennych w programie.

class BodyPart - klasa reprezentująca kończyny zawodików.

int health,block, damage - losowe zmienne reprezentujące wartości poszczególnych atrybutów kończyny.

bool Active - zmienna mowiąca nam czy kończyna jest aktywna czy też została już utracona w walce.

void receiveDamage(int dmg) - prosta funkcja zmniejszająca liczbę hp kończyny w wyniku otrzymania obrażeń.

poza tym klasa zawiera zestaw funkcji get dla odowiednich atrybutów, np. int getHealth(), umożliwiających sprawdzenie stanu danej zmiennej.

class Fighter : public BodyPart - klasa reprezentująca pojedynczego zawodnika.

BodyPart Body[5] - tablica zawierająca poszczególne częsci ciała zawodnika.

int bleedLimit -losowa zmienna reprezentująca ilość krwi jaka zawodnik może stracić

int bleedCounter - licznik utraconej krwi

bool Alive - stan zawodnika, true oznacza żywego, false martwego.

void checkStatusFighter() - funkcja aktualizująca stan zawodnika, zmeinia wartość przypisaną bool Alive na false jeśli: zawodnik stracił głowę, lub stracił za dużo krwi.

int Bleeding() - funckcja obrazująca krawawienie, licznik krawaienia rośnie tym szybciej im więcej kończyn zawodnik stracił.

bool getStatusFighter() - standardowa funkcja zwracająca stan zawodnika.

class Battle : public Fighter - klasa zawierająca w sobie funkcje walki zawodników.

static void Attack(Fighter* Warrior1, Fighter* Warrior2) - funckja pojedyneczego ataku,zawodnika **Warrior1** na zawodnika **Warrior2**, **int attackSource = rand() % 5** wylosowana zostaje częsc ciała **Warrior1** która przeprowadza atak (Jeśli jest niekatyna za pomocą pętli **while** losujemy aż do skutku), następnie w analogiczny sposób zostaje wybrany cel ataku na ciele zawodnika **Warrior2**, **int target = rand() % 5**. Następnie funckja przechodzi do wyliczania skuteczności ataku bazując na wartości bloku atakowanej części ciała, jeśli blok się powiedzie zostanie wyprowadzona kontra:

if (Warrior2->Body[target].getBlock() > rand() % 5 + 1)

Warrior1->Body[attackSource].receiveDamage(Warrior1->Body[attackSource].getDamage());

jeśli się nie powiedzie zależnie od wyniku porównania zostanie zadana połowa

(**if (Warrior2->Body[target].getBlock() == rand() % 5 + 1)**)

lub pełne obrażenia celowi

(**if (Warrior2->Body[target].getBlock() < rand() % 5 + 1)**).

static void Match(Fighter* Warrior1, Fighter* Warrior2) - funckja pojedynczej walki,

na początku walki **Warrior1** otrzymuje obrażenia od **Bleeding()**, następnie sprawdzamy czy przeżył, jeśli tak to wywołujemy funckje **Attack(Warrior1, Warrior2)**, nastepnie ponownie sprawdzamy stan zawodników, po czym **Warrior2** trzymuje obrażenia od **Bleeding()**, następnie wywołujemy funkcje **Attack(Warrior2, Warrior1)**, pętlaulegnie przerwaniu, gdy jeden z zawodników umrze, **Alive=false**.

class Tournament : public Battle - klasa zierająca funkcję turnieju.

static Fighter StartFight64(Fighter Duelists[]) - funkcja turnieju przyjmująca za argument tablicę uczestników.

for (int k = 0; k < 6; k++) - pętla działa tak długo aż z 64 zawodników zostanie 1

(64/2=>32/2=>16/2=>8/2=>4/2=>2/2=>1)

int i = 0 licznik elementów tablicy ustawiony na pierwszy element od lewej

int j = N - 1 licznik elementów tablicy ustawiony na pierwszy element od prawej

szukamy pierwszego żywego zawodnika od lewej strony:

```
while (true)
	{
		while (true)
		{
			if (Duelists[i].getStatusFighter())
				break;
			else
				i++;
				if (i >= j)
				break;
	}
```

następnie analogicznie szukamy pierwszego żywego zawodnika od prawej strony.

po znaleznieniu dwóch żywych zawodników:

Match(&(Duelists[i]), &(Duelists[j])) walczą oni ze sobą.

po 6 turach (zredukowaniu stanu tablicy do jednego ostatniego żywego zawodnika)

skanujemy tablicę ponownie, zwracając zwycięzce.

return Duelists[t];