Turbo ParTool

By Atlas

8 luty 2026

1. Export pliku .par za pomocą PARex

PARex jest to program służący do eksportu pliku `.par` z **KnightShift** do postaci katalogu z plikami `.cpp`, które reprezentują każdą sekcję obiektów pliku `.par`.

Jak używać?:

1. Włączamy program.
2. Wpisujemy nazwę pliku wejściowego wraz z formatem.
3. Klikamy przycisk enter i czekamy.
4. Po wyłączeniu okna konsoli wiemy, że program skończył pracę.

Program działa również w trybie ARGC&ARGV. Przykład użycia:

```
PARex.exe <nazwa_pliku.par>
```

Szybki Export:

Aby w szybki sposób dekompilować plik `.par` należy korzystać z `EXPORT.ps1` lub `EXPORT.bat`. W celu modyfikacji argumentów/konfiguracji należy zedytować plik `.bat/.ps1` przez notatnik.

Przkład pliku .bat:

```
PARex.exe <nazwapliku.par>
```

Przykład pliku .ps1:

```
.\PARex.exe .\<nazwapliku.par>
```

2. Opcjonalna konfiguracja PARex

Plik **PARex.cfg** umożliwia konfigurację sposobu eksportu danych do plików `.cpp`. Użytkownik może zdecydować, czy wybrane wartości będą prezentowane jako pojedyncze bity (flagi) o określonych etykietach, czy w formacie surowym. W przypadku wyłączenia mapowania na flagi, dane zostaną zapisane jako typ `uint32_t`. Rezygnacja z flagowania pozwala na znaczące przyspieszenie procesu eksportu oraz skrócenie czasu kompilacji wygenerowanego kodu. W pliku konfiguracyjnym dostępne są następujące parametry:

Prawda:

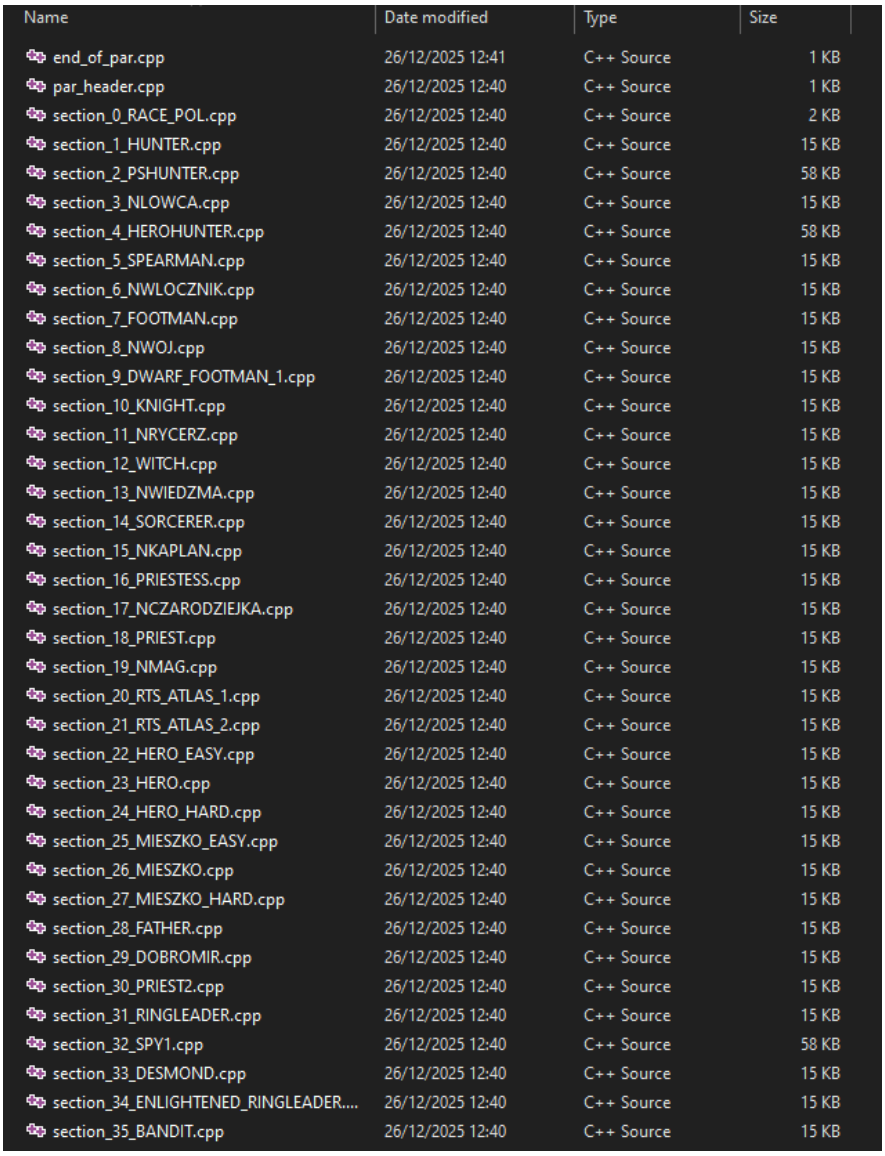
- YES/Yes/yes
- TRUE/True/true
- 1

Fałsz:

- NO/No/no
- FALSE/False/false
- 0

3. Ogólny pogląd na strukturę otrzymanych plików po eksporcie/dekompilacji

Po eksporcie pliku `.par`, otrzymamy katalog w którym znajdują się pliki `.cpp` reprezentujące sekcje z obiektami. Składnia plików jest w języku `C++`. Oprócz plików `.cpp` z sekcjami - otrzymujemy również plik `section_order.txt`, w którym znajduje się kolejność kompilacji plików sekcji.

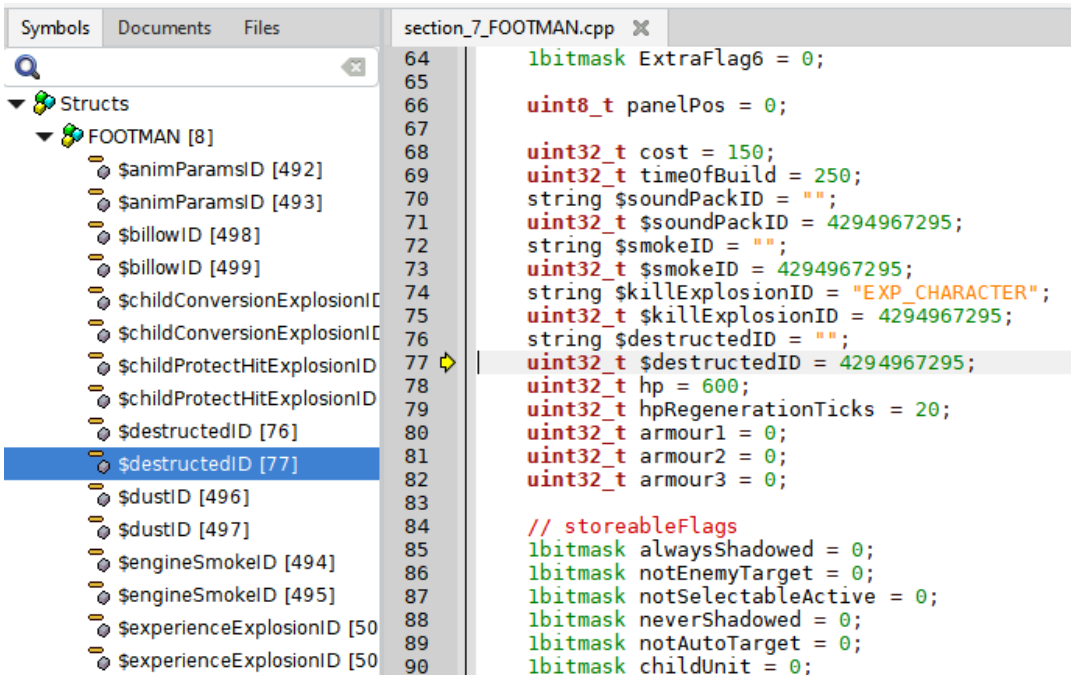


Name	Date modified	Type	Size
end_of_par.cpp	26/12/2025 12:41	C++ Source	1 KB
par_header.cpp	26/12/2025 12:40	C++ Source	1 KB
section_0_RACE_POL.cpp	26/12/2025 12:40	C++ Source	2 KB
section_1_HUNTER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_2_PSHUNTER.cpp	26/12/2025 12:40	C++ Source	58 KB
section_3_NLOWCA.cpp	26/12/2025 12:40	C++ Source	15 KB
section_4_HEROHUNTER.cpp	26/12/2025 12:40	C++ Source	58 KB
section_5_SPEARMAN.cpp	26/12/2025 12:40	C++ Source	15 KB
section_6_NWLOCZNIK.cpp	26/12/2025 12:40	C++ Source	15 KB
section_7_FOOTMAN.cpp	26/12/2025 12:40	C++ Source	15 KB
section_8_NWOJ.cpp	26/12/2025 12:40	C++ Source	15 KB
section_9_DWARF_FOOTMAN_1.cpp	26/12/2025 12:40	C++ Source	15 KB
section_10_KNIGHT.cpp	26/12/2025 12:40	C++ Source	15 KB
section_11_NRYCERZ.cpp	26/12/2025 12:40	C++ Source	15 KB
section_12_WITCH.cpp	26/12/2025 12:40	C++ Source	15 KB
section_13_NWIEDZMA.cpp	26/12/2025 12:40	C++ Source	15 KB
section_14_SORCERER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_15_NKAPLAN.cpp	26/12/2025 12:40	C++ Source	15 KB
section_16_PRIESTESS.cpp	26/12/2025 12:40	C++ Source	15 KB
section_17_NCZARODZIEJKA.cpp	26/12/2025 12:40	C++ Source	15 KB
section_18_PRIEST.cpp	26/12/2025 12:40	C++ Source	15 KB
section_19_NMAG.cpp	26/12/2025 12:40	C++ Source	15 KB
section_20_RTS_ATLAS_1.cpp	26/12/2025 12:40	C++ Source	15 KB
section_21_RTS_ATLAS_2.cpp	26/12/2025 12:40	C++ Source	15 KB
section_22_HERO_EASY.cpp	26/12/2025 12:40	C++ Source	15 KB
section_23_HERO.cpp	26/12/2025 12:40	C++ Source	15 KB
section_24_HERO_HARD.cpp	26/12/2025 12:40	C++ Source	15 KB
section_25_MIESZKO_EASY.cpp	26/12/2025 12:40	C++ Source	15 KB
section_26_MIESZKO.cpp	26/12/2025 12:40	C++ Source	15 KB
section_27_MIESZKO_HARD.cpp	26/12/2025 12:40	C++ Source	15 KB
section_28_FATHER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_29_DOBROMIR.cpp	26/12/2025 12:40	C++ Source	15 KB
section_30_PRIEST2.cpp	26/12/2025 12:40	C++ Source	15 KB
section_31_RINGLEADER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_32_SPY1.cpp	26/12/2025 12:40	C++ Source	58 KB
section_33_DESMOND.cpp	26/12/2025 12:40	C++ Source	15 KB
section_34_ENLIGHTENED_RINGLEADER....	26/12/2025 12:40	C++ Source	15 KB
section_35_BANDIT.cpp	26/12/2025 12:40	C++ Source	15 KB

Rysunek 1: Lista plików .cpp po eksporcie.

4. Polecane IDE do edycji plików .cpp

Polecanym programem do edycji plików `.cpp` jest **geany**. Prócz możliwości edycji kodu źródłowego, posiada również bardzo intuicyjny podgląd na struktury danych. Podgląd na strukturę danych reprezentowany jest przez drzewo, którego liście reprezentują odpowiednie pola w strukturze danych. Kliknięcie dwa razy na nazwę pola - przenosi użytkownika do linii z wybranym polem.



Rysunek 2: Panel boczny *Symbole* wyświetlający listę zmiennych wraz z odnośnikami do konkretnych linii kodu.

5. Komentarze

W plikach sekcji można pisać komentarze za pomocą symboli `//`. Przykład użycia:

```
...
uint32_t armour2 = 0;
uint32_t armour3 = 0;

// storeableFlags                <-- Ta linijka zostanie pominieta.
// ala ma kota                   <-- Ta linijka tez zostanie pominieta.
// KOD NAPISANY PO SLOWIANSKU    <-- Ta tez.
1bitmask alwaysShadowed = 0;
...
```

6. Typy danych

1. **bool** - zajmuje dosłownie jeden bajt i może przechowywać tylko i wyłącznie wartości logiczne: **false, 0, true, 1**.

   Przykład użycia:

   ```
   bool logic = 0;
   bool logic2 = true;
   ```

2. **char** - zajmuje dosłownie jeden bajt i może przechowywać tylko i wyłącznie wartość w postaci symbolu.

   Przykład użycia:

   ```
   char symbol = 'a';
   ```

3. **uint8_t** - zajmuje dosłownie jeden bajt i może przechowywać tylko i wyłącznie wartość w postaci liczby nieujemnej z zakresu **od 0 do 255**.

   Przykład użycia:

   ```
   uint8_t format_0 = 80;
   ```

4. **uint16_t** - zajmuje dosłownie dwa bajty i może przechowywać tylko i wyłącznie wartość w postaci liczby nieujemnej z zakresu **od 0 do 65 535**.

   Przykład użycia:

   ```
   uint16_t panelPos = 2;
   ```

5. **uint32_t** - zajmuje dosłownie cztery bajty i może przechowywać tylko i wyłącznie wartość w postaci liczby nieujemnej z zakresu **od 0 do 4 294 967 295**.

   Przykład użycia:

   ```
   uint32_t listTemplateNum = 340;
   ```

6. **uint64_t** - zajmuje dosłownie osiem bajtów i może przechowywać tylko i wyłącznie wartość w postaci liczby nieujemnej z zakresu **od 0 do 18 446 744 073 709 551 615**.

   Przykład użycia:

   ```
   uint64_t number_of_sections = 1280;
   ```

7. **int8_t** - zajmuje dosłownie jeden bajt i może przechowywać tylko i wyłącznie wartość w postaci liczby z zakresu od **-128 do 127**.

   Przykład użycia:

   ```
   int8_t test = -13;
   ```

8. **int16_t** - zajmuje dosłownie dwa bajty i może przechowywać tylko i wyłącznie wartość w postaci liczby z zakresu od –**32 768 do 32 767**.

   Przykład użycia:

```
int16_t test2 = -1236;
```

9. **int32_t** - zajmuje dosłownie cztery bajty i może przechowywać tylko i wyłącznie wartość w postaci liczby z zakresu od **−2 147 483 648 do 2 147 483 648**.

   Przykład użycia:

   ```
   int32_t mapSign = -1;
   ```

10. **int64_t** - zajmuje dosłownie osiem bajtów i może przechowywać tylko i wyłącznie wartość w postaci liczby z zakresu od **−9 223 372 036 854 775 808 do 9 223 372 036 854 775 807**.

    Przykład użycia:

    ```
    int64_t num = -221280;
    ```

11. **bool[]** - tablica wartości typu bool przechowująca **n** wartości typu bool. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    bool classID_array[] = {0,1,1,1,1,0,0,0,0,0,0,0,0};
    ```

12. **char[]** - tablica symboli zajmująca **n** bajtów i może przechowywać tylko i wyłącznie napis w cudzysłowiach. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    char header_string[header_string_length] = "translateGameParams";
    ```

13. **uint8_t[]** - tablica wartości typu `uint8_t` przechowująca **n** wartości typu `uint8_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    uint8_t test_array[] = {64,1,3,1,23,0,128};
    ```

14. **uint16_t[]** - tablica wartości typu `uint16_t` przechowująca **n** wartości typu `uint16_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    uint16_t test_array_2[] = {64,503,346,22};
    ```

15. **uint32_t[]** - tablica wartości typu `uint32_t` przechowująca **n** wartości typu `uint32_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    uint32_t test_array_3[] = {2264,503,55346,322,0};
    ```

16. **uint64_t[]** - tablica wartości typu `uint64_t` przechowująca **n** wartości typu `uint64_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    uint64_t test_array_4[] = {226433,5034};
    ```

17. **int8_t[]** - tablica wartości typu `int8_t` przechowująca **n** wartości typu `int8_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    int8_t test_array[] = {-64,1,3,-1,-23,0};
    ```

18. **int16_t[]** - tablica wartości typu `int16_t` przechowująca **n** wartości typu `int16_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    int16_t test_array_2[] = {-64,503,-346,22};
    ```

19. **int32_t[]** - tablica wartości typu `int32_t` przechowująca **n** wartości typu `int32_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    int32_t test_array_3[] = {-2264,503,55346,-322,0};
    ```

20. **int64_t[]** - tablica wartości typu `int64_t` przechowująca **n** wartości typu `int64_t`. Zapis rozmiaru talbicy w nawiasach `[]` nie jest wymagany ponieważ kompilator i tak sam sprawdzi jej rozmiar. Wymagane jest aby nawiasy kwadratowe się pojawiły w celu podkreślenia typu będącego tablicą.

    Przykład użycia:

    ```
    int64_t test_array_4[] = {-2264,503,55346,-322,0};
    ```

21. **string** - łańcuch znaków zajmujący maksymalnie **32 767 bajtów** i może przechowywać tylko i wyłącznie napis w cudzysłowiach. Kompilator automatycznie oblicza rozmiar takiego łańcucha znaków i zapisuje go na czterech bajtach przed tym łańcuchem w pliku binarnym.

    Przykład użycia:

    ```
    string mesh = "CHARACTERS\FOOTMAN#nfootman.tex";
    ```

22. **GUID** - identyfikator globalnie unikatowy czyli identyfikator pewnego obiektu w systemie Windows lub wszędzie, gdzie potrzebny jest unikatowy identyfikator. Zajmuje dokładnie **16 bajtów**. W grze KnightShift pojawia sie w wielu różnych plikach binarnych np. w mapach (LND/MIS), w meshach (MSH) lub w pliku parametrów (PAR).

    Przykład:

    ```
    GUID header_guid = {FF863ECE-6C69-468A-B0BC-D0244A73316C};
    ```

23. **struct** - struktura czyli pewna przestrzeń, w której mogą znaleźć się pewne wartości różnych typów danych. Stosujemy struct wtedy gdy tworzymy nowy obiekt w sekcji lub wtedy gdy tworzymy nowy nagłówek sekcji. Kompilator liczy ilość wystąpień słowa struct w pliku w celu obliczenia ilości obiektów w pojedynczej sekcji.

    Przykład użycia:

    ```
    struct Section_0_RACE_POL_Header
    {
    ```

```
 32bituniquekey race = POL;
 uint32_t flag_0 = 10;
 uint32_t number_of_objects = count();
};
```

24. **1bitmask** - zajmuje dosłownie jeden bit i może przechowywać tylko i wyłącznie wartość w postaci wartości logicznej **0 albo 1**. Ten typ danych wykorzystywany jest do reprezentacji flag obiektów. Osiem wartości 1bitmask tworzy jeden bajt.

    Przykład użycia:

```
// shadowType

// pierwszy bajt
1bitmask _shadowSimple = 0;
1bitmask _shadowStatic = 0;
1bitmask _shadowAnimated = 1;
1bitmask _shadowVertical = 0;
1bitmask unknown1 = 0;
1bitmask unknown2 = 0;
1bitmask unknown3 = 0;
1bitmask unknown4 = 0;

// drugi bajt
1bitmask _moveWind = 0;
1bitmask _noReflect = 0;
1bitmask unknown5 = 0;
1bitmask _hasNotBigEnoughBox = 0;
1bitmask _noSubObjectShadow = 0;
1bitmask _symmetricShadow = 0;
1bitmask unknown6 = 0;
1bitmask unknown7 = 0;

// trzeci bajt
1bitmask _shadowOldStatic = 0;
1bitmask _shadowOldRound = 1;
1bitmask _shadowOldEllipse1 = 0;
1bitmask _shadowOldEllipse2 = 0;
1bitmask _shadowOldEllipse3 = 0;
1bitmask _shadowOldTree = 0;
1bitmask _hasOldSoftwareRendering = 0;
1bitmask ExtraFlag0 = 0;

// czwarty bajt
uint8_t shadowType = 0;
```

Inny przykład:

```
// third byte of panelPos
1bitmask tab1 = 0;
1bitmask ExtraFlag0 = 0;
1bitmask ExtraFlag1 = 0;
1bitmask ExtraFlag2 = 0;
1bitmask ExtraFlag3 = 0;
1bitmask ExtraFlag4 = 0;
1bitmask ExtraFlag5 = 0;
1bitmask ExtraFlag6 = 0;
```

25. **8bitmask** - Zajmuje 8 bitów czyli jeden bajt. Wprowadzając wartość do tego typu danych wpisujemy stałe, których wartości przedstawiono w ściągawce cheat sheet. Na wartościach tych można dokonywać operacji OR (suma logiczna), za pomocą której można łączyć flagi. `8bitmask` wykorzystywany jest wyłącznie w reprezentacji wartości o etykiecie **passiveMask**.

    Przykład użycia:

    ```
    // first byte of passiveMask
    8bitmask passiveMask = (_artefactPassive_ | _mapOtherPassive_);
    ```

26. **32bitmask** - Zajmuje 32 bity czyli cztery bajty. Wprowadzając wartość do tego typu danych wpisujemy stałe, których wartości przedstawiono w ściągawce cheat sheet. Na wartościach tych można dokonywać operacji OR (suma logiczna), za pomocą której można łączyć flagi.

    Przykład użycia:

    ```
    // magicAnimType
    32bitmask magicAnimType =
     (equipmentAnimTypeMagic2 | magicTalk2 | magicTalkNone);
    ```

27. **32bituniquekey** - Zajmuje 32 bity czyli cztery bajty. Do tego typu danych wprowadzamy wyłącznie jedną stałą. Ten typ danych obsługuje jedynie stałe przez co nie można przypisać mu wartości liczbowej w sposób bezpośredni. Jeżeli chcemy skorzystać z wartości liczbowej, to typ należy zmienić na `uint32_t` lub `int32_t`. Stałe, które można wprowadzić do tego typu przedstawiono w ściągawce cheat sheet.

    Przykład użycia:

    ```
    32bituniquekey standType = standAccurate;
    ```

7. Funkcje

Kompilator ParIm obsługuje również funkcje, które przedstawiono poniżej:

- **uint32_t count();** - funkcja, która zlicza ilość obiektów w sekcji. Funkcja działa w taki sposób, że liczy ilość wystąpień struct w pliku sekcji, a następnie na podstawie tej informacji oblicza i zwraca ilość obiektów w sekcji. Funkcja zadziała w sposób prawidłowy jeśli pole ma nazwę `number_of_objects`.

  Przykład użycia:

  ```
  uint32_t number_of_objects = count();
  ```

- **uint64_t count();** - funkcja, która zlicza ilość wszystkich sekcji. Funkcja działa w taki sposób, że liczy ilość wystąpień plików w pliku order, a następnie na podstawie tej informacji oblicza i zwraca całkowitą ilość sekcji. Funkcja zadziała w sposób prawidłowy jeśli pole ma nazwę `number_of_sections`.

  Przykład użycia:

  ```
  uint64_t number_of_sections = count();
  ```

8. Wartości powiązane z typami

Dla niektórych typów danych możemy wyróżnić stałe, które kojarzone są z odpowiednimi wartościami.

8.1. 8bitmask

1. passiveMask

Nazwa stałej	Wartość w HEX
mapOtherPassive	0x00
bridgePassive	0x01
pontoonBridgePassive	0x02
singleBridgePassive	0x03
singlePontoonBridgePassive	0x04
bridgeRuinPassive	0x05
pontoonBridgeRuinPassive	0x06
artefactPassive	0x07
tunnelEntrancePassive	0x08
healthPlacePassive	0x09
conversionPlacePassive	0x0A
teleportPassive	0x0B
birdPassive	0x0C
waterAnimalPassive	0x0D
mapNothingPassive	0x10
mapBuildingPassive	0x20
mapRockPassive	0x30
mapTreePassive	0x40
mapWallPassive	0x50
mapEditorPassive	0x60

8.2. 32bitmask

1. objectType

Nazwa stałej	Wartość w HEX
animNone	0x00000000
animWalk	0x00000001
animRotor	0x00000002
animRotation	0x00000002
moveLand	0x00000000
moveAmphibia	0x00000100
moveShip	0x00000200
moveFlyable	0x00000300
notMoveable	0x00010000

2. magicAnimType

Nazwa stałej	Wartość w HEX
equipmentAnimTypeNone	0x00000000
equipmentAnimTypeFight	0x00000001
equipmentAnimTypeFight2	0x00000002
equipmentAnimTypeShoot	0x00000003
equipmentAnimTypeMagic1	0x00000004
equipmentAnimTypeMagic2	0x00000005
equipmentAnimTypeMagic3	0x00000006
equipmentAnimTypeMagic4	0x00000007
magicTalkNone	0x00000000
magicTalk1	0x00010000
magicTalk2	0x00020000
magicTalk3	0x00030000
magicTalk4	0x00040000

8.3. 32bituniquekey

1. magicType

Nazwa stałej	Wartość w HEX
NULL	0x00000000
magicImmortalShield	0x00000001
magicFreeze	0x00000002
magicCapturing	0x00000003
magicStorm	0x00000004
magicSeeing	0x00000005
magicConversion	0x00000006
magicFireRain	0x00000007
magicRemoveStormFireRain	0x00000008
magicTeleportation	0x00000009
magicGhost	0x0000000A
magicWildAnimal	0x0000000B
magicTrap	0x0000000C
magicGetHP	0x0000000D
magicSingleFreeze	0x0000000E
magicBlindAttack	0x0000000F
magicTimedCapturing	0x00000010
magicOurWildAnimal	0x00000011
magicOurHoldWildAnimal	0x00000012
magicOurMagicMirror	0x00000013
magicRandConversion	0x00000014
magicAroundDamage	0x00000015
magicSelfHealing	0x00000016
magicFireWall	0x00000017

2. buildingType

Nazwa stałej	**Wartość w HEX**
buildingNormal	0x00000000
buildingFactory	0x00000001
buildingHarvestFactory	0x00000002
buildingGate	0x00000003
buildingBridgeGate	0x00000004
buildingTower	0x00000005
buildingWall	0x00000006
buildingCopula	0x00000007

3. trapType

Nazwa stałej	**Wartość w HEX**
NULL	0x00000000
typeHoldTrap	0x00000001
typeDamageTrap	0x00000002
typeHoldTrapOnce	0x00000003

4. wasteSize

Nazwa stałej	**Wartość w HEX**
smallFlyingWaste	0x00000000
mediumFlyingWaste	0x00000001
bigFlyingWaste	0x00000002

5. groupTemplateNum

Nazwa stałej	**Wartość w HEX**
groupSword	0x0000005B
groupDrop	0x0000005C
groupMag	0x0000005D
groupSpecial	0x0000005E
groupAnimal	0x0000005F
groupMulti	0x00000060
groupBuilding	0x00000061

6. positionType

Nazwa stałej	**Wartość w HEX**
positionStartingPoint	0x00000000
positionMarkPoint	0x00000001
positionProductionPoint	0x00000002

7. explosionFlags

Nazwa stałej	**Wartość w HEX**
noEarthquake	0x00000000
smallEarthquake	0x00000001
mediumEarthquake	0x00000002
bigEarthquake	0x00000003

8. raceFlags

Nazwa stałej	**Wartość w HEX**
eCanBePlayedInSkirmish	0x00000001

9. standType

Nazwa stałej	**Wartość w HEX**
standNone	0x00000000
standAccurate	0x00000001
standVertical	0x00000002
standCoarsly	0x00000003
standSwing	0x00000004
standWater	0x00000005
standMoveDownSmall	0x00000010
standMoveDownMedium	0x00000020
standMoveDownBig	0x00000030
standTurn	0x00000040
standTurnToFlat	0x00000080
standMoveSmall	0x00000100
standMoveMedium	0x00000200
standMoveBig	0x00000300
standWaterPlant	0x00000345
standTree	0x00000340
standTreeFall	0x00000380
standRock	0x00000150
standStone	0x00000140
standFish	0x00000045

10. experienceExplosionPos

Nazwa stałej	**Wartość w HEX**
expPosZero	0x00000000

11. equipmentFlags

Nazwa stałej	Wartość w HEX
shieldArmourType	0x00000001
maxArmourType	0x00000002

12. hitType

Nazwa stałej	Wartość w HEX
singleHit	0x00000000
multiHit	0x00000001

13. type (missile)

Nazwa stałej	Wartość w HEX
NULL	0x00000000
missileSword	0x00000001
missileInvisible	0x00000002
missileCannon	0x00000003
missileDropBomb	0x00000004
missileBomb	0x00000005
missileElectric	0x00000006
missileLightning	0x00000007
missileMeteor	0x00000008

14. race

Nazwa stałej	Wartość w HEX
NEUTRAL	0x00000000
POL	0x00000001

9. Dodawanie nowego obiektu

W celu dodania nowego obiektu, należy skopiować strukturę (struct) reprezentującą obiekt, na którym będziemy się wzorować i wkleić np. od razu po obiekcie, który kopiujemy.

```
// KOPIUJEMY STRUKTURĘ WOLF PRZEDSTAWIONĄ PONIŻEJ...
struct WOLF
{
 string obj_name = "WOLF";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};

// I PRZYKŁADOWO WKLEJAMY JĄ TU...

struct DOG
{
 string obj_name = "DOG";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1,1, ... ,0,0,0,0,0,0,0,1,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\LABRADOR#Dog.tex";
 string lowResMesh = "";
 string lowRes2Mesh = "";
 string interfaceMesh = "";

 ...

 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
```

```
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};
```

Po dokonaniu opracji mamy następujący pogląd:

```
struct WOLF
{
 string obj_name = "WOLF";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};

struct WOLF
{
 string obj_name = "WOLF";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
```

```
 uint32_t $childConversionExplosionID = 4294967295;
};

struct DOG
{
 string obj_name = "DOG";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1,1, ... ,0,0,0,0,0,0,0,1,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\LABRADOR#Dog.tex";
 string lowResMesh = "";
 string lowRes2Mesh = "";
 string interfaceMesh = "";

 ...

 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};
```

Teraz należy zmienić nazwę struktury i nazwę obiektu. Poniżej przedstawiono lokalizacje pól do zmiany:

```
struct WOLF // <--- Tutaj jest nazwa struktury.
{
 string obj_name = "WOLF"; // <--- Tutaj jest nazwa obiektu.
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
```

```
 uint32_t $childConversionExplosionID = 4294967295;
};
```

Po dokonaniu zmian, kod struktury wygląda następująco:

```
struct SAINT_WOLF_TEST
{
 string obj_name = "SAINT_WOLF_TEST";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};
```

Parametry obiektu można dowolnie modyfikować. Plik sekcji można zapisać, a następnie skompilować plik PAR. Warto pamiętać również o dodaniu nowego obiektu do pliku `EditorDef.txt` w celu sprawdzenia go w edytorze map.

10. Dodawanie nowej sekcji

Aby dodać nową sekcję, należy skopiować plik innej, na której będziemy się wzorować i wkleić go w tym samym katalogu. Następnie należy zmienić nazwę pliku, tak aby nie kolidowała z innymi plikami.

Przykład:

- plik bazowy = `section_8_NWOJ.cpp`
- nowy plik sekcji = `section_8_NS_0_NOWA_SEKCJA.cpp`

Wnętrze pliku sekcji:

W pliku sekcji powinniśmy zauważyć strukturę nagłówka:

```
struct Section_8_NWOJ_Header
{
 32bituniquekey race = POL;
 uint32_t flag_ = 1;
 uint32_t number_of_objects = count();
};
```

Nazwa struktury powinna mieć nazwę taką samą jak nazwa pliku, ale na samym końcu powinien być dopisek `_Header` . Zmodyfikowany nagłówek prezentuje się nastepująco:

```
struct Section_8_NS_0_NOWA_SEKCJA_Header
{
 32bituniquekey race = POL;
 uint32_t flag_ = 1;
 uint32_t number_of_objects = count();
};
```

Wszystkie obiekty poniżej nagłówka należy usunąć. Aby dodać nowe, wklej do pliku reprezentujące je struktury. Dodawanie nowych obiektów zostało wytłumaczone w rozdziale poprzednim.

Jeśli dodano już nową sekcję, to należy dopisać jej nazwę również do pliku **section_order.txt**, w którym zapisana jest kolejność kompilacji sekcji. Najlepiej jest umieścić nazwę pliku nowej sekcji, od razu po nazwie pliku na którym bazowaliśmy. Nazwa w liście powinna być taka sama jak nazwa nowego pliku sekcji ale bez formatu.

Przykład listy przed dodaniem nowej sekcji:

```
...
section_6_NWLOCZNIK
section_7_FOOTMAN
section_8_NWOJ
section_9_DWARF_FOOTMAN_1
section_10_KNIGHT
...
```

Przykład listy po dodaniu nowej sekcji:

```
...
section_6_NWLOCZNIK
section_7_FOOTMAN
section_8_NWOJ
section_8_NS_0_NOWA_SEKCJA
```

```
section_9_DWARF_FOOTMAN_1
section_10_KNIGHT
...
```

11. Kompilacja plików .cpp do postaci .par przez PARim

PARim jest to program, a dokładnie kompilator służacy do kompilacji całego katalogu z plikami sekcji `.cpp` do postaci `.par`.

Jak używać?:

1. Włączamy program.
2. Wpisujemy nazwe katalogu.
3. Klikamy enter i za chwilkę powinno się zrobić.
4. Wyłączenie okna konsoli sygnalizuje zakończenie pracy programu.

Program działa również w trybie ARGC&ARGV. Przykład użycia:

```
PARim.exe <nazwa katalogu wejściowego>
```

Szybki Import:

Aby w szybki sposób kompilować katalog z plikami `.cpp`, należy korzystać z `IMPORT.ps1` lub `IMPORT.bat`. W celu modyfikacji argumentów/konfiguracji należy zedytować plik `.bat/.ps1` przez notatnik.

Przykład pliku .bat:

```
PARim.exe <nazwa katalogu wejściowego>
```

Przykład pliku .ps1:

```
.\PARim.exe .\<nazwa katalogu wejściowego>
```